ALEXANDRIA UNIVERSITY
FACULTY OF ENGINEERING
DEPARTMENT OF ENGINEERING MATHEMATICS AND PHYSICS

COMPUTER ALGEBRA AND ITS APPLICATIONS

A thesis submitted to the
Department of Engineering Mathematics and Physics
in partial fulfillment of the requirements for the degree of
Master of Science
in Engineering Mathematics

By

Hazem Mohamed El-Alfy

B.Sc., Faculty of Engineering, Alexandria University, 1997

Supervised by:

Prof. Dr. Abdel-Karim Aboul-Hassan

Dr. Mohamed Sayed

August 2001

We certify that we have read this thesis and that, in our opinion, it is fully adequate, in scope and quality, as a dissertation for the degree of Master of Science.

Judgment Committee

Prof. Dr. Ahmed A. Belal

Dept. of Computer Science and Automatic Control

Faculty of Engineering, Alexandria University.

Prof. Dr. Ahmed M. A. ElSayed

Dept. of Mathematics

Faculty of Science, Alexandria University.

Prof. Dr. Abdel-Karim Aboul-Hassan

Dept. of Engineering Mathematics and Physics

Faculty of Engineering, Alexandria University.

For the Faculty Council

Prof. Dr. Hassan A. Farag

Vice Dean for Graduate Studies and Research

Faculty of Engineering, Alexandria University.

Supervisors:

Prof. Dr. Abdel-Karim Aboul-Hassan

Department of Engineering Mathematics and Physics,
Faculty of Engineering,
Alexandria University.

Dr. Mohamed Sayed

Department of Engineering Mathematics and Physics,
Faculty of Engineering,
Alexandria University.

ACKNOWLEDGMENT

It is a pleasure for me to acknowledge the many people who have contributed to this work. Listing all of them would require dozens of pages, and may always miss some others.

First and before all, ***glory be to Allah***, the first teacher to mankind.

I would wish to thank my supervisors, ***Prof. Dr. Abdel-Karim Aboul-Hassan*** and ***Dr. Mohamed Sayed*** for their considerable help in my research; something I can't deny.

I would like to extend a special note of thanks to ***Prof. Franz Winkler*** at the Research Institute for Symbolic Computation at Linz University, Austria. He was very helpful in providing me with several references.

Finally, I gratefully acknowledge the support, encouragement and patience of my parents.

Abstract

In the recent decades, it has been more and more realized that computers are of enormous importance for numerical computations. However, these powerful general-purpose machines can also be used for transforming, combining and computing symbolic algebraic expressions. In other words, computers can *not only* deal with numbers, but also with abstract symbols representing mathematical formulas. This fact has been realized much later and is only now gaining acceptance among mathematicians and engineers. [Franz Winkler, 1996].

Computer Algebra is that field of computer science and mathematics, where computation is performed on symbols representing mathematical objects rather than their numeric values.

This thesis attempts to present a definition of computer algebra by means of a survey of its main topics, together with its major application areas. The survey includes necessary algebraic basics and fundamental algorithms, essential in most computer algebra problems, together with some problems that rely heavily on these algorithms. The set of applications, presented from a range of fields of engineering and science, although very short, indicates the applied nature of computer algebra systems.

A recent research area, central in most computer algebra software packages and in geometric modeling, is the implicitization problem. Curves and surfaces are naturally reperesented either parametrically or implicitly. Both forms are important and have their uses, but many design systems start from parametric representations. Implicitization is the process of converting curevs and surfaces from parametric form into implicit form.

We have surveyed the problem of implicitization and investigated its currently available methods. Algorithms for such methods have been devised, implemented and tested for practical examples. In addition, a new method has been devised for curves for which a direct method is not available. The new method has been called *near implicitization* since it relies on an approximation of the input problem. Several variants of the method try to compromise between accuracy and complexity of the designed algorithms.

The problem of implicitization is an active topic where research is still taking place. Examples of further research points are included in the conclusion.

جامعة الإســـــــــــــــكندرية
كلـــــــــــــــية الهندســـــــــــــــــــة
قسم الرياضيـــــــــات والفيزيـــــــــــاء الهندسيـــــــــــة

جبر الحاسبـــــــــــــات وتطبيقـــاته

رسالـــة مقدمـــة إلـــى
قسم الرياضيـــات والفيزيـــاء الهندسيـــة
كإنجاز جزئي من متطلبـــات الحصـــول على درجـــة

الماجستيـــر

في الرياضيات الهندسية

مقدمـــة مـــن

حازم محمد الألفي

بكالوريوس من كلية الهندسة – جامعة الإسكندرية

لجنـــة الإشـــراف

أ. د. عبد الكريم أبو الحسن محمد

د. محمد سيد محمد

أغسطس 2001

ملـــخص الرســـالة

تزايد استخدام الحاسبات في حل المسائل الرياضية غير العددية خلال الـــسنوات الماضـــية، نتيجة إدراك علماء الرياضيات والمهندسين أن تلك الأجهزة قادرة على التعامل مع هذه النوعية مـــن المسائل بكفاءة لا تقل عن تعاملها مع الحسابات العددية. وقد أصبح اســـتخدام الحاســـبات ضـــروريا وحيويا في التعامل مع المسائل غير العددية، نظرا لأن حل مثل هذه المسائل يدويا يعتبر عملا مضنيا بالإضافة إلى أنه يستهلك وقتا كبيرا كما أنه معرض للأخطاء.

ويعتبر "جبر الحاسبات" – موضوع الرسالة – هو ذلك الفرع مـــن الرياضـــيات المخـــتص بدراسة المسائل غير العددية وتحليلها وتطبيقها على أجهزة الحاسب ومن ثم حلها بواسطتها.

وتتناول هذه الرسالة موضوع جبر الحاسبات عن طريق مراجعة مكثفة لأساسيات ذلك العلم، مع اختيار أحد موضوعاته الرئيسية (تضمين الدوال) ودراسته بشكل أكثر توسعا، ثم بيان تطبيق أحد حزم برامج جبر الحاسبات في مجال المعادلات التفاضلية العادية، وذلك على النحو التالي.

- ❖ الباب الأول: مقدمة عن جبر الحاسبات مع تعريف مختصر له.
- ❖ الباب الثاني: مراجعة لأساسيات جبر الحاسب مع ذكر التعاريف المختلفة له ومن ثم وصـــف جبر الحاسب ومناقشة تعريفه لجبر الخوارزميات. كما يحتوي الباب علـــى مراجعـــة لخوارزميـــات أساسية لحل العديد من مسائل جبر الحاسب، بالإضافة إلى مجموعة من تطبيقات هذا العلم.
- ❖ الباب الثالث: دراسة موضوع تضمين الدوال، أي تحويلها من الصورة البارامترية إلى الصورة الضمنية، والذي يعتبر أحد الموضوعات الأساسية لحزم برامج جبر الحاسب. وقد تمت دراسة وتحليل الطرق والوسائل المختلفة في هذا الموضوع، مع اقتراح طرق جديدة تقريبية للـــدوال التـــي لايمكـــن التعامل معها بما هو متاح من وسائل.
- ❖ الباب الرابع: تصميم خوارزميات لما تم عرضه في الباب السابق من وسائل التضمين، ومـــن ثم برمجتها وتطبيقها على الحاسب.
- ❖ الباب الخامس: خلاصة النتائج ونقاط بحث مقترحة.

Table of Contents

Chapter 1: INTRODUCTION

In this thesis, we attempt first to answer the question: what is Computer Algebra? The answer is extracted from the existing literature [1] [2] [4] [5]. Secondly, a review of some computer algebra basics is given. Those reviewed basics are elements of abstract algebra and three mathematical objects fundamentally needed, namely, Euclidean algorithm, Chinese remainder algorithm and computation of Gröbner bases algorithm. Such objects are needed and used to build computer algebra systems (CAS).

Since polynomial systems have a wide range of applications, in algebraic geometry, automated geometric theorem proving, computer aided design (CAD) and computer aided manufacturing (CAM) systems, computer graphics, virtual reality and other fields, we picked up the problem of implicitization to investigate.

Algorithms of implicitization are given when not provided in the literature. Other algorithms are suggested as well. These are proved and implemented using the software package Matlab, with some sample runs to furnish their applicability.

1.1 WHAT IS COMPUTER ALGEBRA?

In this section, we attempt to define the term *computer algebra*. It is opposed to numerical analysis and its uses and limitations are presented.

1.1.1 An attempt at a definition

Computer Algebra, also known as *Symbolic Computation*, *Algebraic Manipulation* or *Formula Manipulation*, is a scientific field in computer science and mathematics, where computation is performed on symbols representing mathematical objects rather than their numeric values.

Franz Winkler [2] restates R. Loos' definition made in his introduction to Buchberger et al. (1983): "Computer algebra is that part of computer science which designs, analyzes, implements, and applies algebraic algorithms".

Gathen and Gerhard [1] present the following definition: "Computer Algebra is a more recent area of computer science where mathematical tools and computer software are developed for the *exact* solutions of equations".

In [5], it is stated that the principal function of a Computer Algebra system is the rearrangement of data so that the relevant information in it is easier to extract. In the implementation of the Computer Algebra System designed in this reference, it is stressed on the fact that a symbolic solution of a problem indicates how the solution was reached in terms of the symbols constituting the problem, instead of its numeric value.

1.1.2 Algebraic calculations

Algebraic computation or symbol manipulation or computer algebra is the field of scientific computation which develops, analyzes, implements and uses algebraic algorithms.

Typical features of algebraic computation are:

- Computation with arbitrary precision numbers--no rounding.
- Computation with symbols and variables (e.g. *x*, *y*).
- Computation with functions (e.g. *sin*, *cos*).
- Manipulation of formulas.
- Symbolic operations (differentiation, integration, factorization, etc.)

As opposed to numerical computation, symbolic computation has the following characteristics:

Numerical Computation	**Symbolic Computation**
Computations with numbers only e.g.: $2 + 3 = 5$	Computing in algebraic structures (polynomial rings, finite fields, etc…) in addition to computing with numbers (algebraic number domains). e.g.: $x + 2x = 3x$, $\frac{1}{6}+\frac{1}{4}=\frac{5}{12}$.
Approximate results e.g.: 0.86602…, $cos(3.1415)$= -0.999…	Exact representation of results e.g.: $\frac{\sqrt{3}}{2}$, $cos(\pi) = -1$
Numeric results e.g.: $\int_0^{1/2} \frac{x}{x^2-1}dx = 0.1438$	Symbolic results e.g.: $\int \frac{x}{x^2-a}dx = \frac{1}{2}\ln\left\|x^2-a\right\|$
Numeric evaluation e.g.: a^2-b^2 = ?? (undefined a,b)	Symbolic simplification e.g.: a^2-b^2 = (a-b)(a+b)

1.1.3 Why do we need algebraic calculations?

- There exist algorithms which cannot easily be performed manually with pencil and paper (e.g., factorization).
- Algebraic solutions are usually more compact than a set of numerical solutions; algebraic solution gives more direct information about the relationship between the variables than figures.
- From algebraic expressions, one can deduce how changes in the parameters affect the result of the computation. As a consequence one can build decision algorithms on computer algebra, e.g. for the factorizability of polynomials, the equivalence of algebraic expressions, the solvability of certain differential equation problems, the parametrizability of algebraic curves
- Algebraic solutions are always exact, numerical solutions will normally be approximates; this can arise from rounding and truncation errors, further errors can creep in when the user interpolates data given in tabular form.
- Computer algebra can save both time and effort in solving a wide range of problems; much larger problems can be investigated than by using traditional methods.
- Computer algebra reduces the need for tables of functions, series and integrals; thus avoids errors when using such tables.
- Traditional teaching of applied mathematics has to involve much time in teaching techniques of solution; computer algebra systems tend to produce solutions quickly and without errors, so they enable more time to be devoted to studying the properties of the solution. Nowadays, there exist many attempts of using computer algebra systems (CAS) in computer aided learning (CAL).
- Using of computer algebra allows also very effective construction of numerical algorithms and their semiautomatic programming by code generation, the effectiveness of the work and reliability of the results can be strongly increased.

1.1.4 Limitations of Computer Algebra

The previous sections present computer algebra much more superior than numerical analysis. This is not the case, however. Symbolic computation has its limitations and thus can't supersede numerical solutions. Some examples of such cases are illustrated hereunder.

1. *High complexity problems*: There are problems for which symbolic computation of an exact solution is prohibitively expensive, e.g., the problem of finding the solutions to a system of algebraic equations:

$$f_1(x_1, \ldots, x_n) = 0,$$

$$\ldots$$

$$f_m(x_1, \ldots, x_n) = 0.$$

where the f_i's are polynomials with, say, integral coefficients. This is the most natural application of Gröbner bases, and they were originally invented for this. Unfortunately, computing Gröbner bases is at least exponential in the number of variables n. In fact, Mayr proved in 1992 that the Ideal Membership problem (a problem that is reducible to computing a Gröbner basis in polynomial time) is EXPonential-SPACE complete, thus EXPTIME [1].

2. *Problems with no symbolic solution*: There are many problems that don't have closed form or exact symbolic solutions, e.g. differential equations. Symbolic solutions are available for only a small class of simple differential equations, such as integration problems and homogeneous linear differential equations. However, not much is known concerning symbolic algorithms for other types of differential equations, in particular partial differential equations [2].

3. *Cumbersome results*: Exact solutions may be too complex for the user of Computer Algebra System to assess, even for an order of magnitude. This is not a limitation in the usual sense of the word, but rather an inconvenience of working *only* with exact results.

1.2 ALGORITHMIC ALGEBRA

A closely related term to computer algebra is algorithmic algebra. It can be briefly defined as the science of developing algorithms for the solution of various problems of algebra. In this section, this term is defined in more details, and areas where it is applied are presented.

1.2.1 An overview of active topics in algebraic computation

In the last years, there has been dramatic progress in all areas of algebraic computation. Many working mathematicians have access to and actually use algebraic software systems for

their research. No end of this development is in sight. As reported in the proceedings of the Vienna Conference for Algebraic Computation [4], some of the active topics in this area cover the following subjects:

- Algorithmic methods for indefinite integration.
- Solutions of systems of algebraic equations.
- Factorization of polynomials.
- Geometry of algebraic curves and surfaces: implicitization and parametrization.
- Algorithmic methods for computing closed form solutions of differential equations (this topic is in its infancy; many differential equations are still unsolved symbolically).

Most of these topics rely heavily on *algorithmic algebra* defined hereunder.

1.2.2 Algorithmic Algebra

In his textbook, *Algorithmic Algebra*, B. Mishra proposes the following definition of algorithmic algebra [3]: “[People] who have been instrumental in the rapid growth of "algorithmic algebra" (…) are working toward a common goal, namely that of developing an algorithmic foundation for various problems in algebra”.

In the same reference, he discriminates among, somewhat overlapping, four areas of interest in this subject:

- Constructive algebra, “under which an object exists only if we can construct it” as opposed to the classical interpretation where “an object exists if its non existence is contradictory”.
- Computational algebra, where one is concerned with the feasibility, complexity and implementation of algorithms.
- Symbolic computation, where the researcher’s goal is to develop algorithms that are efficient in practice.
- Applications, where algorithms are developed to serve specific application areas of symbolic computation.

Most computer algebra algorithms require one, if not more, of the following results from algorithmic algebra:

- Algebraic domains and polynomials.
- Greatest common divisors: the Euclidean algorithm.
- Modular computation: the Chinese remainder algorithm.
- Gröbner bases

- Resultants and subresultants.

1.3 ORGANIZATION OF THE THESIS

This work proceeds as follows.

Chapter 1, *Introduction*, is an introduction as its name indicates. An attempt to answer the question: "what is computer algebra?" is given. A definition of a closely related term, *algorithmic algebra*, is also given.

Chapter 2, *Computer Algebra tools and applications*, is a survey of important results from algorithmic algebra, used in most computer algebra problems. It starts with an overview of algebraic systems, with a special emphasis on polynomials, which are required in most computer algebra algorithms. Then, three fundamental algorithms are presented: the Euclidean algorithm, the Chinese remainder algorithm and the Gröbner bases computation algorithm. Specific applications presented after the discussion of each such computer algebra fundamental algorithm cover a wide range of essential problems present in almost every computer algebra system. Additional general applications supplemented at the end of the chapter cover an even wider range of applications.

Chapter 3, *the implicitization problem,* presents the problem and overviews more results from algebra required for this specific problem. In addition, it reviews some implicitization methods (for rational and trigonometric curves) and suggests newly devised ones (near implicitization variants).

Our work, started at the end of chapter 3, continues in chapter 4, *Implementation*. Algorithms are given for all methods reviewed and suggested in the preceding chapter. These algorithms are implemented using the software package Matlab. Several sample runs furnish the applicability of these algorithms on practical examples.

Finally, chapter 5 concludes the thesis and suggests further research points.

Chapter 2: COMPUTER ALGEBRA TOOLS AND APPLICATIONS

2.1 ALGEBRAIC PRELIMINARIES

This section presents some basic notions used throughout the thesis. For conciseness, only definitions and basic facts are introduced, illustrated with examples. Further details and proofs can be referred to in the literature [3] [6] [7] [8].

Operations on algebraic structures, presented in this section are essential in basic computer algebra algorithms, such as the Euclidean algorithm and the Chinese remainder algorithm presented later. These algorithms, in turn, are the basis of many computer algebra applications, such as the symbolic factorization of polynomials, which serves in the exact solution of polynomial equations, partial fraction decomposition, integration of rational functions and many other applications.

Almost every computer algebra algorithm requires dealing with univariate and multivariate polynomials. These are also overviewed in the present section.

2.1.1 Sets, functions and operations [6]

Sets

A *set* is any identifiable collection (finite or infinite) of objects of any sort. The objects which make up a particular set are called its *elements* or *members*. Sets can be countable or uncountable.

The C*artesian product* of two sets X and Y is the set of all *ordered pairs* (*tuples*) whose first entry belongs to X and second entry belongs to Y.

$$X \times Y = \{ (x,y) \mid x \in X, y \in Y\}$$

Functions

A *function f* from X into Y is a subset of $X \times Y$ such that:

$$\forall x \in X \text{ there is one and only one } y \in Y \text{ such that } (x,y) \in f.$$

X is called the *domain* of *f*. Y is called the *range* of *f*.

Operations

A *binary operation* * on X is a function of $X \times X$ into X.

Given an operation *, the notation $a * b$ is used for $*(a,b)$.

Closure: for an operation * defined on X, and a subset $A \subseteq X$, it is said that A is *closed* under the operation * if: $a * b \in A$, $\forall a,b \in A$.

2.1.2 Algebraic Systems

An algebraic system is a set S together with operations $*_1, *_2, \ldots, *_k$ defined on it (i.e. S is closed under these operations. According to the number of operations, and to certain properties which X and $*_i$ possess, different types of algebraic systems are defined.

Identity and inverse

Let * be an operation on S, and consider the algebraic system $\{S,*\}$.

$e \in S$ is an *identity element* of the system $\{S,*\}$if:

$$\forall x \in S,\ e * x = x * e = x.$$

If e is an identity of $\{S,*\}$, then $x^{-1} \in S$ is called an *inverse* of $x \in S$, if:

$$x^{-1} * x = x * x^{-1} = e.$$

Since most of the systems dealt with don't have more than two operations, it is common to use the symbols "+" and "$\cdot$" for $*_1$ and $*_2$. The "$\cdot$" may also be omitted when it is implicitly understood to be the operation in the context. Usual notations of the identity elements (if any) with respect to "+" and "$\cdot$" can be "0" and "1" respectively. Finally, the inverses of x w.r.t. the previous operations will be "$-x$" and "x^{-1}" respectively.

2.1.2.1 ***Groups*** [3]

Definition: A *group* G is a nonempty set with a binary operation "$\cdot$" such that:

1. G is closed under the operation "$\cdot$" :

$$\forall\ a,b \in G,\quad a \cdot b \in G.$$

2. The operation "$\cdot$" is associative:

$$\forall\ a,b,c \in G,\quad (a \cdot b) \cdot c = a \cdot (b \cdot c)$$

3. G possesses an identity element:

$$\exists\ e \in G \text{ such that } \forall\ a \in G,\ e \cdot a = a \cdot e = a.$$

4. Every element of G has an inverse:

$$\forall\ a \in G, \exists\ a^{-1} \in G \text{ such that: } a^{-1} \cdot a = e.$$

If $\{G, \cdot\}$ doesn't possess an identity element, (i.e. only the first two conditions are satisfied), the system is called a *semigroup*. If condition 3 is also satisfied, but not every element (or none) is invertible, the system is called a *monoid*.

A group is called an *Abelian group* if its operation is commutative, i.e.:

$$\forall\ a,b \in G,\ a \cdot b = b \cdot a.$$

Examples: [7]

1. The set of $n \times n$ matrices with real entries forms an Abelian group under addition, and a monoid under multiplication (the identity matrix being the identity element, but a singular matrix is not invertible).
2. Integers under multiplication form a semigroup.
3. The 3rd roots of the unit: 1, ω and ω^2 form a finite Abelian group under multiplication. This group is also known as a *cyclic group*, where each element is a power of one particular element of the group (Note that $\omega^3 = 1$).

Definitions:

- A *subgroup* H of a group G is a non empty subset of G which is itself a group satisfying the four conditions of a group definition. Otherwise stated:

A subset of a group $H \subseteq G$ is a subgroup if and only if: $\forall\ a,b \in H,\ a \cdot b^{-1} \in H$.

- The *product* of two subsets H_1 and H_2 of a group G is:

$$H_1H_2 = \{h_1 \cdot h_2 \mid h_1 \in H_1 \text{ and } h_2 \in H_2\}.$$

In what follows, it doesn't matter to omit the "·" sign.

- A *left coset* of a subgroup H of a group G is defined as:

$$aH = \{ah \mid \forall h \in H\}, \text{ where } a \text{ is any element of } G.$$

- A *right coset* is similarly defined as: $Ha = \{ha \mid \forall h \in H\}$.

In particular, if $a \in H$ then $aH = Ha = H$.

According to *Lagrange theorem*, the family of (left or right) cosets of a subgroup H of G constitutes a *partition* of G. For finite groups, the number of distinct cosets of H is $|G|/|H|$, where the $|\cdot|$ denotes the cardinality of sets. [7]

- A subgroup H of a group G is called a *normal* (or *self-conjugate*) *subgroup* of G if:

$$\forall a \in G,\ aH = Ha.$$

As a result, every subgroup of an Abelian group is a normal subgroup.

- The *quotient group* of a group G with respect to a *normal* subgroup H is the set:

$$G/H = \{aH \mid a \in G\}, \text{ under the operation } (xH)\cdot(yH) = x\cdot yH.$$

This is a set of left cosets, and one can easily prove it is indeed a group, or read the proof in either of [3] or [7].

Example:

Consider the Abelian group $(Z, +, 0)$; i.e. the group of integers under addition, with identity 0. The multiples of a positive integer m form a subgroup. The cosets are of the form:

$$i + mZ = \{i + km \mid k \in Z, 0 \le i < m \}$$

Two elements a and b of Z (integers) are *congruent modulo m* (belong to the same coset) if their difference $(a\text{-}b)$ is divisible by m. This is denoted as: $a \equiv b \bmod m$.

*2.1.2.2 **Rings and Ideals*** [3]

A *ring* R is a set with two binary operations "+" and "·" such that:

1. R is an Abelian group with respect to addition. It has an identity element 0 and every $x \in R$ has an additive inverse $-x$.
2. R is a semigroup with respect to multiplication.
3. Multiplication is *distributive* over addition, i.e.:

$$\forall\ a,b,c \in R,\ \ a \cdot (b + c) = a \cdot b + a \cdot c,\ \text{ and }\ (a + b) \cdot c = a \cdot c + b \cdot c.$$

If in addition, R has an identity "1" with respect to "·", then R is a *ring with identity*.

If the multiplication operation is commutative, R is a *commutative ring*.

The group $(R, +, 0)$ is known as the *additive group* over the ring R.

In what follows, the main concern will be for commutative rings with identity, thus they can be called just rings.

Examples:

1. The set of integers Z, the rational numbers Q and the real numbers R are examples of rings with respect to the ordinary addition and multiplication
2. The set of even numbers forms a ring without identity.
3. The set of integers modulo m forms a *finite ring*. $[i]_m = \{i + mk \mid k \in Z\}$.

- A *subring R'* of a ring R is a nonempty subset of R which is itself a ring satisfying its three conditions mentioned above.

- An *ideal* is a subset of a ring $I \subseteq R$, which satisfies the following conditions:

1. I is an additive subgroup of the additive group of R:

$$\forall\ a,b \in I,\ (a\text{-}b) \in I.$$

2. I is closed under multiplication with ring elements:

$$\forall\ (a \in R, b \in I),\ \ ab \in I.$$

The ideals $\{0\}$ and R are called *improper ideals* of R. All other ideals are *proper*.

Note that any ideal of R is also a subring of R. The converse is not true.

Example:

The set of integers that are divisible by 12 form the ideal $I = \{12r \mid r \in Z\}$.

- A set $B \subseteq R$ *generates* the ideal I if $I = \{r_1b_1 + \ldots + r_nb_n \mid r_i \in R, b_i \in B\}$.

We say that B is a *basis* or a *generating set* of the ideal I, and denote it as:

$$I = \langle b_1, \ldots, b_n \rangle$$

- A *Noetherian ring* is one in which any ideal has a finite basis.

Definitions:

- An element $x \in R$ is called a *zero divisor* if there exists $y \neq 0$ in R such that $xy = 0$. (x and y need not be distinct).
- An element $x \in R$ is called a *unit* if there exists $y \in R$ such that $xy = 1$. (The element x is thus *invertible*).
- A ring R is called an *integral domain* if it has no nonzero zero divisors. In an integral domain, the *cancellation law* holds: if $a \neq 0$ and $ab = ac$ then $b = c$.

Thus, in an integral domain, $R \setminus \{0\}$ forms a semigroup w.r.t. multiplication.

- A nonzero nonunit p in an integral domain R is *reducible* if there are nonunits $a, b \in R$ such that $p = ab$, otherwise, p is irreducible. Units are neither reducible nor irreducible.
- A ring R is called a *Unique Factorization Domain* (*UFD*) if every nonzero nonunit $a \in R$ can be written as a product of irreducible elements in a unique way up to reordering and multiplication by units. [1]

Example:

Z is a UFD, and $12 = 2 \cdot 2 \cdot 3 = (-3) \cdot 2 \cdot (-2)$ are both decompositions of 12 into irreducibles. Note that the units of Z are 1 and -1, the only integer numbers whose inverses are integers.

- An integral domain R together with a function $d: R \to N \cup \{-\infty\}$ is called a *Euclidean domain* if for all $a, b \in R$, with $b \neq 0$, a can be divided by b with remainder, i.e.:

$$\forall\ a,b \in R,\ b \neq 0,\ \exists\ q,r \in R \text{ such that } a = qb + r \text{ and } d(r) < d(b).$$

q is the quotient and r the remainder. [1]

2.1.2.3 ***Fields and Modules*** [3]

- A *field* F is an integral domain in which every nonzero element is a unit (i.e. invertible). Thus a field consists of two Abelian groups with respect to each of its operations: the additive group $(F, +, 0)$ and multiplicative group $(F \setminus \{0\}, \cdot, 1)$.

Examples: Familiar examples of fields are the field of rational numbers Q, the field of real numbers R and the field of complex numbers C. We have: $Q \subset R \subset C$.

- A *subfield* E of a field F is a subring which itself is a field. We say E is a *subfield* of F and F is an *extension field* of E.
- If R is an integral domain, then $F = \{a/b \mid a,b \in R,\ b \neq 0\}$ is called the *field of fractions* or *quotient field* of R. The field operations can be defined on representatives of the quotient field in the following way:

$$\frac{a}{b}+\frac{c}{d}=\frac{ad+bc}{bd}\ ,\ \ \frac{a}{b}\cdot\frac{c}{d}=\frac{ac}{bd}\ ,\ \ -\frac{a}{b}=\frac{-a}{b}\ ,\ \ \left(\frac{a}{b}\right)^{-1}=\frac{b}{a}$$

Modular arithmetic and finite fields: [2]

As already introduced in the example on group cosets, but stated here in other words, every integer m generates an ideal $\langle m \rangle$ in Z. Two integers a, b are *congruent modulo m* if and only if m divides $(a - b)$ or $(a - b) \in \langle m \rangle$. This is also written as $a \equiv b \bmod m$.

The congruence relation is an equivalence relation which partitions the set of integers Z into equivalence classes called *residue classes* modulo m. The residue classes can be represented as: $[i]_m = \{i + mk \mid k \in Z,\ 0 \leq i < m\}$, which can be verbosely stated as the set of integers with remainder i when divided by m.

Now, the set of residue classes forms a finite commutative ring with respect to addition and multiplication modulo m illustrated in the example in this section. It is denoted as Z_m.

Finally, Z_m will be a *finite field* if and only if m is a prime number. In general, a finite field need not have prime cardinality. However, every finite field has cardinality p^n for some prime p. This field is usually called the *Galois field* of order p^n.

Example - (Modular arithmetic): [6]

Understanding how to compute on the set Z_m of congruence classes modulo m is crucial for computer algebra algorithms. This is called *modular arithmetic*.

Consider for example Z_4. $[0]_4$ denotes the set of integers divisible by 4. $[1]_4$ denotes those integers, such as 9 and -3, which have remainder 1 when divided by 4, and so on for the classes $[2]_4$ and $[3]_4$. For simplicity, the brackets and subscripts can be omitted. Thus Z_4 has the following addition and multiplication tables.

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

·	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	0	2
3	0	3	2	1

Modules: [3]

Given a ring R, an Abelian group G, and a mapping

$$\mu : R \times G \to G$$
$$\langle r, g \rangle \mapsto rg$$

- We say G is an *R-module* if, for all r, $s \in R$ and x, $y \in G$, the following axioms are satisfied:

$$r(x + y) = rx + ry,$$
$$(r + s)x = rx + sx,$$
$$(rs)x = r(sx),$$
$$1x = x.$$

Thus, an R-module is an additive Abelian group G on which the ring R acts linearly.

- If R is a field F, then an F-module is said to be an *F-vector space*.

This section on algebraic systems is concluded with the chart that follows [8], which attempts to put into perspective most of the systems considered, together with brief properties they have. Note the convention used here, in that a structure "inherits" the properties of its "parent", and only those properties specific to it are indicated.

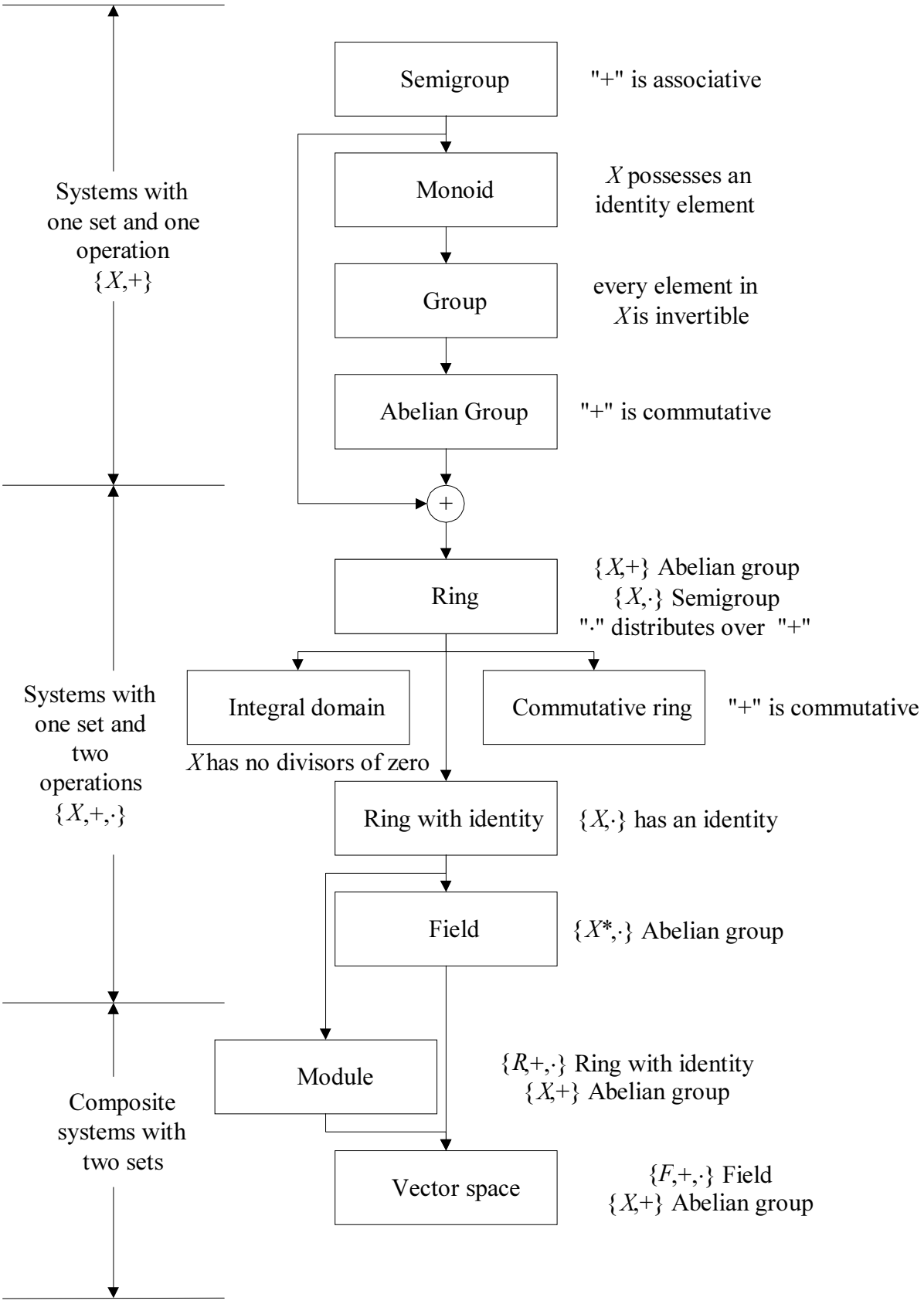


Figure 2-1 Relationship between basic algebraic systems

2.1.2.4 ***Homomorphism*** [3]

- Let $(R, +_R, \cdot_R, 0_R, 1_R)$ and $(S, +_S, \cdot_S, 0_S, 1_S)$ be two rings (commutative with identity). A *ring homomorphism* h is a map from R to S satisfying the conditions:

$$h(0_R) = 0_S\ ,\ h(1_R) = 1_S\ ,\ h(a +_R b) = h(a) +_S h(b),\ h(a \cdot_R b) = h(a) \cdot_S h(b).$$

- A homomorphism h is called an *isomorphism* from R into S iff h is bijective (i.e. 1-1 and onto).In this case we say that R and S are *isomorphic* and denote it by $R \cong S$.

- The *kernel* of a homomorphism h: $R \to S$ is defined as:

$$\ker h = \{a \in R \mid h(a) = 0\}.$$

- The *image* of a homomorphism h: $R \to S$ is defined as:

$$\text{im } h = \{b \in S \mid \exists a \in R,\ h(a) = b\}.$$

One can easily verify that ker h is an ideal of R. The quotient ring of R with respect to ker h has the usual notation: R / ker h. The elements of R / ker h are the cosets of ker h in R, i.e. x + ker h.

Now consider the ring homomorphism

$$f : R/\ker h \to \text{im } h$$
$$x + \ker h \mapsto h(x)$$

Then f is an isomorphism, hence h: $R \to S$ induces a ring isomorphism: R / ker $h \cong$ im h. This is known as the *homomorphism theorem* for rings.

2.1.3 Polynomial Rings [3]

Let S be a ring, and x a symbol called a *variable* or *indeterminate* not belonging to S. An expression of the form: $f(x) = \sum^n_{i=0} a_i x^i$, where $a_i \in S$, (n finite) is called a *univariate polynomial* over the ring S, and the ring elements a_i are called the *coefficients* of f. Assume that $a_k = 0$ for all $k > 0$.

- The *leading coefficient* of f is a_n, and a polynomial whose $a_n = 1$ is called a *monic polynomial*.

- *Addition* and *multiplication* on the ring of polynomials are defined as follows:

$$\text{Let } f = \sum_i a_i x^i \text{ and } g = \sum_j b_j x^j\ ,$$
$$f + g = \sum_k c_k x^k\ ,\quad \text{where } c_k = a_k + b_k\ ,$$
$$f \cdot g = \sum_k c_k x^k\ ,\ \text{where } c_k = \sum^n_{i=0} a_i b_{k-i}\ .$$

With the operations defined as shown, it can be verified that the set of polynomials forms a commutative ring with zero element 0 and identity element 1.

- The *polynomial ring* thus obtained by adjoining the symbol x to S, is hence denoted as: $R = S[x]$. The variable x is just a placeholder.
- The *degree* of a nonzero polynomial f, denoted $\deg(f)$, is the highest power of x appearing in f. By convention, $\deg(0) = -\infty$.

- The *ring of multivariate polynomials* over S is obtained by adjoining the variables $x_1, \ldots, x_n$, successively, to S. Then $R = S[x_1]\ldots[x_n] = S[x_1, \ldots, x_n]$.

To understand this more closely, consider the bivariate polynomials $S[x, y]$. These can be viewed as univariate polynomials in y with coefficients in the ring $S[x]$.

Thus R consists of the multivariate polynomials of the form: $\sum a_{e1,\ldots,en}\, x_1^{e1} \ldots x_n^{en}$.

- A *power product* (or *monomial*) is an element of R of the form: $p = x_1^{e1} \ldots x_n^{en}$, $e_i \geq 0$.
- The *total degree* of the power product is $\deg(p) = \sum_{i=1}^{n} e_i$.
- A *term* consists of a power product together with its coefficient. Thus a polynomial is simply a sum of a finite set of terms.
- The *total degree* of a multivariate polynomial f is the maximum of the total degrees of the monomials in it.

To define an analogous of leading coefficient for multivariate polynomials, an ordering of the terms must be defined first. Many ordering techniques are available, among which the *lexicographic ordering* and the *total lexicographic ordering*.

Let $p = x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$ and $q = x_1^{b_1} x_2^{b_2} \cdots x_n^{b_n}$ be two power products.

- We say that $p > q$ according to lexicographic ordering ($p >_{\text{lex}} q$) if:
  $a_i \neq b_i$ for some i, then for the *minimum* such i, $a_i > b_i$.
- We say that $p > q$ according to total lexicographic ordering ($p >_{\text{tlex}} q$) if:
  a) $\deg(p) > \deg(q)$, or else,
  b) in case $\deg(p) = \deg(q)$, $a_i \neq b_i$ for some i, then for the *minimum* such i, $a_i > b_i$
  .

Examples:

In the case of polynomials in 4 variables, w, x, y, z, we have:

(i) According to lexicographic ordering:

$$1 < z < z^2 < \ldots$$
$$< y < yz < \ldots < y^2 \ldots$$
$$< x < xz < \ldots < xy < \ldots < x^2 \ldots$$

$$< w < wz < \ldots < wy < \ldots < wx < \ldots < w^2 \ldots$$

(ii) According to total lexicographic ordering:

$$1 < z < y < x < w < z^2 < yz < y^2 < xz < xy < x^2 < wz < wy < wx < w^2 \ldots$$

- The *leading term* of a polynomial f is the monomial in f whose power product (monomial) is largest relative to the chosen ordering. Its coefficient is called the *leading coefficient*.

Example:

According to any of the previous ordering methods, the leading term of $f = 4xy + y - 5$ is $4xy$.

2.1.4 Representation of numbers and polynomials [1]

2.1.4.1 Motivation

One of the basic purposes of computer algebra systems is exact arithmetic. To achieve this goal, integers have to be represented exactly, regardless of their size. Computers store data in pieces called *words*. It is common to have machines that use 64-bit words. Then one machine word contains a *single precision* integer between 0 and 2^{64}-1. In practice, of course, the size of the machine memory bounds the integers that can be represented. But it is certainly not acceptable that the word length of the machine limits this size.

How can we represent integers outside the range $\{0, \ldots, 2^{64}\text{-}1\}$? The answer is to use *multiprecision integers* [1], which are lists of 64-bit words. Such structures are needed to represent integers of arbitrary sizes, that can grow "as the need arises" [2].

2.1.4.2 Positional notation for integers

A radix-r representation of an integer a with n digits takes the form:

$$a = (-1)^s \sum_{0 \le i \le n} a_i \cdot r^i, \text{ where } s, \text{ the sign, is 0 or 1, and } a_i \in \{0, \ldots, r\text{-}1\}.$$

This representation is known as the *positional notation*. In the case of multiprecision integers mentioned earlier, $r = 2^{64}$. This allows of course for sufficiently large numbers to be represented, the constraint being to some extent the memory size.

Example: [2]

Consider for example the integer 2000014720401, decimal for simplicity. Consider a positional number system with *basis* $r = 10000$. Then, the previous number would be represented by the following list: [+,2,0,1472,401], starting with most significant digits.

2.1.4.3 Performing integer arithmetic

Here is a review of the fundamentals of multiprecision arithmetic by considering the simplest operation of addition of two integers of positive sign. In [1], the algorithms of other operations are introduced.

Let a and b be two multiprecision integers: $a = \sum_{0 \le i \le n} a_i \cdot 2^{64i}$, $b = \sum_{0 \le i \le n} b_i \cdot 2^{64i}$. It is required to compute the sum $s = \sum_{0 \le i \le n+1} s_i \cdot 2^{64i}$, such that $s = a + b$.

First, assume the use of a hypothetical processor that has at its disposal a command for the addition of two single word (single precision) integers x and y. The output of the operation is a word w plus a *carry flag* c. This is a special bit in the processor status word which indicates whether the result exceeds 2^{64} (the largest representable word integer) or not. The carry flag can assume only the values 0 (no overflow) and 1 (overflow). There are also instructions to set and clear the flag.

The key point in the addition of multiprecision integers is that of *carry propagation*. In the worst case, a carry form the addition of the least significant words a_0 and b_0, may affect the addition of a_n and b_n. An example of this case is the addition of $a = 2^{64(n+1)} - 1$ and $b = 1$. Here is the algorithm:

Input: a, b

Output: $s = a + b$. (c represent the carry flag).

$c_0 := 0$;

for $i := 0, \ldots, n$ **do**

$s_i := a_i + b_i + c_i$, $c_{i+1} := 0$;

if $s_i \ge 2^{64}$ **then** $s_i := s_i - 2^{64}$, $c_{i+1} := 1$;

$s_{n+1} := c_{n+1}$

return s

Example:

Let $a = 9438$, $b = 945$, and use the radix 10 instead of 2^{64}.

$a = 9 \cdot 10^3 + 4 \cdot 10^2 + 3 \cdot 10 + 8$, $b = 9 \cdot 10^2 + 4 \cdot 10 + 5$.

i	4	3	2	1	0
a_i		9	4	3	8
b_i		0	9	4	5
c_i	1	1	0	1	0
s_i	1	0	3	8	3

Related data types occurring in current processors and software systems are single and multiprecision *floating point numbers*. These represent approximations of real numbers, and arithmetic operations are subject to *rounding errors*, in contrast to arithmetic operations on multiprecision integers which are exact. Computation with floating point numbers is the main topic of *numerical analysis*, which is a theme of its own, not discussed in this text.

2.1.4.4 Representation of polynomials

Recall the definition of a polynomial $a \in R[x]$, in the variable x over the ring R. It consists of a finite sequence $(a_0, \ldots, a_n)$ of ring elements (the *coefficients* of a), for some $n \in N$ (the *degree* of a). It is written as: $a = a_n x^n + \ldots + a_1 x + a_0$.

The previous representation of polynomials is quite analogous to the positional notation of integers introduced in a previous section. A number a has the radix r representation:

$$a = a_n r^n + \ldots + a_1 r + a_0 \text{ , with digits } a_0 \text{ , } \ldots, a_n \in \{0, \ldots, r\text{-}1\}.$$

Furthermore, if we take polynomials over $Z_r = \{0, \ldots, r\text{-}1\}$, the ring of integers modulo r, with addition and multiplication modulo r, the representations are similar. This similarity is an important point for computer algebra: many algorithms apply to both the integers and polynomial cases, with small modifications.

Thus, polynomials can be represented using arrays whose i^{th} element is a_i. This assumes of course that a way of representing the coefficients from R exists.

2.1.4.5 Polynomial arithmetic

Arithmetic algorithms for polynomials are relatively easier than their counterpart for integers. The basic distinction lies in the absence of the carry rule in the polynomial case. While the low digits have some influence on the high digits in the addition of integers, each term in a polynomial is independent of the other.

In what follows, simple algorithms for addition and multiplication of polynomials are presented. More arithmetic on the polynomials may be referred to in [1].

Addition:

Input: $a = \sum_{0 \le i \le n} a_i \cdot x^i$, $b = \sum_{0 \le i \le n} b_i \cdot x^i$ in $R[x]$, where R is a ring.

Output: $s = a + b \in R[x]$.

for $i := 0, \ldots, n$ **do** $s_i := a_i + b_i$;

return s

Multiplication:

The first version of the algorithm mimics the method of multiplication learned in school. Note also this is a more general algorithm that lets the two input polynomials be of different degrees.

Input: $a = \sum_{0 \le i \le n} a_i \cdot x^i$, $b = \sum_{0 \le i \le m} b_i \cdot x^i$ in $R[x]$, where R is a ring.

Output: $p = a \cdot b \in R[x]$.

for $i := 0, \ldots, n$ **do** $d_i := a_i x^i \cdot b$;

return $p = \sum_{0 \le i \le m} d_i$.

The multiplication $a_i x^i \cdot b$ is realized as the multiplication of each b_j by a_i plus a shift by i places. The variable x serves just as a placeholder, and there is no arithmetic involved in multiplying by x or any powers of it.

Note also that the previous algorithm is not the "best" for multiplication. One basic drawback of a computer implementation of such algorithm is the requirement of storing n+1 intermediate polynomials with m+1 coefficients each. A way around this is to interleave the final addition with the computation of $a_i x^i \cdot b$. This uses only a space for the storage of two polynomials, i.e. n+m coefficients.

Although the purpose of this text is not to analyze the efficiency of algorithms, but rather to introduce elementary and simple ones, here is a "better" multiplication algorithm that uses only intermediate results of size 1.

Here the product's coefficients p_k are computed one by one. The degrees of a and b being n and m respectively, and that of p, n+m, the loop for computing the p_k is:

for $k := 0, \ldots, n+m$ **do**

    $p_k := 0$

    for $i := \max\{0,k-m\}, \ldots, \min\{n,k\}$ **do**

        $p_k := p_k + a_i \cdot b_{k-i}$

2.1.4.6 ***Storage of numbers and polynomials*** [2]

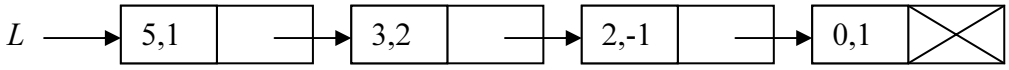
In many algorithms of computer algebra, intermediate results may get very large although inputs and outputs are of moderate sizes. For example, in computing *greatest common divisors* of polynomials, in integration problems or Gröbner basis computations, intermediate polynomials of degree 70 are very common for inputs of degree 5 or 6.

To be able to store these objects efficiently, a need for data structures that reflect the expansion and shrinking of the objects during the computation arises. Arrays have the drawback of being fixed structures. One has to specify the maximum length of the array beforehand. A data structure that overcomes this constraint is called *list*.

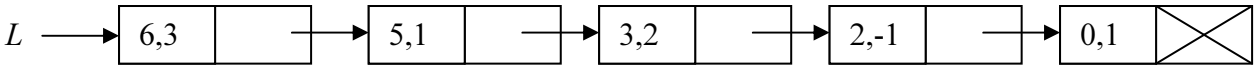
Lists are suited for representing objects of variable length because they are not fixed in size. Of course, practically, we have memory constraints in any computer. But these less severe than constraints on each data object.

Example:

Consider the polynomial $x^5 + 2x^3 - x^2 + 1$ represented by the list [[5,1], [3,2], [2,-1], [0,1]]. Then the term $3x^6$ is added. The list becomes: [[6,3], [5,1], …, [0,1]]. An illustration of this example follows, where *L* is a pointer to the first element of the list.



After insertion of the term $3x^6$:



This section shall not go deeper than that in implementation techniques. It only introduces some of the techniques used in symbolic computation systems.

2.2 THE EUCLIDEAN ALGORITHM

The Euclidean algorithm is basic in computing greatest common divisors (gcds) and this, in turn, is an extremely frequent operation in any computation in computer algebra.

2.2.1 Basic notions

Definition: Let R be a ring and $a, b, c \in R$. Then c is a *greatest common divisor* or *gcd* of a and b if:

(i) $c \mid a$ and $c \mid b$ (c divides both a and b),

(ii) if $d \mid a$ and $d \mid b$, then $d \mid c$, for all $d \in R$.

Recall that in a Euclidean domain R, we can divide any $a, b \in R$, $b \neq 0$, with *quotient* q and *remainder* r, so that: $a = qb + r$. The used notation is $q = a$ div b and $r = a$ rem b.

- A *unit* $u \in R$ is any element with a multiplicative inverse $v \in R$, so that $uv = 1$.
- Two elements a, b are *associate* if $a = ub$ for a unit $b \in R$, denoted: $a \sim b$.

In general, gcds need not be unique. However, all gcds of a and b are precisely the associates of one of them. As an example, the only units in Z are 1 and -1. The gcds of 12 and 15 in Z are 3 and -3 which are associates. [1]

2.2.2 The Classical Euclidean Algorithm [1], [2]

The following algorithm computes greatest common divisors in an arbitrary Euclidean domain.

Input: $f, g \in R$, where R is a Euclidean domain.

Output: gcd of f and g.

$r_0 := f, r_1 := g$

$i := 1;$

while $r_i \neq 0$ **do** $r_{i+1} := r_{i-1}$ rem r_i , $i := i+1$

return r_{i-1} .

Example:

Tracing the algorithm with input $f = 126$ and $g = 35$ gives

i	r_{i-1}	r_i	$r_{i+1} = r_{i-1}$ rem r_i
1	126	35	21
2	35	21	14
3	21	14	7
4	14	7	0

It is important to note that many algorithms that apply to integers apply to polynomials as well. The similarity between the two objects was highlighted in the previous section. Thus, the gcd algorithm is well suited to polynomials with only minor changes.

As mentioned in the previous section, the gcds of two elements of R are associates of one of them. Now, take R to be $Q[x]$, the ring of polynomials with rational coefficients. It is noted that Q is a field. Hence every nonzero rational number is a unit (i.e., invertible). As a result, $ua \sim a$ in $R = Q[x]$ for all nonzero $u \in Q$ and all $a \in R$. In terms of gcds, we have an infinite number of gcds for f and $g \in R$, all multiples of a certain $a \in R$, among which only one representative should be chosen.

A reasonable choice is the polynomial with leading coefficient 1. Any polynomial a can be *normalized* (leading coefficient = 1) by dividing it by its leading coefficient. It turns out that, in the polynomial case, the classical Euclidean algorithm should be modified such that all the remainders r_i are normalized. [1]

2.2.3 The Extended Euclidean Algorithm [1], [2]

Putting into consideration the above remark on normalization, the following variant of the classical Euclidean algorithm, works with normalized remainders and computes, not only the gcd, but also a representation of it as a linear combination of the inputs.

A function *normal*(.) is used. For $R = Z$, normal(a) = $|a|$, and for $R = F[x]$, normal(a) = a/lc(a), where lc(.) is the leading coefficient of a (by convention, lc(0) = 1).

Input: $f, g \in R$, (R is a Euclidean domain with normal form).

Output: gcd(f,g), $s, t \in R$, such that $sf + tg$ = gcd(f,g), l as computed below.

$a_0 := \text{lc}(f)$, $r_0 := \text{normal}(f)$, $s_0 := a_0^{-1}$, $t_0 := 0$,

$a_1 := \text{lc}(g)$, $r_1 := \text{normal}(g)$, $s_1 := 0$, $t_1 := a_1^{-1}$;

$i := 1$;

while $r_i \neq 0$ **do**

$q_i := r_{i-1}$ div r_i

a_{i+1} := lc(r_{i-1} rem r_i)

r_{i+1} := normal(r_{i-1} rem r_i)

s_{i+1} := (s_{i-1} - $q_i s_i$) / a_{i+1}

t_{i+1} := (t_{i-1} - $q_i t_i$) / a_{i+1}

i := i + 1;

l := i - 1;

return r_{i-1}, s_{i-1}, t_{i-1}, l.

The variables used in the algorithm are:

l: the *Euclidean length* of the pair (f,g).

r_i: the *remainders*, q_i: the *quotients*.

s_i and t_i from the i^{th} iteration satisfy: $s_i f + t_i g = r_i$ for all $0 \le i \le l+1$. In particular, for $i = l$, the output of the algorithm, s_l and t_l satisfy: $sf + tg = r_l = \gcd(f,g)$. They are called the *Bézout coefficients* of f and g. [1]

Example:

(a) $R = Z$, $f = 126$, $g = 35$.

Tracing the algorithm yields the following table.

i	q_i	a_i	r_i	s_i	t_i
0		1	126	1	0
1	3	1	35	0	1
2	1	1	21	1	-3
3	1	1	14	-1	4
4	2	1	7	2	-7
5		1	0	-5	18

The final result output by the algorithm as read out of row 4 is:

$$\gcd(126,35) = 7 = 2 \cdot 126 + (-7) \cdot 35.$$

(ii) $R = Q[x]$, $f = 12x^3 - 28x^2 + 20x - 4$, $g = -12x^2 + 10x - 2$.

The tracing of the algorithm follows.

i	q_i	a_i	r_i	s_i	t_i
0		12	$x^3 - 7/3x^2 + 5/3x - 1/3$	1/12	0
1	$x - 3/2$	-12	$x^2 - 5/6x + 1/6$	0	-12
2	$x - 1/2$	1/4	$x - 1/3$	1/3	$1/3x - 1/2$
3		1	0	$-1/3x + 1/6$	$-1/3x^2 + 2/3x - 1/3$

From row 2, the result is:

$$\gcd(f,g) = x - 1/3 = 1/3 \cdot (12x^3 - 28x^2 + 20x - 4) + (1/3x - 1/2)(-12x^2 + 10x - 2).$$

2.3 APPLICATIONS OF THE EUCLIDEAN ALGORITHM

Before presenting some of the Euclidean algorithm applications, we review modular arithmetic first [1]. Moreover, the review is also a prerequisite for the Chinese remainder algorithm presented in the section that follows.

2.3.1 Modular arithmetic

In *modular arithmetic*, one computes with remainders arithmetic expressions modulo some nonzero integer. Taking a number m, two integers a and b are equal modulo m if they have the same remainder on division by m.

$$a \equiv b \bmod m \Rightarrow m \mid (a-b), \text{ i.e. } a-b \text{ is divisible by } m.$$

The basic rule for computing with congruences is:

$$a \equiv b \bmod m \Rightarrow a*c \equiv b*c \bmod m,$$

where a, b, $c \in Z$ and * is one of +, -, ·. Using this rule inductively, expressions can be computed modulo m very efficiently by first reducing all integers modulo m, and then, step by step, performing an arithmetic expression in Z and immediately reducing the result modulo m again. In this way, the intermediate results never exceed m^2.

Examples:

(a) Compute of an expression, efficiently, modulo 7.

$$20 \cdot (-89) + 32 \equiv 6 \cdot 2 + 4 \equiv 12 + 4 \equiv 5 + 4 \equiv 9 \equiv 2 \bmod 7.$$

(b) Another familiar example from school.

The remainder of a number modulo 9 (or 3) equals the remainder of the sum of its decimal digits. In particular, a number is divisible by 9 (or 3) if and only if this sum is so.

Why does this work? Let $a = \sum_{0 \le i < n} a_i \cdot 10^i$ be the decimal representation of $a \in N$. Since $10 \equiv 1 \bmod 9$ (and mod 3), $a \equiv \sum_{0 \le i < n} a_i \cdot 1^i = \sum_{0 \le i < n} a_i \bmod 9$.

Modular arithmetic can also be performed with polynomials over a ring R. The mathematical concept behind modular arithmetic is the *residue class ring*. If R is a ring (say Z or $F[x]$), and $m \in R$, then $\langle m \rangle = mR = \{mr \mid r \in R\}$, the set of all multiples of m, is the *ideal* generated by m. The residue class ring $R / mR = R / \langle m \rangle = \{f + mr \mid f, r \in R\}$ consists of all residue classes of $R / \langle m \rangle$. This is also written Z_m for $Z / \langle m \rangle$.

For the polynomial case, the simplest nontrivial modulus is a linear polynomial $x - u$. For any polynomial $f \in R[x]$, and $u \in R$, the polynomial $f(x) - f(u)$ has u as a zero hence is divisible by $(x - u)$. If we let $q = (f(x) - f(u)) / (x - u)$, then $f = q \cdot (x - u) + f(u)$. By the division theorem, $f(u)$ is the remainder of f on division by $(x - u)$. Thus, $f \equiv f(u) \bmod (x - u)$, and calculating modulo $(x - u)$ is the same as evaluating at u.

It is common to perform computations modulo polynomials *and* modulo integers. Then there are two levels of computation: coefficient operations, which are performed modulo m in Z_m, and polynomial operations, which are performed modulo some polynomial $f \in Z_m[x]$. That is we are working in "double" residue class rings of the form $Z_m[x]/\langle f \rangle$.

Computationally, it is equivalent to work with a *set of representatives* of the elements of the residue class ring. There are two representations, both useful in specific applications:

- the least non-negative representation $\{0, 1, \ldots, m - 1\}$, and
- the symmetric least absolute value representation $\{a \in Z \mid -m/2 \le a \le m/2\}$.

For example, $Z / 5Z = Z_5$ can be represented as $\{0, 1, 2, 3, 4, 5\}$ or $\{-2, -1, 0, 1, 2\}$.

Example:

$(4x + 1)(3x^2 + 2x) = 12x^3 + 11x^2 + 2x$ in $Z[x]$

$(4x + 1)(3x^2 + 2x) = 2x^3 + x^2 + 2x$ in $Z_5[x]$

Now, division with remainder yields

$2x^3 + x^2 + 2x = 2 \cdot (x^3 + 4x) + x^2 - 6x$ in $Z[x]$

$2x^3 + x^2 + 2x = 2 \cdot (x^3 + 4x) + x^2 + 4x$ in $Z_5[x]$

$\therefore (4x + 1)(3x^2 + 2x) \equiv x^2 + 4x \bmod (x^3 + 4x)$ in $Z_5[x]$, or equivalently,

$(4t + 1)(3t^2 + 2t) = t^2 + 4t$ in $Z_5[x]/\langle x^3 + 4x \rangle$, where $t = x \bmod x^3 + 4x$.

2.3.2 Modular exponentiation

An important tool for modular exponentiation is *repeated squaring* (or square and multiply). This technique works in any set with an associative multiplication, but here, it is used in residue class ring. Modular exponentiation will be required later in computing modular inverses. Here follows the algorithm.

Input: $a \in R$, a ring with identity, $n \in N$ ($n \neq 0$)

Output: $a^n \in R$.

Find the binary representation of $n = 2^k + n_{k-1} \cdot 2^k + \ldots + n_1 \cdot 2 + n_0,\ n_i \in \{0, 1\}$

$b := a$;

for $i := k\text{-}1, k\text{-}2, \ldots, 0$ **do**

    if $n_i = 1$ **then** $b := b^2a$ **else** $b := b^2$

return b

This procedure uses $\log_2 n$ squarings plus at most $\log_2 n$ multiplications by a (if the binary representation of n is all 1's).

Example:

It is required to compute a^{13}. The binary representation of 13 is: $1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2 + 1$. Tracing the algorithm yields: $a^{13} = ((a^2 \cdot a)^2)^2 \cdot a$. If $R = Z_{17} = Z / \langle 17 \rangle$ and $a = 8$ mod 17, then the computation of 8^{13} mod 17 proceeds as follows:

$$\begin{aligned} 8^{13} &\equiv (((8^2 \cdot 8)^2)^2 \cdot 8 \equiv (((-4 \cdot 8)^2)^2) \cdot 8 \\ &\equiv (2^2)^2 \cdot 8 = 4^2 \cdot 8 \\ &\equiv -1 \cdot 8 = -8 \bmod 17. \end{aligned}$$

This is much faster than first evaluating $8^{13} = 549755813888$ and then dividing by 17 with remainder.

2.3.3 Modular inverses

All modular arithmetic presented up to now has been addition and multiplication. How about division and inversion. Do expressions like a^{-1} mod m and a/b mod m make sense? In what follows two methods for computing modular inverses are presented [1]. By some way or the other, computing gcds is required.

- Euclidean method:

Let R be a Euclidean domain, $a, m \in R$, and $S = R / mR$. Then a mod m is a unit (i.e. has an inverse) if and only if gcd(a,m) = 1. In this case, the modular inverse of a mod m can be computed by means of the extended Euclidean algorithm.

The proof of this statement is by construction, and, in the mean time, it shows how the inverse is computed.

By definition, a is invertible modulo $m \Leftrightarrow \exists s \in R$ s.t. $sa \equiv 1 \bmod m$

$$\Leftrightarrow \exists s,t \in R \text{ s.t. } sa + tm = 1$$

$$\Leftrightarrow \gcd(a,m) = 1$$

Note that $km \equiv 0 \bmod m$, for any $k \in R$.

Examples:

(i) Let $R = Z$, $m = 29$, and $a = 12$. Then 12 is invertible mod 29 since gcd(29,12) = 1.

The extended Euclidean algorithm computes $5 \cdot 29 + (-12) \cdot 12 = 1$.

Thus $(-12) \cdot 12 \equiv 17 \cdot 12 \equiv 1 \bmod 29$, hence 17 is the inverse of 12 mod 29.

(ii) Let $R = Q[x]$, $m = x^3 - x + 2$, and $a = x^2$. The last row in the Euclidean algorithm reads:

$$(¼x + ½)(x^3 - x + 2) + (-¼x^2 - ½x + ¼)x^2 = 1,$$

and $(-x^2 - 2x + 1) / 4$ is the inverse of x^2 mod $x^3 - x + 2$.

- Fermat's method:

Let $p \in N$ be a prime and $a, b \in Z$. Then the binomial theorem implies that:

$$(a+b)^p = \sum_{i=0}^{p} \frac{p!}{i!(p-i)!} a^{p-i} b^i = a^p + pa^{p-1}b + \frac{p(p-1)}{2} a^{p-2}b^2 + \cdots + b^n.$$

For all $0 < i < p$, the binomial coefficient is divisible by p, since the numerator (p!) is and the denominator is not. A surprising consequence is:

$$(a + b)^p \equiv a^p + b^p \bmod p. \text{ (the "freshman's dream").}$$

This property can be used in the proof of the *Fermat's little theorem*:

If $p \in N$ is prime and $a \in Z$, then $a^p \equiv a \bmod p$. And, if p doesn't divide a, then $a^{p-1} \equiv 1 \bmod p$.

Since computations are performed modulo p, the theorem needs only to be proved for $0 \le a \le p$-1, which is done by induction on a. The case $a = 0$ is trivial, and for $a > 1$, we have

$a^p = ((a - 1) + 1)^p$

$\equiv (a - 1)^p + 1^p$ by the "freshman's dream"

$\equiv (a - 1) + 1$ by the induction hypothesis

$\equiv a \bmod p$,

Since p is prime, gcd(a,p) = 1, hence a is invertible. Multiplying both sides by a^{-1}, the second part of the theorem follows immediately.

Fermat's theorem, together with repeated squaring, gives us an alternative method to compute inverses in Z_p. Since $a^{p-2}a = a^{p-1} \equiv 1 \bmod p$, then $a^{-1} \equiv a^{p-2} \bmod p$, where a^{p-2} is computed using repeated squaring.

2.3.4 Linear Diophantine equations [1]

A direct application of the extended Euclidean theorem is the solution of what is known as linear Diophantine equations.

Let R be a Euclidean domain, $a, f, g \in R$ be given. The solution of the *linear Diophantine equation* is the pair $(s, t) \in R^2$ such that:

$$sf + tg = a.$$

The existence of a solution and its computation are the subject of the following theorem and its proof.

Let R be a Euclidean domain, $a, f, g \in R$, and $h = \gcd(f,g)$.

(a) The diophantine equation $sf + tg = a$ has a solution $(s, t) \in R^2$ iff h divides a.

(b) If $R = F[x]$ for a field F, $h \neq 0$, the equation is solvable, and $\deg(f) + \deg(g) - \deg(h) > \deg(a)$, then there is a unique solution $(s, t) \in R^2$ such that $\deg(s) < \deg(g) - \deg(h)$ and $\deg(t) < \deg(f) - \deg(h)$.

Proof.

(i) If $s, t \in R$ satisfy the equation, then gcd(f,g) divides $sf + tg$ and hence a. Conversely, assume that $h = \gcd(f,g)$ divides a. The claim is trivial if $h = 0$. Otherwise, we can compute $s^*, t^* \in R$ such that $s^*f + t^*g = h$, using the extended Euclidean algorithm. Thus the required solution is $(s, t) = (s^* \cdot a/h, t^* \cdot a/h)$.

(ii) For the proof of this part, the extended Euclidean algorithm is presented more concisely. Assuming all polynomials to be already normalized, with leading coefficients 1, doesn't affect their degree. The algorithm follows hereunder for convenience. Simultaneous assignment statements use the vector form for short. [2]

Input: $f, g \in R[x]$.

Output: $h, s, t \in R[x]$ such that $h = \gcd(f,g)$, and $sf + tg = h$.

$(r_0, r_1, s_0, s_1, t_0, t_1) := (f, g, 1, 0, 0, 1)$; $i := 1$;

while $r_i \neq 0$ **do**

$q_i := r_{i-1} \text{ div } r_i$;

$(r_{i+1}, s_{i+1}, t_{i+1})$:= $(r_{i-1}, s_{i-1}, t_{i-1}) - q_i \cdot (r_i, s_i, t_i)$; %% Same as computing remainders

$i := i + 1$;

return $(r_{i-1}, s_{i-1}, t_{i-1})$.

Assume the number of the iterations is l. For a degree analysis, note that, obviously, $\deg(q_i) = \deg(r_{i-1}) - \deg(r_i)$ for $1 \le i \le k-1$. For $k = 2$, the claim holds obviously.

If $k > 2$, then for $2 \le i \le k-1$ we have $\deg(r_i) = \deg(r_1) - \sum_{j=2}^{i}\deg(q_j) < \deg(r_1) - \sum_{2\le j\le i-1}\deg(q_j)$. Similarly, $\deg(s_i) \le \sum_{2\le j\le i-1}\deg(q_j)$ and $\deg(t_i) \le \sum_{1\le j\le i-1}\deg(q_j)$. (Note the index range for t_i). Adding the inequalities, we have: $\deg(r_i) + \deg(s_i) < \deg(r_1)$ and $\deg(r_i) + \deg(t_i) < \deg(r_1) + \deg(q_1)$. Putting $i = k-1$, and $\deg(q_1) = \deg(r_0) - \deg(r_1) = \deg(a) - \deg(b)$, the required result follows.

2.3.5 Squarefree factorization[2]

By just computing gcds, we can produce a partial solution of the problem of factorization of polynomials, namely the so-called *squarefree* factorization.

The algebraic system worked on here is any field $F[x_1, \ldots, x_n]$, which has the *unique factorization domain property*, in that any polynomial in f can be factored into a product of *irreducible* polynomials in a *unique way*. By "unique", it is meant uniqueness up to ordering of the factors and multiplication by units. Units of Z are 1 and -1, and those of F are any nonzero constants. If $a = ub$ for $a, b, u \in F$, where u is a unit, then a and b are associates.

Definitions: A polynomial $f(x_1, \ldots, x_n)$ in $F[x_1, \ldots, x_n]$ is *squarefree* iff every nontrivial factor $g(x_1, \ldots, x_n)$ of f (i.e. g not associate of f nor constant) occurs with multiplicity 1 in f.

- The *derivative* f' of a polynomial $f = \sum_{0\le i\le n} a_i x^i \in F[x]$ is $f' = \partial f / \partial x = \sum_{0\le i\le n} i a_i x^{i-1}$. It satisfies the following properties: [1]

(a) *Linearity*: $(f + g)' = f' + g'$.

(b) *Leibniz rule*: $(f \cdot g)' = fg' + f'g$.

There is a simple criterion for deciding squarefreeness, presented in the following theorem.

Let $f(x)$ be a nonzero polynomial in $F[x]$. Then $f(x)$ is squarefree if and only if $\gcd(f, f') = 1$. (Where f' is the derivative of f).

Proof:

If $f(x)$ is not squarefree, i.e. for some non constant $g(x)$, we have $f = g^2h$, then

$$f' = 2gg'h + g^2h'.$$

So f and f' have a non trivial gcd, namely g.

On the other hand, if f is squarefree, it can be written as $f(x) = \prod_{1\le i\le n} f_i(x)$, where the f_i are pairwise relatively prime (i.e. gcd = 1) and irreducible polynomials. Then,

$$f'(x) = \sum_{i=1}^{n}\left(f_i' \prod_{\substack{j=1 \\ j\neq i}}^{n} f_j \right).$$

Now it is easy to see that none of the irreducible factors f_i is a divisor of f', since f_i divides all the terms of f' except the i^{th}. Hence $\gcd(f, f') = 1$, and this finishes the proof.

The square free factorization problem

The squarefree factorization problem for a polynomial $f(x)$ consists of determining the squarefree pairwise relatively prime polynomials $g_1(x), \ldots, g_s(x)$, such that $f(x) = \prod_{1\le i\le s} g_i(x)^i$. The method used in the factorization procedures can be summarized as follows.

Let $a_1 = f$ and $a_2 = \gcd(a_1, a_1')$. Then,

$$a_1'(x) = \sum_{i=1}^{n}\left(i g_i^{\,i-1} g_i' \prod_{\substack{j=1 \\ j\neq i}}^{n} g_j \right), \text{ thus } a_2(x) = \prod_{1\le i\le s} g_i(x)^{i-1} = \prod_{2\le i\le s} g_i(x)^{i-1}.$$

And $c_1 = a_1/a_2 = \prod_{1\le i\le s} g_i(x)$ contains every squarefree factor exactly once. It is now required to isolate each squarefree factor. This is performed, one by one, as follows.

Let $a_3 = \gcd(a_2, a_2') = \prod_{3\le i\le s} g_i(x)^{i-2}$ as previously, and $c_2 = a_2/a_3 = \prod_{2\le i\le s} g_i(x)$. Thus, $c_2(x)$ contains every squarefree factor of multiplicity ≥ 2 exactly once. It is now easy to isolate $g_1(x)$ by noting that $g_1(x) = c_1/c_2$.

One more iteration summarizes the procedure:

$a_4 = \gcd(a_3, a_3') = \prod_{4\le i\le s} g_i(x)^{i-3}$, $c_3 = a_3/a_4 = \prod_{3\le i\le s} g_i(x)$. So $g_2(x) = c_2/c_3$.

The iterations proceed until $c_{s+1}(x) = 1$.

<u>Algorithm</u>

Input: $f(x) \in F[x]$.

Output: $g_1(x), \ldots, g_s(x) \in F[x]$, such that $f(x) = \prod_{1\le i\le s} g_i(x)^i$

$a_1 := f$; $\quad a_2 := \gcd(a_1, a_1')$; $\quad c_1 := a_1/a_2$;

$a_1 := a_2$; $\quad a_2 := \gcd(a_2, a_2')$; $\quad c_2 := a_1/a_2$;

$g_1 := c_1/c_2$;

$i := 2$;

while $c_2 \neq 1$ **do**

$a_1 := a_2$; $a_2 := \gcd(a_2, a_2')$;

$c_1 := c_2$; $c_2 := a_1/a_2$;

$g_i := c_1/c_2$;

$i := i + 1$;

return $g_1, \ldots, g_s$.

Example:

The squarefree factorization of $f(x) = x^7 + x^6 - x^5 - x^4 - x^3 - x^2 + x + 1$ over Z is:

$$(x^2 + 1)(x - 1)^2(x + 1)^3.$$

A tracing of the algorithm yields the same result.

i	a_1	a_2	c_i	g_i
0	$x^7 + x^6 - x^5 - x^4 - x^3 - x^2 + x + 1$	$x^3 + x^2 - x - 1$	$x^4 - 1$	n/a
1	$x^3 + x^2 - x - 1$	$x + 1$	$x^2 - 1$	$x^2 + 1$
2	$x + 1$	1	$x + 1$	$x - 1$
3	1	1	1	$x + 1$

2.3.6 Squarefree partial fraction decomposition [2]

The squarefree partial fraction decomposition is an indirect application of the Euclidean algorithm. It uses two of its applications, namely the squarefree factorization and the solution of linear Diophantine equations.

Let p/q be a *proper rational function* over the field F, i.e., $p, q \in F[x]$, $\gcd(p, q) = 1$ and $\deg(p) < \deg(q)$. Let $q = q_1 \cdot q_2^2 \ldots q_k^k$ be the squarefree factorization of q.

Then, $\frac{p(x)}{q(x)} = \sum_{i=1}^{k} \frac{a_i(x)}{q_i(x)^i}$, $\deg(a_i) < \deg(q_i^i)$, $1 \le i \le k$, $a_i \in F[x]$, is called *the incomplete squarefree partial fraction decomposition* of p/q.

And, $\frac{p(x)}{q(x)} = \sum_{i=1}^{k} \sum_{j=1}^{i} \frac{b_{ij}(x)}{q_i(x)^j}$, $\deg(b_{ij}) < \deg(q_i)$, $1 \le j \le i \le k$, $b_{ij} \in F[x]$, is called the (*complete*) *squarefree partial fraction decomposition* of p/q.

The algorithm

The first step is to compute the incomplete decomposition, i.e. the a_i's, since the algorithm of the previous section can obtain the factorization of the denominator.

The method consists of isolating the partial fractions one by one. Denote the isolated term by a_i / q_i^i and the rest of the fraction (still not expanded) by c_i / d_i. Then,

$$\frac{p}{q} = \frac{a_1}{q_1} + \cdots + \frac{a_i}{q_i} + \frac{c_i}{d_i}, \text{ where } d_{i\text{-}1} = {q_{i+1}}^{i+1} \ldots q_k^k.$$

The a_i and c_i can be determined iteratively as follows.

Initially, the whole fraction is not expanded so that $c_0 / d_0 = p / q$. The first iteration reads:

$$p / q = a_1 / q_1 + c_1 / d_1.$$

Multiplying both sides by q results in: $p = a_1 q_2^2 \ldots q_k^k + c_1 q_1$, or, in terms of the algorithm's variables: $c_0 = a_1 d_1 + c_1 q_1$. Since $\deg(a_1) < \deg(q_1)$ and $\deg(c_1) < \deg(d_1)$, this equation can be solved as a linear Diophantine equation, as presented earlier in the text.

In general, $c_{i\text{-}1} / d_{i\text{-}1} = p / q - a_1 / q_1 - \ldots - a_{i\text{-}1} / {q_{i\text{-}1}}^{i\text{-}1}$. Hence the unachieved partial fraction decomposition can be read at the i^{th} step as: $c_{i\text{-}1} / d_{i\text{-}1} = a_i / q_i^i + c_i / d_i$. Multiplying both sides by d_i yields the Diophantine equation: $c_{i\text{-}1} = c_i \cdot q_i^i + a_i \cdot d_i$.

The algorithm follows immediately.

Input: $p, q \in F[x]$ such that $\gcd(p, q) = 1$.

Output: $a_1, \ldots, a_k$ and $q_1, \ldots, q_k$ such that: $p / q = \sum_{i=1}^{k} (a_i / q_i^i)$, the incomplete squarefree partial fraction decomposition of p / q.

$(q_1, \ldots, q_k)$:= Squarefree_Factorization(q);

$c_0 := p$; $\quad d_0 := q$; $\quad i := 1$;

while $i < k$ **do**

$d_i := d_{i\text{-}1} / q_i^i$;

Solve $c_{i\text{-}1} = c_i \cdot q_i^i + a_i \cdot d_i$ for c_i and a_i using the extended Euclidean algorithm;

$i := i + 1$;

$a_k := c_{k\text{-}1}$;

return.

Once the incomplete partial fraction decomposition is obtained, the complete decomposition results by successive division by q_i. Namely, if $a_i = s \cdot q_i + t$, then $a_i / q_i^i = s / q_i^{i\text{-}1} + t / q_i^i$.

Example:

$$\frac{p(x)}{q(x)} = \frac{4x^8 - 3x^7 + 25x^6 - 11x^5 + 18x^4 - 9x^3 + 8x^2 - 3x + 1}{3x^9 - 2x^8 + 7x^7 - 4x^6 + 5x^5 - 2x^4 + x^3}.$$

The squarefree factorization of $q(x)$ is $(3x^2 - 2x + 1)(x^2 + 1)^2x^3$. And the application of the incomplete squarefree partial fraction decomposition algorithm yields:

$$\frac{p(x)}{q(x)} = \frac{4x}{3x^2 - 2x + 1} + \frac{-x^3 + 2x + 2}{(x^2 + 1)^2} + \frac{x^2 - x + 1}{x^3}.$$

By successive division of the numerators by the corresponding q_i's, the complete decomposition is attained:

$$\frac{p(x)}{q(x)} = \frac{4x}{3x^2 - 2x + 1} + \frac{-x}{x^2 + 1} + \frac{3x + 2}{(x^2 + 1)^2} + \frac{1}{x} + \frac{-1}{x^2} + \frac{1}{x^3}.$$

2.3.7 Integration of rational functions [2]

The integration of rational functions is intimately related to the ideas and algorithms mentioned above. In fact, the problem is to compute $\int (p(x) / q(x))\, dx$, where $p, q \in Q[x]$, $\gcd(p,q) = 1$, and q is monic. From classical calculus, this integral can be solved by decomposing p / q into its complete partial fraction decomposition, which requires factorization of q. The result can be expressed as: $\int (p(x) / q(x))\, dx = g(x) / q(x) + c_1 \ln(x - a_1) + \ldots + c_n \ln(x - a_n)$, where $g(x) \in Q[x]$, $a_1, \ldots, a_n$ are the different roots of q in the complex field C.

However, complete factorization of the denominator, which is relatively expensive, can be avoided, using instead only its squarefree factors.

First, the squarefree factorization of the denominator is $q = f_1 \cdot f_2^2 \ \ldots \ f_r^r$, where the $f_i \in Q[x]$ are squarefree, $f_r \neq 1$, $\gcd(f_i, f_j) = 1$ for $i \neq j$. Based on this squarefree factorization, the squarefree partial fraction decomposition of p / q is:

$$\frac{p(x)}{q(x)} = g_0 + \sum_{i=1}^{r} \sum_{j=1}^{i} \frac{g_{ij}}{f_i^j} = g_0 + g_{11}/f_1 + g_{21}/f_2 + \ldots + g_{r1}/f_r + \ldots + g_{rr}/f_r,$$

where $g_0, g_{ij} \in Q[x]$, $\deg(g_{ij}) < \deg(f_i)$, for all $1 \leq j \leq i \leq r$.

Integrating g_0 is trivial, so that the problem remains in integrating the fractions g / f^n. First consider the case where $n \geq 2$ and note that $\deg(g) < \deg(f)$ and that f is squarefree. The computation of $\int (g / f^n)\, dx$ is reduced to that of an integral of the form $\int (h / f^{n-1})\, dx$, where $\deg(h) < \deg(f)$, using the *Hermite* reduction process.

The reduction proceeds as follows. Since f is squarefree, $\gcd(f, f') = 1$. Thus, one can solve the linear Diophantine equation $g = c \cdot f + d \cdot f'$ where $\deg(c)$, $\deg(d) < \deg(f)$. By integration by parts, the required reduction is achieved.

$$\int \frac{g}{f^n} = \int \frac{c \cdot f + d \cdot f'}{f^n} = \int \frac{c}{f^{n-1}} + \int \frac{d \cdot f'}{f^n}$$
$$= \int \frac{c}{f^{n-1}} - \frac{d}{(n-1) f^{n-1}} + \int \frac{d'}{(n-1) f^{n-1}}$$
$$= -\frac{d}{(n-1) f^{n-1}} + \int \frac{c + d'/(n-1)}{f^{n-1}}$$

Repeating the reduction process, collecting all the rational partial results and the remaining integrals, then putting everything over a common denominator, the integration reaches the form:

$$\int \frac{p}{q} = g_0 + \frac{g}{\underbrace{f_2 \cdot f_3^2 \cdots f_r^{r-1}}_{\bar{q}}} + \int \frac{h}{\underbrace{f_1 \cdot f_2 \cdots f_r}_{q^*}},$$

where deg(g) < deg($\bar{q}$) and deg(h) < deg(q*).

The remaining integral $\int (h / q^*)$ can be computed in the following way:

Let $q^*(x) = (x - a_1) \ldots (x - a_n)$, where $a_1, \ldots, a_n$ are the distinct roots of q*. Then

$$\int \frac{h(x)}{q^*(x)} dx = \sum_{i=1}^{n} \int \frac{c_i}{x - a_i} dx = \sum_{i=1}^{n} c_i \ln(x - a_i), \text{ with } c_i = \frac{h(a_i)}{q^{*\prime}(a_i)}, 1 \le i \le n.$$

Example:

The example of the last section is used. We had:

$$\frac{p(x)}{q(x)} = \frac{4x^8 - 3x^7 + 25x^6 - 11x^5 + 18x^4 - 9x^3 + 8x^2 - 3x + 1}{3x^9 - 2x^8 + 7x^7 - 4x^6 + 5x^5 - 2x^4 + x^3},$$

with squarefree partial fraction decomposition

$$\frac{p(x)}{q(x)} = \frac{4x}{3x^2 - 2x + 1} + \frac{-x}{x^2 + 1} + \frac{3x + 2}{(x^2 + 1)^2} + \frac{1}{x} + \frac{-1}{x^2} + \frac{1}{x^3}.$$

Consider the integration of the third term of this decomposition, i.e. $\int \frac{3x + 2}{(x^2 + 1)^2} dx$.

First, the Diophantine equation $g = c \cdot f + d \cdot f'$ is solved. By the Euclidean algorithm, one has: $3x + 2 = 2 \cdot (x^2 + 1) + (-x + 3/2) \cdot (2x)$.

Integration by parts yields

$$\int \frac{3x+2}{(x^2+1)^2}\,dx = \int \frac{2}{x^2+1}\,dx + \int \frac{(-x+3/2)\cdot(2x)}{(x^2+1)^2}\,dx$$
$$= \int \frac{2}{x^2+1}\,dx + \frac{(-x+3/2)\cdot(-1)}{x^2+1} - \int \frac{1}{x^2+1}\,dx$$
$$= \frac{x-3/2}{x^2+1} + \int \frac{1}{x^2+1}\,dx$$

The remaining integral can be evaluated as

$$\int \frac{1}{x^2+1}\,dx = \frac{i}{2}\cdot\ln(1-ix) - \frac{i}{2}\ln(1+ix) = \tan^{-1}(x)\,.$$

2.4 THE CHINESE REMAINDER ALGORITHM

As mentioned in [2], the Chinese remainder algorithm enables to solve many problems in computer algebra by the modular method. The basic idea is to reduce a problem into several problems modulo different primes, solving the simpler problems modulo the primes, and then combining the partial solutions by the Chinese remainder algorithm.

Let R be a Euclidean domain. We review here the Chinese remainder problem.

Chinese remainder problem

Given: $r_1, \ldots, r_n \in R$ (the remainders)

$m_1, \ldots, m_n \in R^*$ (the moduli), pairwise coprime, i.e. $\gcd(m_i, m_j) = 1$, for $1 \le i < j \le r$.

Find: $r \in R$, such that $r \equiv r_i \bmod m_i$ for $1 \le i \le n$.

Several algorithms can be found in [2]. Concerning this problem, an algorithm given in [1] is as follows.

Input: $r_1, \ldots, r_n, m_1, \ldots, m_n \in R$ (as above).

Output: $r \in R$ such that $r \equiv r_i \bmod m_i$ for $1 \le i \le n$.

$m := m_1 \cdot m_2 \ldots m_n$, (the product of the moduli);

for $i := 1, \ldots, n$ **do**

$c_i := m / m_i$;

$d_i := c_i^{-1} \bmod m_i$, by the Euclidean algorithm: $d_i c_i + t_i m_i = 1$;

return $r = (\sum_{1 \le i \le n} c_i d_i r_i) \bmod m$.

To prove that this algorithm solves the problem correctly, observe that $c_i = m / m_i$ is divisible by m_j for $j \ne i$. Thus, $c_i d_i r_i \equiv 0 \bmod m_j$ for all $j \ne i$. Therefore,

$$\sum_{1 \le i \le n} c_i d_i r_i \equiv c_j d_j r_j \equiv r_j \bmod m_j \text{ , for all } 1 \le j \le n,$$

as required.

Example:

It is required to find an integer r such that:

$$r \equiv 3 \bmod 4, \qquad r \equiv 5 \bmod 7, \qquad r \equiv 2 \bmod 3.$$

Here $R = Z$ and $m = 4 \cdot 7 \cdot 3 = 84$.

i	m_i	r_i	$c_i = m / m_i$	$d_i = c_i^{-1} \bmod m_i$	$c_i\, d_i\, r_i$
1	4	3	21	1	63
2	7	5	12	3	180
3	3	2	28	1	56

Hence, $r = 63 + 180 + 56 = 299 \equiv 47 \bmod 84$.

2.5 APPLICATIONS OF THE CHINESE REMAINDER ALGORITHM [1]

2.5.1 The interpolation problem

As most computer algebra algorithms seen so far, the Euclidean domain of definition R can be generalized to the field of polynomials $F[x]$. Here, the problem is almost unchanged, except for the addition of the constraint that $\deg(r_i) < \deg(m_i)$ for $1 \le i \le n$, which yields the result $\deg(r) < \deg(m)$.

The special case where the Chinese remainder algorithm is applied to linear moduli, i.e. all the moduli m_i are linear polynomials of the form $(x - u_i)$, results in what is known as the *interpolation problem.*

Given: $u_1, \ldots, u_n \in F$ (scalars), such that $u_i \neq u_j$ for $i \neq j$.

$r_1, \ldots, r_n \in F$

Find: $f(x) \in F[x]$ such that $f(u_i) = r_i$, $1 \le r \le n$.

First, recall that $f(x) \equiv f(u) \bmod (x - u)$. This is as such for the following reason. For any polynomial $f \in R[x]$, and $u \in R$, the polynomial $f(x) - f(u)$ has u as a zero hence is divisible by $(x - u)$. If we let $q = (f(x) - f(u)) / (x - u)$, then $f = q \cdot (x - u) + f(u)$. By the division theorem, $f(u)$ is the remainder of f on division by $(x - u)$. Thus, $f \equiv f(u) \bmod (x - u)$, and calculating modulo $(x - u)$ is the same as evaluating at u. By a similar reasoning, the inverse of $f(x)$ modulo $(x - u)$, i.e. in $F[x]/\langle x - u\rangle$, is $f(u)^{-1}$.

Applying the Chinese remainder algorithm, the variables take on the following values.

$$c_i = (x - u_1)\cdot(x - u_2) \ldots (x - u_{i-1})\cdot(x - u_{i+1}) \ldots (x - u_n),$$

$$d_i = c_i^{-1} \bmod (x - u_i) = 1 / [(u_i - u_1)\cdot(u_i - u_2) \ldots (u_i - u_{i-1})\cdot(u_i - u_{i+1}) \ldots (u_i - u_n)],$$

Hence, $l_i = c_i \cdot d_i$ is the familiar *Lagrange interpolant* coefficient:

$$l_i = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - u_j}{u_i - u_j}$$

And the solution $f(x) = \sum_{1 \le i \le n} l_i r_i$ is nothing more but the *Lagrange interpolation polynomial*, satisfying $f(u_i) = r_i$, $1 \le r \le n$.

2.5.2 Modular algorithms

An important general concept in computer algebra is *modular algorithms*. It consists of, rather than solving a problem directly (say over a Euclidean domain R), solving it modulo several integers m_i, where the solutions in the *image domains* $R/\langle m \rangle$ are easier, then combining the results. The Chinese remainder algorithm is an important tool for this purpose. The basic idea is to solve a problem over the Euclidean domain R by reducing it to several problems modulo different primes, solving the simpler problems modulo the primes, and then combining the partial solutions using the mentioned algorithm. In many situations, this modular method results in algorithms with extremely good complexity behavior [2]. The following figure illustrates the general scheme for modular algorithms.

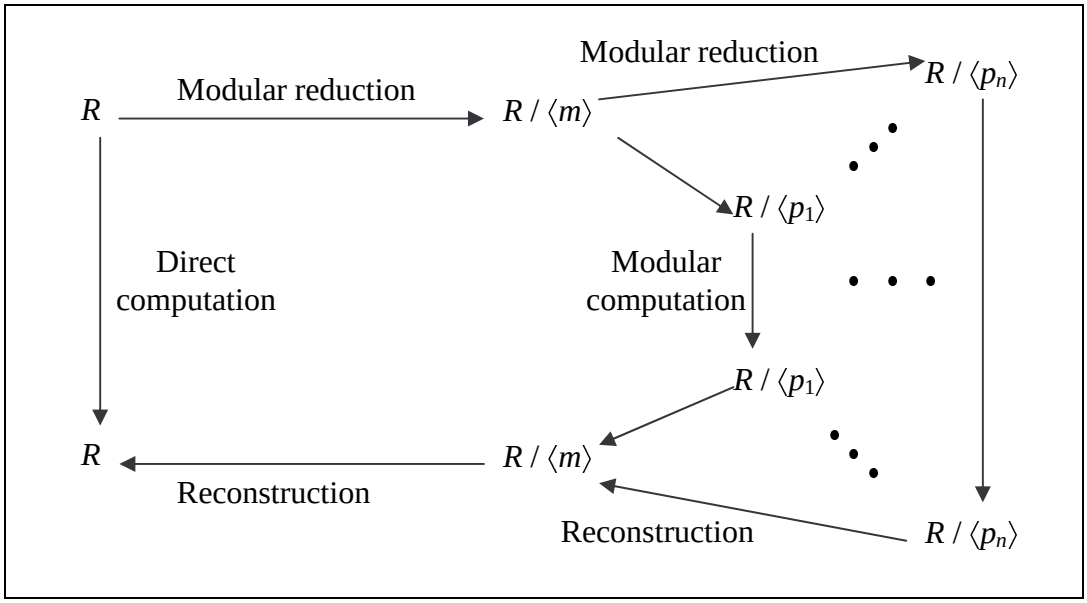


Figure 2-2. General scheme for modular algorithms.

In what follows, two applications illustrating this principle are presented.

2.5.3 Modular determinant computation

In what follows, we see how the Chinese remainder algorithm can simplify the computation of determinants. The problem is to compute the determinant $\det A \in Z$ of an $n \times n$ matrix $A = (a_{ij})_{1 \le i,j \le n} \in Z$. It is known from linear algebra that this problem can be solved by

means of Gaussian elimination over Q, where the matrix is reduced to an upper triangular form, then its determinant is the product of the diagonal elements, $a_{i\,i}$, $1 \le i \le n$.

The problem costs at most $2n^3$ operations, and this is polynomial time in the input size of course. But the complexity arises from the intermediate results, numerators and denominators, which grow exponentially. The cost of one operation, which depends on the number of digits of its operands, rises unexpectedly. To illustrate this effect, consider the matrix at the k^{th} stage during the elimination process, and suppose for simplicity that A is not singular and that no row or column permutations are required. The matrix reaches the following form.

$$\begin{vmatrix} * & & & & & & \\ & \ddots & & & * & & \\ & & * & & & & \\ & & a_{kk}^{(k)} & \cdots & a_{kj}^{(k)} & \cdots \\ 0 & & \vdots & & \vdots & \\ & & a_{ik}^{(k)} & \cdots & a_{ij}^{(k)} & \cdots \\ & & \vdots & & \vdots & \end{vmatrix}$$

A star "*" denotes an arbitrary rational number, and the upper diagonal entries are nonzero. The diagonal element $a_{kk}^{(k)} \neq 0$ is the new pivot element, and the entries of the matrix for the rows $k \le i \le n$ and the columns $k \le j \le n$ change according to the formula

$$a_{i\,j}^{(k+1)} = a_{i\,j}^{(k)} - \left(a_{i\,k}^{(k)} / a_{k\,k}^{(k)} \right) a_{k\,j}^{(k)} .$$

If b_k is an upper bound for the absolute value of the numerators and denominators of all $a_{i\,k}^{(k)}$, so that in particular $| a_{i\,j} | \le b_0$ for $1 \le i,j \le n$, then, substituting in the formula above, gives the bounds: $b_k \le 2b_{k-1}^4 \le 4b_{k-2}^{4^2} \le \cdots \le 2^k b_0^{4^k}$ which is exponential in the input size.

The simplest way to control the intermediate results sizes is to chose a prime p which is guaranteed to be larger than $2 \cdot | \det A |$ and perform Gaussian elimination on A modulo p. In this way, it is guaranteed that $d \equiv \det A \bmod p$ falls in the range $-p/2 < r < p/2$. An upper bound on the size of the determinant of a matrix is given by *Hadamard's inequality*:

$| \det A | \le n^{n/2} b^n$, where $b = \max_{1 \le i,j \le n} | a_{i\,j} |$ is the maximum absolute value of an entry of A.

Example:

Let $A = \begin{pmatrix} 4 & 5 \\ 6 & -7 \end{pmatrix}$. After Gaussian elimination, the matrix has the form $\begin{pmatrix} 4 & 5 \\ 0 & -29/2 \end{pmatrix}$, so that $\det A = -58$.

The Hadamard bound $|\det A| \le 2^1 7^2 = 98$ is reasonably close to $|\det A| = 58$. We choose a prime $p > 2 \cdot 98$, say $p = 199$, and perform Gaussian elimination modulo 199. The inverse of 4 is 50, and $\det(A \bmod 199) = \det\begin{pmatrix} 4 & 5 \\ 0 & 85 \end{pmatrix} = 141 = -58$ in Z_{199}.

There is not much progress using the Hadamard's bound, specially for large matrices. The following modular algorithm shows a remarkable improvement.

Input: $A = (a_{ij})_{1 \le i,j \le n} \in Z_{n \times n}$, with $|a_{ij}| \le b$ for all i, j.

Output: $\det A \in Z$.

$r := \lceil \log_2(2n^{n/2}b^n + 1) \rceil$;

choose r distinct prime numbers $m_1, \ldots, m_r$.

for $1 \le i \le r$ **do** compute $A \bmod m_i$.

for $1 \le i \le r$ **do**

compute $d_i \equiv \det A \bmod m_i$ using Gaussian elimination over $Z / \langle m_i \rangle$.

call the Chinese remainder algorithm to compute $d \in Z$ s.t. $d \equiv d_i \bmod m_i$, for $1 \le i \le r$.

return d.

To prove the algorithm works correctly, $\det A \equiv d_i \bmod m_i$ for $1 \le i \le r$. Hence, using the Chinese remainder algorithm, $\det A \equiv d \bmod m$, where $m = m_1 \cdot m_2 \ldots m_r$.

Since $m \ge 2^r > 2n^{n/2}b^n \ge 2|d|$, then actually $d = \det A$.

Example:

Using the same matrix of the previous example, and taking the first four prime numbers as moduli, the following modular determinants are computed.

$$\det A \equiv 0 \bmod 2 \qquad \det A \equiv 2 \bmod 3$$

$$\det A \equiv 2 \bmod 5 \qquad \det A \equiv -2 \bmod 7$$

In this case, $m = 2 \cdot 3 \cdot 5 \cdot 7 = 210$. The Chinese remainder problem $d \equiv d_i \bmod m_i$, $1 \le i \le 4$ has the solution $d \equiv -58 \bmod 210$. Using only the first three primes results in the erroneous solution $d = 2$.

A detailed cost analysis can be referred to in [1]. It shows that the modular method guarantees the running time will always be polynomial, considering word operations.

2.5.4 Partial fraction decomposition

The partial fraction decomposition problem has already been presented as an application of the Euclidean algorithm. A different approach is presented here where the Chinese remainder algorithm is used.

Let F be a field, $f_1, \ldots, f_r \in F[x]$ nonconstant monic and pairwise coprime polynomials, $e_1,\ldots, e_r \in N$ positive integers, and $f = f_1^{e_1} \cdot f_2^{e_2} \cdots f_r^{e_r}$. For another polynomial $g \in F[x]$ of degree less than $n = \deg f$, the *partial fraction decomposition* of the rational function $g / f \in F[x]$ with respect to the given factorization of the denominator f is:

$$\frac{g}{f} = \frac{g_{11}}{f_1} + \cdots + \frac{g_{1e_1}}{f_1^{e_1}} + \cdots + \cdots + \frac{g_{r1}}{f_r} + \cdots + \frac{g_{re_r}}{f_r^{e_r}},$$

with $g_{ij} \in F[x]$ of smaller degree than f_i , for all i, j.

The Chinese remainder algorithm is used in a partial solution of the problem by determining the *incomplete partial fraction decomposition*

$$\frac{g}{f} = \frac{c_1}{f_1^{e_1}} + \cdots + \frac{c_r}{f_r^{e_r}},$$

with $c_i \in F[x]$ and $\deg c_i < e_i \cdot \deg f_i$ for all i.

The method is as follows. By multiplying both sides of the incomplete decomposition by the denominator f, we obtain

$$g = c_1 \prod_{j \ne 1} f_j^{e_j} + \cdots + c_r \prod_{j \ne r} f_j^{e_j}, \qquad (2.1)$$

with unknowns $c_1, \ldots, c_r$. For any $i \le r$, each summand with the exception of the i^{th} one is divisible by $f_i^{e_i}$, whence $g \equiv c_i \prod_{j \ne i} f_j^{e_j} \bmod f_i^{e_i}$. Now each f_j is relatively prime to f_i hence invertible modulo $f_i^{e_i}$, thus

$$c_i \equiv g \prod_{j \ne i} f_j^{-e_j} \bmod f_i^{e_i},$$

which together with deg c_i < deg $f_i^{e_i}$ uniquely determines c_i . On the other hand, let c_i be defined as above, and let g^* be the right hand side in (2.1), then g^* is a polynomial of degree less than n = deg f and $g^* \equiv g \bmod f_i^{e_i}$ for all i. Now the Chinese remainder algorithm implies that $g^* \equiv g \bmod f$, and since both polynomials have degree less than n, they are equal.

It remains to obtain the full partial fraction decomposition from the incomplete one. Successive division of the obtained c_i's by their respective denominators completes this task. That is, if $c_i = q \cdot f_i + r$, with deg r < deg f_i , then $c_i / f_i^e = q / f_i^{e-1} + r / f_i^e$.

Example:

Let $\frac{f}{g} = \frac{x^3 + 4x^2 - x - 2}{x^4 - x^2}$ with partial fraction decomposition $\frac{1}{x} + \frac{2}{x^2} + \frac{1}{x-1} + \frac{-1}{x+1}$, with respect to the factorization $x^2(x-1)(x+1)$. Here, $m_1 = x^2$, $m_2 = x - 1$, $m_3 = x + 1$, and $f = m_1 \cdot m_2 \cdot m_3$. Computing the remainders gives:

$$r_1 = g \bmod x^2 = -x - 2, \qquad r_2 = g \bmod x - 1 = 2, \qquad r_3 = g \bmod x + 1 = 2.$$

Then, applying the Chinese remainder algorithm, the terms $c_i \equiv g \prod_{j \neq i} f_j^{-e_j} \bmod f_i^{e_i}$ are obtained.

$$\begin{aligned}
c_1 &\equiv r_1 \cdot (m/m_1)^{-1} &&= (-x-2)(x^2 - 1)^{-1} &&\equiv (-x - 2)(-1)^{-1} &&= x + 2 \bmod x^2,\\
c_2 &\equiv r_2 \cdot (m/m_2)^{-1} &&= 2(x^3 + x^2)^{-1} &&\equiv 2 \cdot 2^{-1} &&= 1 \bmod x - 1,\\
c_3 &\equiv r_3 \cdot (m/m_3)^{-1} &&= 2(x^3 - x^2)^{-1} &&\equiv 2 \cdot (-2)^{-1} &&= -1 \bmod x + 1.
\end{aligned}$$

Hence, the incomplete decomposition is:

$$\frac{x^3 + 4x^2 - x - 2}{x^4 + x} = \frac{x+2}{x^2} + \frac{1}{x-1} + \frac{-1}{x+1}$$

The complete partial fraction decomposition results from successive division of the first numerator by its denominator, which gives $x + 2 = 1 \cdot x + 2$, thus giving the required result.

2.6 GRÖBNER BASES [1]

In this section, Gröbner bases are presented. These offer an important algorithmic approach to deal with polynomials in several variables.

2.6.1 Basic definitions and notation

In this section, familiarity with the basic terminology of multivariate polynomials is required. A short introduction to this material can be found in the section on "polynomial

rings" at the beginning of this thesis. Some of the required material is reviewed in this section with a different notation that will be useful in this text.

- Let F be a field, $R = F[x_1, \ldots, x_n]$ a polynomial ring in n variables over F, and $f_1, \ldots, f_s \in R$. The polynomials $f_1, \ldots, f_s$ *generate* (or form a *basis* of) the ideal I if:

$$I = \langle f_1, \ldots, f_s \rangle = \{ \sum_{1 \le i \le s} q_i f_i \mid q_i \in R \}$$

- The *variety of I* is the set:

$$V(I) = \{ u \in F^n \mid f(u) = 0 \ \forall f \in I \} = \{ u \in F^n \mid f_1(u) = \ldots = f_s(u) = 0 \}$$

Example:

Let $f_1 = x^2 + y^2 - 2$ and $f_2 = y - 1$ in $R[x, y]$, and $I = \langle f_1, f_2 \rangle$. Then

$V(I) = \{ (u, v) \in R^2 \mid u^2 + v^2 - 2 = v - 1 = 0 \} = \{ (u, 2) \in R^2 \mid u^2 = 1 \} = \{ (-1, 2), (1, 2) \}$ is the intersection of the circle $V(x^2 + y^2 - 2)$ with the line $V(y - 1)$.

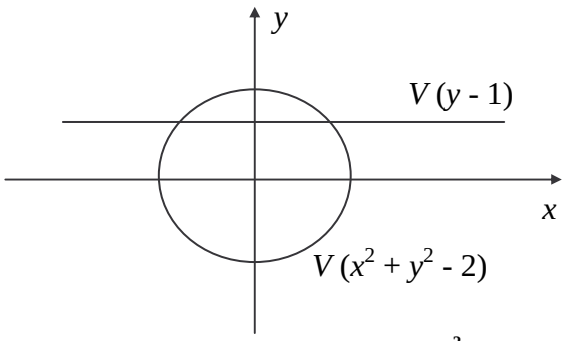


Figure 2-3. A circle and a line in R^2.

- A monomial is an element of R of the form: $x_1^{a_1} \cdot x_2^{a_2} \cdots x_n^{a_n}$, $a_i \in N$. It is denoted as $\boldsymbol{x}^{\alpha}$ where the vector $\alpha = (a_1, \ldots, a_n) \in N^n$.

Ordering the terms of multivariate polynomials is essential in dealing with them. Many techniques are available. Let $\alpha = (a_1, \ldots, a_n)$, $\beta = (b_1, \ldots, b_n) \in N^n$, then, two commonly used ordering methods, or *monomial orders* are:

- *Lexicographic order*: $\alpha <_{\text{lex}} \beta$ iff for the minimum i such that $a_i \neq b_i$, $a_i < b_i$.
- *Total lexicographic order*: $\alpha <_{\text{tlex}} \beta$ iff $\sum_{1 \le i \le n} a_i < \sum_{1 \le i \le n} b_i$ or otherwise, if $\sum_{1 \le i \le n} a_i = \sum_{1 \le i \le n} b_i$ then $\alpha <_{\text{lex}} \beta$.

Example:

Let $f = 4xyz^2 + 4x^3 - 5y^4 + 7xy^2z \in Q[x, y, z]$. Note that $x > y > z$ since $x \equiv x_1, y \equiv x_2, z \equiv x_3$.

Lexicographic order: $f = 4x^3 + 7xy^2z + 4xyz^2 - 5y^4$.

Total Lexicographic order: $f = 7xy^2z + 4xyz^2 - 5y^4 + 4x^3$.

Let $f = \sum_\alpha c_\alpha \boldsymbol{x}^\alpha \in R$ be a nonzero polynomial, and < an ordering technique, then:

- The *degree* of *f*, $\deg(f) = \max\{ \alpha \in N^n \mid c_\alpha \neq 0 \}$, where max is with respect to <.
- The *leading term* of *f* $\mathrm{lt}(f) = c_{\deg(f)} \boldsymbol{x}^{\deg(f)} \in R$.

2.6.2 Multinomial division with remainder

Division with remainder, needed in computing Gröbner bases, solves the following problem. Given a polynomial *f* and a set of polynomials $G = \{g_1, \ldots, g_s\}$, it is required to express *f* as follows

$$f = q_1 g_1 + \ldots + q_s g_s + r,\ q_1, \ldots, q_s, r \in R$$

The notation is: $r = f \text{ rem } (g_1, \ldots, g_s) = r \text{ rem } G$, which is the remainder.

The process of the division is illustrated hereunder by examples. The ordering technique used will be the lexicographic order.

Example (1):

$f = xy^2 + 1$, $g_1 = xy + 1$, $g_2 = y + 1$.

	$xy + 1$ $\quad y + 1$
$xy^2 + 1$	y
$-(xy^2 + y)$	
$-y + 1$	-1
$-(-y - 1)$	
2	

The first difference from the univariate division is that there are two divisors instead of one. The quotient of the leading term got in each step is recorded in the column below the respective divisor. In the last line, 2 is not divisible by any of the leading terms of g_1 or g_2, and the division process stops. Hence $f = y \cdot g_1 \ -1 \cdot g_2 + 2$.

The division procedure is not deterministic. The quotients and remainder need not be unique.

	$xy + 1$ $\quad y + 1$
$xy^2 + 1$	xy
$-(xy^2 + xy)$	
$-xy + 1$	$-x$
$-(-xy - x)$	
$x + 1$	

Here, $f = 0 \cdot g_1 + (xy - x) \cdot g_2 + (x + 1)$.

Example (2):

$f = x^2y + xy^2 + y^2$, $g_1 = xy - 1$, $g_2 = y^2 - 1$.

	$xy - 1$	$y^2 - 1$	r
$x^2y + xy^2 + y^2$	x		
$-(x^2y - x)$			
$xy^2 + x + y^2$	y		
$-(xy^2 - y)$			
$x + y^2 + y$			x
$-x$			
$y^2 + y$		1	
$-(y^2 - 1)$			
$y + 1$			

Another difference from the univariate case appears in this example. In the third step, the leading term x is not divisible by any of the leading terms of g_1 or g_2. It is moved to the remainder column and the division process continues further. The result is $f = (x + y)\cdot f_1 + 1\cdot f_2 + (x + y + 1)$.

Again, non-determinism may be checked by verifying that $f = x\cdot f_1 + (x + 1)\cdot f_2 + (2x + 1)$.

To make the division process deterministic, one can order the g_i's and always choose the smallest possible i in each division step.

Example (3):

$f = xy^2 - x$, $g_1 = xy + 1$, $g_2 = y^2 - 1$.

Ordering the g_i's in the given order, we get:

	$xy + 1$	$y^2 - 1$
$xy^2 - x$	y	
$-(xy^2 + y)$		
$-x - y$		

And $f = y \cdot f_1 + 0 \cdot f_2 + (-x - y)$.

However, starting by g_2 in the division procedure, f is expressed as $f = 0 \cdot f_1 + x \cdot f_2 + 0$, so that $f \in \langle g_1, g_2 \rangle$, the ideal generated by g_1 and g_2.

2.6.3 Hilbert's basis theorem and Gröbner bases

This section presents most of the material required to compute Gröbner bases. For conciseness, only definitions and theorems are given without any proofs.

Let $R = F[x]$, the Euclidean domain of univariate polynomials. One can show that $\langle g_1, \ldots, g_s \rangle = \langle \gcd(g_1, \ldots, g_s) \rangle$. Dividing f by g with remainder yields q, $r \in F[x]$ with $f = qg + r$ and $\deg(r) < \deg(g)$. Then $f \in \langle g \rangle \Leftrightarrow r = 0$.

Generalizing this property of *ideal membership* to the case of multivariate polynomials require first a unique remainder from the division process. As the last example of the previous section shows, this is not the case. However, this requirement can be realized by choosing a special basis G for each ideal such that the remainder on division by that basis is unique. Such a basis is called a *Gröbner basis* and is defined in this section.

- A *monomial ideal* $I \subseteq R$ is an ideal generated by monomials in R so that there exists a subset $A \subseteq N^n$ with $I = \langle \boldsymbol{x}^A \rangle = \langle \{\boldsymbol{x}^\alpha \mid \alpha \in A\} \rangle$.

For $\beta \in N^n$, $\boldsymbol{x}^\alpha \in I \Leftrightarrow \exists\ \alpha \in A \mid \boldsymbol{x}^\alpha$ divides $\boldsymbol{x}^\beta$.

- For any subset $G \subseteq R$ different from $\varnothing$ and $\{0\}$, the set of *leading terms of G*,

$$\mathrm{lt}(G) = \{\mathrm{lt}(G) \mid g \in G\}.$$

- *Hilbert's basis theorem*

Every ideal $I \in R = F[x_1, \ldots, x_n]$ is finitely generated. More precisely, there exists a finite subset $G \subseteq I$ such that $\langle G \rangle = I$ and $\langle \mathrm{lt}(G) \rangle = \langle \mathrm{lt}(I) \rangle$.

A proof of the theorem can be referred to in [1].

- Let $<$ be a monomial order and $I \subseteq R$ an ideal. A finite set $G \subseteq I$ is a *Gröbner basis* for I with respect to $<$ if $\langle G \rangle = I$ and $\langle \mathrm{lt}(G) \rangle = \langle \mathrm{lt}(I) \rangle$.

Thus Hilbert's theorem implies that every ideal in $F[x_1, \ldots, x_n]$ has a Gröbner basis.

The ideal membership problem can now be solved, as stated earlier. If G is a Gröbner basis for the ideal $I \subseteq R$ with respect to a monomial order $<$, and $f \in R$, then $f \in I$ iff f rem $G = 0$.

Note that Hilbert's theorem does *not* say that *every* subset $G \subseteq I$ that generates I, is such that $\langle \mathrm{lt}(G) \rangle = \langle \mathrm{lt}(I) \rangle$. It can happen that $\langle G \rangle = I$ but $\langle \mathrm{lt}(G) \rangle \neq \langle \mathrm{lt}(I) \rangle$. This is illustrated in the following example.

Example:

Let $g = x^3 - 2xy$, and $h = x^2y - 2y^2 + x \in Q[x, y]$. The used ordering is $<_{\mathrm{tlex}}$, and $G = \{g, h\}$. Let $I = \langle G \rangle$. Then $x^2 = -yg + xh$. and $x^2 \in \langle \mathrm{lt}(I) \rangle$ but $x^2 \notin \langle \mathrm{lt}(G) \rangle = \langle x^3, x^2y \rangle$.

What happened here is that the linear combination $a\boldsymbol{x}^{\alpha}g + b\boldsymbol{x}^{\beta}h$, ($g, h \in G$ and $\alpha,\beta \in N^n$) yielded a polynomial whose leading term is not divisible by any of lt(G), due to cancellation of leading terms. When such cancellation occurs, it necessarily comes from *S-polynomials*.

- Let $g, h \in R$ be nonzero, deg(g) = $\alpha = (a_1, \ldots, a_n)$, deg($h$) = $\beta = (b_1, \ldots, b_n)$, and γ = ($\max\{a_1,b_1\}, \ldots, \max\{a_n,b_n\}$). The *S-polynomial* of g and h is

$$S(g, h) = (\boldsymbol{x}^{\gamma} / \mathrm{lt}(g))g - (\boldsymbol{x}^{\gamma} / \mathrm{lt}(h))h.$$

Since $\boldsymbol{x}^{\gamma} / \mathrm{lt}(g), \boldsymbol{x}^{\gamma} / \mathrm{lt}(h) \in R$, we have $S(g, h) \in \langle g, h \rangle$.

Example:

Again, let $g = x^3 - 2xy$, and $h = x^2y - 2y^2 + x \in Q[x, y]$. Here, $\alpha = (3, 0)$, $\beta = (2, 1)$, $\gamma = (3, 1)$. Then, $S(g, h) = (x^3y/x^3)g - (x^3y/x^2y)h = -x^2$, the term which caused cancellation in this example.

- *Gröbner basis test*

A finite set $G = \{g_1, \ldots, g_s\} \subseteq R$ is a Gröbner basis of the ideal $\langle G \rangle$ if and only if:

$$S(g_i , g_j) \text{ rem } (g_1, \ldots, g_s) = 0 \text{ for } 1 \le i \le j \le s.$$

Again, [1] presents the proof.

2.6.4 Computation of Gröbner bases

Now, the Gröbner bases may be computed. A simplified version of *Buchberger's algorithm* for computing a Gröbner basis is presented in [1]. The algorithm follows.

Input: $f_1, \ldots, f_s \in R = F[x_1, \ldots, x_n]$ and a monomial order <.

Output: Gröbner basis $G \subseteq R$ for the ideal $I = \langle f_1, \ldots, f_s \rangle$ w.r.t. <, with $f_1, \ldots, f_s \in G$.

$G := \{f_1, \ldots, f_s\}$

while true **do** // infinite loop

  $U := \varnothing$

  order the elements of G as $g_1, \ldots, g_t$

  for $1 \le i < j \le t$ **do**

    $r := S(g_i , g_j)$ rem $(g_1, \ldots, g_t)$

    if $r \ne 0$ **then** $U := U \cup \{r\}$

  if $U = \varnothing$ **then return** G **else** $G := G \cup U$.

Notes on the complexity of this algorithm are found in [1] and [2]. Earlier mathematicians mention a doubly exponential upper bound (2^{2^n}) in the degrees of polynomials in the basis.

Mayr (1996) was able to provide an exponential space algorithm for computing a Gröbner basis.

Example:

For the third time, let $f_1 = x^3 - 2xy$, and $f_2 = x^2y - 2y^2 + x \in Q[x, y]$, with total lexicographic order $<_{tlex}$ and $y <_{tlex} x$.

Initially, $G = \{f_1, f_2\}$ is not a Gröbner basis since $f_3 = S(f_1, f_2)$ rem $(f_1, f_2) = -x^2 \neq 0$. Hence f_3 is added to the basis and $G = \{f_1, f_2, f_3\}$ with $S(f_1, f_2)$ rem $(f_1, f_2, f_3) = 0$

Next, $S(f_1, f_3) = 1 \cdot f_1 - (-x) \cdot f_3 = -2xy$, then $S(f_1, f_3)$ rem $(f_1, f_2, f_3) = -2xy = f_4$.

After adding f_4 to the basis, $S(f_1, f_3)$ rem $(f_1, f_2, f_3, f_4) = 0$.

Now, $S(f_1, f_4) = y \cdot f_1 - (-1/2x^2) \cdot f_4 = -2xy^2 = y \cdot f_4$ so that $S(f_1, f_4)$ rem $(f_1, f_2, f_3, f_4) = 0$.

But $S(f_2, f_3) = 1 \cdot f_2 - (-y) \cdot f_3 = -2y^2 + x$, and $S(f_2, f_3)$ rem $(f_1, f_2, f_3, f_4) = -2y^2 + x = f_5$.

After adjoining f_5 to the basis, one can check that

$$S(f_i, f_j) \text{ rem } (f_1, f_2, f_3, f_4, f_5) = 0 \text{ for } 1 \leq i < j \leq 5.$$

Thus $\{(f_1, f_2, f_3, f_4, f_5\}$ forms a Gröbner basis.

2.7 APPLICATIONS OF GRÖBNER BASES [1]

A direct application of Gröbner bases is the ideal membership problem. They are also essential in geometric theorem proving, implicitization problems and for solving systems of multivariable polynomial equations.

2.7.1 Ideal membership problem

This application is the most natural one and has already been presented in the text. If G is a Gröbner basis for the ideal $I \subseteq R$ with respect to a monomial order $<$, and $f \in R$, then $f \in I$ if and only if f rem $G = 0$.

2.7.2 Geometric theorem proving

Geometric theorems can often be formulated in terms of polynomial equations. This is a kind of problem that can be solved using Gröbner bases. The following example illustrates how this formulation can help in “automatic” proofs of theorems in geometry.

Example:

A well-known geometric theorem says that the three medians of a triangle intersect at one point, and that the intersection point (the center of gravity) trisects each median. Refer to the figure below.

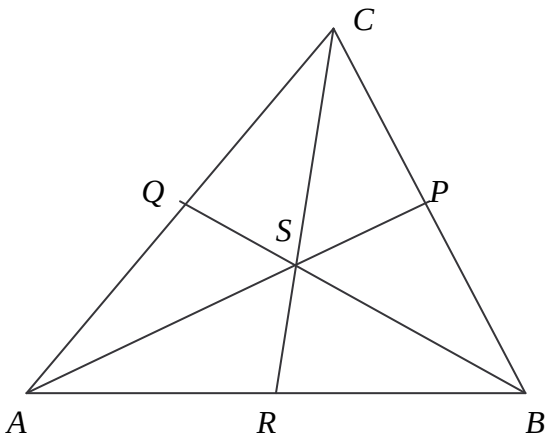


Figure 2-4. The three medians of a triangle intersect at the center of gravity.

Since the assumptions and the conclusion of the theorem are invariant under translation, rotation and scaling, it may be assumed that two of the vertices of the triangle are $A = (0, 0)$ and $B = (1, 0)$, and the third point is $C = (x, y) \in R^2$. Then the midpoints of the three edges BC, AC and AB are $P = ((x+1)/2, y/2)$, $Q = (x/2, y/2)$ and $R = (1/2, 0)$, respectively.

Let $S = (u, v)$ be the intersection point of the two medians AP and BQ. The condition that S lies on AP is equivalent to saying that AS and AP have the same slope, so that $u/v = (x+1)/y$ or

$$f_1 = uy - v(x + 1) = 0.$$

Similarly, the condition that S lies on BQ can be expressed as

$$f_2 = (u - 1)y - v(x - 2) = 0.$$

Now, the claims are that S lies on the third median CR, or

$$g_1 = -2(u - x)y - (v - y)(1 - 2x) = -2uy - (v - y) + 2vx = 0,$$

and that S trisects each of the three medians, so that

$$(u, v) = AS = 2SP = (x + 1 - 2u, y - 2v),$$

$$(u - 1, v) = BS = 2SQ = (x - 2u, y - 2v),$$

$$(u - x, v - y) = CS = 2SR = (2u - 1, 2v),$$

or equivalently,

$$g_2 = 3u - x - 1 = 0 \text{ and } g_3 = 3v - y = 0.$$

And the theorem is equivalent to "$f_1 = f_2 = 0 \Rightarrow g_1 = g_2 = g_3 = 0$", where $f_1, f_2 \in R[u, v, x, y]$ are the hypotheses polynomials, and the conclusions yield $g_1, g_2, g_3 \in R[u, v, x, y]$.

In general, a geometric problem can be formulated as a set of hypothesis polynomials $f_1, \ldots, f_s$ in $R[x_1, \ldots, x_n]$ and one or several conclusion polynmomials $g \in R[x_1, \ldots, x_n]$ and the theorem is true if and only if $V(f_1, \ldots, f_s) \subseteq V(g)$. In particular, the theorem can be proved by showing that $g \in \langle f_1, \ldots, f_s \rangle$.

Back to the example, consider the ideal $\langle f_1, f_2 \rangle$ for which it is required to compute a Gröbner basis. Using the lexicographic monomial order $<_{\text{lex}}$, and $u >_{\text{lex}} v > x > y$, we have

$$f_1 = uy - vx - v, \qquad f_2 = uy - vx + 2v - y.$$

With respect to this monomial order,

$$S(f_1, f_2) = f_1 - f_2 = -3v + y = -g_3 \neq 0.$$

After adding g_3 to the basis,

$S(f_1, g_3)$ rem $(f_1, f_2, g_3) = -uy^2 - 3v^2x - 3v^2$ rem $(f_1, f_2, g_3) = 0$,

$S(f_1, f_2)$ rem $(f_1, f_2, g_3) = -uy^2 - 3v^2x + 6v^2 - 3vy$ rem $(f_1, f_2, g_3) = 0$.

Thus $\{f_1, f_2, g_3\}$ is a Gröbner basis, i.e. $g_3 \in \langle f_1, f_2 \rangle$, and the third conclusion is proved. One can easily show the same for g_1 and g_2.

2.7.3 Implicitization

Let $f_1, \ldots, f_n \in F[t_1, \ldots, t_m]$, and consider the hyperplane V given in parametric form by

$$x_1 = f_1(t_1, \ldots, t_m),$$

$$\ldots$$

$$x_n = f_n(t_1, \ldots, t_m),$$

so that V can be "explicitly" described as $V = \{ a \in F^n \mid \exists\, b \in F^m, a = (f_1(b), \ldots, f_n(b))\}$. The task is now to find polynomials $g_1, \ldots, g_s \in F[x_1, \ldots, x_n]$ such that V has the "implicit" representation $V = V(I)$, where $I = \langle g_1, \ldots, g_s \rangle$.

To solve the implicitization problem, consider $J = \langle x_1 - f_1, \ldots, x_n - f_n \rangle \subseteq F[t_1, \ldots, t_m, x_1, \ldots, x_n]$, order the variables so that $t_1 > \ldots > t_m > x_1 > \ldots > x_n$, and compute a Gröbner basis G for J with respect to $<_{\text{lex}}$. Then some $g \in G$ will depend only on $x_1, \ldots, x_n$. These are candidates for the g_i's.

Example:

The twisted cubic C is given by the parametric equations

$$x = t, \quad y = t^2, \quad z = t^3.$$

The Gröbner basis J with respect to $t > z > y > x$ is $\{t - x, z - x^3, y - x^2\}$. Thus the implicitization for C is $g_1 = y - x^2$, $g_2 = z - x^3$. In fact the curve is the intersection of the two surfaces g_1 and g_2 in 3-D space. In the figure below, C is the black bold curve.

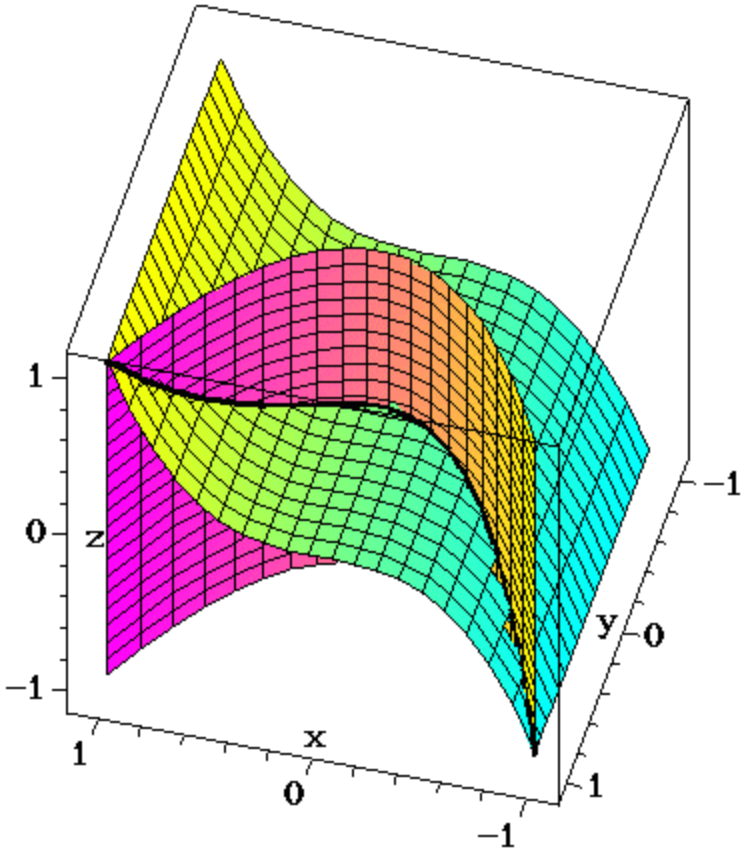


Figure 2-5. The twisted cubic (thick line) in R^3.

2.7.4 Solving systems of polynomial equations

Given is a system of polynomials $f_1, \ldots, f_m \in F[x_1, \ldots, x_n]$ and want to find the set of solutions

$$V = \{a \in F^n \mid f_1(a) = 0, \ldots, f_m(a) = 0\}.$$

The Gröbner basis with respect to lexicographic order and $x_1 <_{\text{lex}} \ldots < x_n$ will often include some polynomials only containing x_1, some only with x_1, x_2, and so on. These can be solved by back substitution: first for x_1, then for x_2, and so on.

Example:

Let $f = (y^2 + 6)(x - 1) - y(x^2 + 1)$, $g = (x^2 + 6)(y - 1) - x(y^2 + 1)$, and $f, g \in C[x, y]$. It is required to find all common roots of the system of polynomial equations $f = g = 0$.

A Gröbner basis of $I = \langle f, g \rangle$ w.r.t. lexicographic order $<_{lex}$ and $x <_{lex} y$ consists of f, g, and the three polynomials

$$h = -f - g = x^2 - 5x + y^2 - 5y + 12, \qquad p = y^2x - 5yx + 6x + y^3 - 6y^2 + 11y - 6,$$
$$q = y^4 - 6y^3 + 15y^2 - 26y + 24.$$

The last polynomial contains only the variable y, and, by factoring it, $q = (y - 2)(y - 3)(y^2 - y + 4)$ and hence, $V(q) = \{2, 3, 1/2 \pm i\,(\sqrt{15})/2\}$.

Substituting $y = 2$ in h and p, it results that $p(x, 2) = 0$ for all x. And $h(x, 2) = x^2 - 5x + 6$, hence two common zeros are (2, 2) and (3, 2). by substituting $y = 3$ and $y = 1/2 \pm i\,(\sqrt{15})/2$ in $h = p = 0$ then solving for x, the other four points can be found:

$$\{(2, 3), (3, 3), (\frac{1 \pm \sqrt{15}i}{2}, \frac{1 \mp \sqrt{15}i}{2})\}.$$

2.8 GENERAL APPLICATION AREAS OF COMPUTER ALGEBRA

The three typical steps in mathematical applications are:

- creating a mathematical model for the problem at hand,
- solving the model mathematically, and
- interpreting the solution in the original problem.

Virtually any application where the results are required to be exact or symbolic can be considered as an application of computer algebra. To mention some fields involved with symbolic computation, we have:

- Physics: plasma physics, physics of fluids, electron optics, non-linear optics, molecular physics, electronics.
- Mechanics: celestial mechanics, calculation of orbits, gravitational fields.
- Mathematics: number theory, group theory, computational geometry, numerical analysis.
- Other areas: chemistry, biology, robotics, and economy.

[Cf.: Richard Liska, `http://www-troja.fjfi.cvut.cz/~liska/ca/`]

An illustration of a few application areas follows hereunder.

2.8.1 Celestial mechanics

The famous example of Delauny. He calculated the orbit of the moon under the influence of the sun and a non-spherical earth with a tilted ecliptic. His work took 20 years to complete

and was published in 1867. Recalculation on a small computer in 1970 took only 20 hours of CPU time, and it showed hand calculations to be correct to 9 decimal places. [1]

2.8.2 Geometric theorem proving

There are several computer algebra approaches to proving theorems in Euclidean geometry that can be stated as polynomial equations. An example is: "*The altitude pedal of the hypotenuse of a right-angled triangle and the midpoints of the three sides of the triangle are co-circular.*"

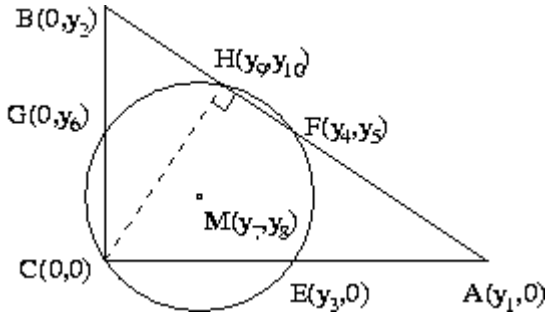


Figure 2-6. Example of a geometric theorem proving.

The hypotheses of this geometric statement, describing a correct drawing of the corresponding figure, are polynomial equations in the coordinates of the points in the figure. The same holds for the conclusion. [2]

Hypotheses:

$h_1 \equiv 2y_3 - y_1 = 0$ (E is the midpoint of $\overline{AC}$),

$h_2 \equiv (y_7 - y_3)^2 + (y_8)^2 - (y_7 - y_4)^2 - (y_8 - y_5)^2 = 0$ ($\overline{EM}$ and $\overline{FM}$ are equally long),

$\vdots$

$h_m \equiv \ldots$

Conclusions:

$c \equiv (y_7 - y_3)^2 + (y_8)^2 - (y_7 - y_9)^2 - (y_8 - y_{10})^2 = 0$ ($\overline{EM}$ and $\overline{HM}$ are equally long),

2.8.3 Computational algebraic geometry

This field includes many applications, among which implicitization and parameterization only are presented.

There are two standard ways to represent planar curves: parametric and implicit forms. Both forms have their use, and some problems are modeled using one form or the other. There exist algorithms that convert one form to the other, as illustrated below.

The *tacnode* illustrated in the next figure has the implicit equation

$$f(x, y) = 2x^4 - 3x^2y + y^4 - 2y^3 + y^2 = 0$$

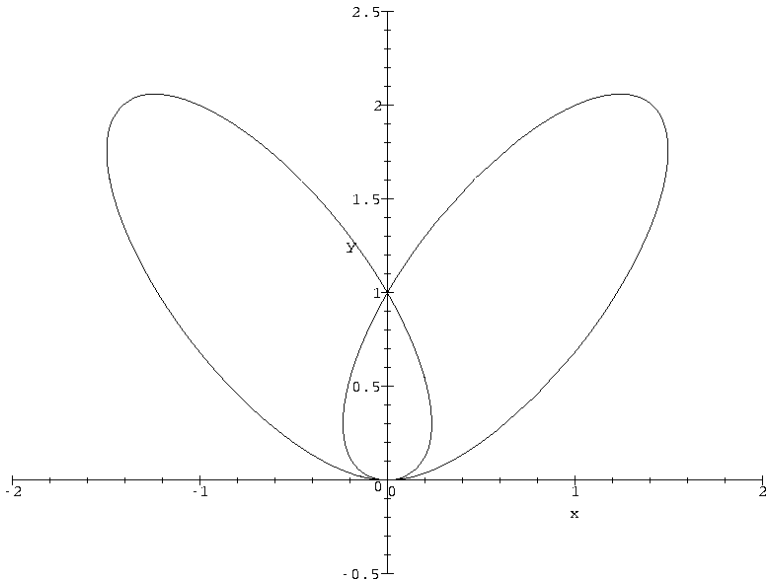


Figure 2-7. The tacnode curve.

Computer algebra offers algorithms that convert to and from the parametric equations: [2]

$$x(t) = \frac{t^3 - 6t^2 + 9t - 2}{2t^4 - 16t^3 + 40t^2 - 32t + 9} \quad , \quad y(t) = \frac{t^2 - 4t + 4}{2t^4 - 16t^3 + 40t^2 - 32t + 9}$$

2.8.4 Robotics

The next figure shows a very simple robot, one-armed with two joints. The arm is fixed at one end with a joint to a point, say the origin of the Cartesian plane. The other joint in the middle rotates freely. The distance between the two joints is 2, and the distance from the second joint to the endpoint is 1. The joint between the two arms is at position (x,y) and the endpoint at position (z,w).

Furthermore, there is a line $mt + c = 0$ with some fixed parameters m,c. A simple question is: can the robot reach the line?

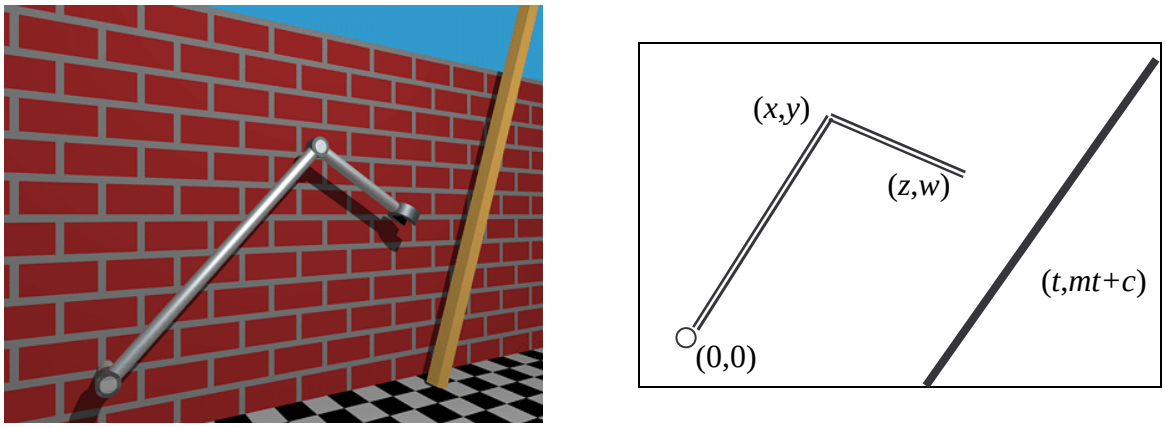


Figure 2-8. A two-segment robot. (3D view and schematic).

The possible positions $(x,y,z,w) \in R^4$ of the robot are characterized by the algebraic equations

$$(x^2 + y^2) = 4, \quad (z - x)^2 + (w - y)^2 = 1,$$

and an answer to the question is either a quadruple (x,y,z,w) satisfying these equations together with the line equation $w = mz + c$, or a proof that no such quadruple exists. [2] This is the kind of problems that can be used with the help of Gröbner bases.

2.8.5 Chemistry

The cyclohexane molecule C_6H_{12} is a hydrocarbon consisting of six carbon atoms connected to each other in a cycle, and twelve hydrogen atoms, two attached to each carbon atom.

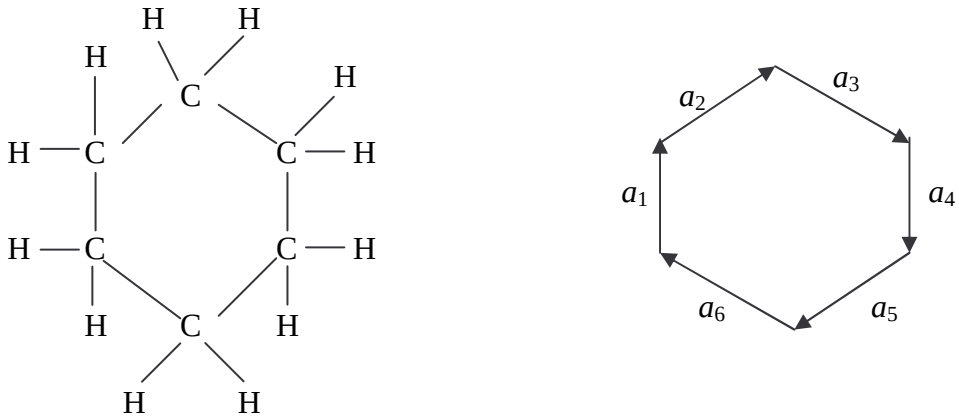


Figure 2-9. The structure formula for cyclohexane and the carbon bonds.

The figures above show the structure formula for the cyclohexane and the orientation given to the bonds $a_1, \ldots, a_6$. The four bonds of one carbon atom (two bonds to adjacent carbon atoms and two to hydrogen atoms) are arranged in the Euclidean space in the form of a tetrahedron, with the carbon in the center and its bonds pointing to the four corners. The angle between

any two bonds is $cos^{-1}(-1/3)$. Now, the problem is to determine the conformation (i.e. the configuration of the molecule) of the cyclohexane.

The unknowns are the orientations of the six bonds in three-space: $a_1, \ldots, a_6 \in R^3$, where a_i is a three-component vector. The following equations are obtained:

$$\vec{a}_1 \cdot \vec{a}_1 = \vec{a}_2 \cdot \vec{a}_2 = \cdots = \vec{a}_6 \cdot \vec{a}_6 = 1,$$

$$\vec{a}_1 \cdot \vec{a}_2 = \vec{a}_2 \cdot \vec{a}_3 = \cdots = \vec{a}_6 \cdot \vec{a}_1 = \frac{1}{3},$$

$$\vec{a}_1 + \vec{a}_2 + \cdots + \vec{a}_6 = 0.$$

The $\cdot$ is the usual scalar product (dot product) of vectors.

These equations can be solved with a computer algebra system, yielding the conformations observed by chemists and illustrated below. [1]

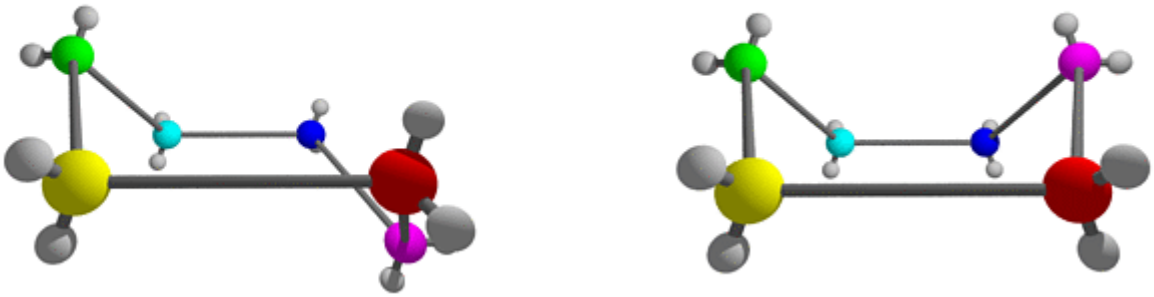


Figure 2-10. The chair and boat conformations of cyclohexane.

Chapter 3: THE IMPLICITIZATION PROBLEM

In this chapter, we review some of the features of the implicitization problems [10] [11] [13]. Section 3.1 contains some definitions with a comparison on the use of implicit and parametric forms. Then, section 3.2 reviews some needed results from algebra, namely resultants [1] [3] and homogeneous systems of linear equations [27]. Section 3.3 is basic in this thesis. It reviews some implicitization methods with emphasis on describing some existing algorithms. Moreover, algorithms are devised when they are not provided in the literature and afterwards implemented in chapter 4. Finally, newly suggested implicitization methods are devised and also subsequently implemented in chapter 4. Complexity of algorithms as related to the mentioned algorithms is addressed.

3.1 INTRODUCTION

3.1.1 Preliminaries [2]

- *Algebraic geometry*, also known as *computational geometry*, is the study of geometric objects defined as the zeros of polynomial equations. Using the algebra of polynomial rings, many of the techniques in algebraic geometry become computationally feasible.
- For any field F, the *n-dimensional affine* space over F is defined as:

$$F^n = \{(a_1, \ldots, a_n) \mid a_i \in F\}.$$

Thus, F^2 is the affine plane over F.

- An *affine plane algebraic curve* C in F^2 is the set of zeros of a polynomial $f(x, y) \in F[x, y]$, i.e.

$$C = \{(a_1, a_2) \mid f(a_1, a_2) = 0, (a_1, a_2) \in F^2\}.$$

f is called a *defining polynomial* of the curve C.

- The affine curve C in F^2 defined by the polynomial $f(x, y)$ is called *parameterizable*, iff there are functions $g(t)$ and $h(t)$ such that:

a. for almost all (i.e. for all but a finite number of exceptions) $t_0 \in F$, $(g(t_0), h(t_0))$ is a point on C,

b. for almost every point (x_0, y_0) on C, there is a $t_0 \in F$ such that $(x_0, y_0) = (g(t_0), h(t_0))$.

- The *parametric representation* of a plane curve is a mapping from the real line domain ($t \in R$) to the plane domain $(x, y) \in R^2$.

- The *implicitization problem* is the process of converting curves and surfaces from parametric form into implicit form.

3.1.2 Parametric and implicit forms

In geometric modeling and computer aided design systems, curves and surfaces can be represented either parametrically or implicitly. For example, the unit sphere can be written as $x^2 + y^2 + z^2 - 1 = 0$ in implicit form, or written as

$$x = \frac{2s}{1+s^2+t^2}, y = \frac{2t}{1+s^2+t^2}, z = \frac{1-s^2-t^2}{1+s^2+t^2}$$

in parametric form. Implicit representations of curves and surfaces provide a straightforward way to detect whether or not a point lies on a curve or surface. Parametric representations of curves and surfaces are very convenient for shape control and for rendering. Both the parametric form and the implicit form are important, but many curve and surface design systems start from parametric representations. Implicitization is thus required to convert the parametric form into implicit form [10] [13].

Generation of plane curves

To illustrate the usefulness of parametric equations in generating points along curves, the simple example of the plot of an ellipse is presented. Consider the implicit equation of the ellipse $4x^2 + 9y^2 = 36$. By expressing y as an explicit equation in x, i.e., $y = \frac{1}{3}\sqrt{36-4x^2}$, the ellipse is plotted at regular x-intervals equal to 0.05 using Matlab. Notice that the plot is a vector of points computed at the various values of x. The discontinuities appear clearly as the absolute value of the slope of the curve exceeds 1, near $x = 2.7$. These are due to non-regularly spaced y-values corresponding to regularly spaced x-values.

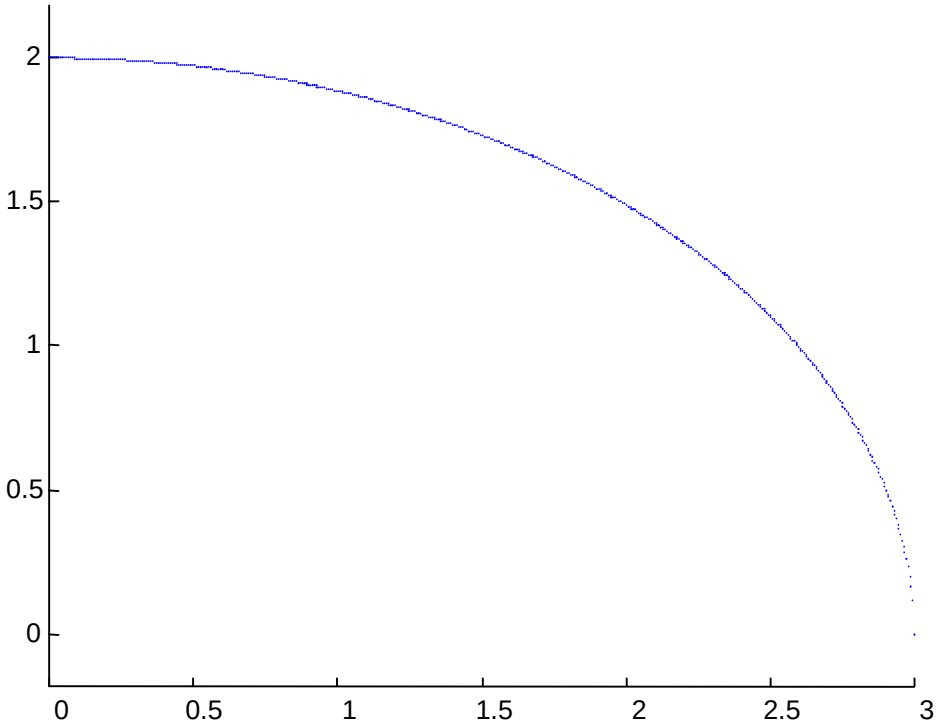


Figure 3-1. Explicit plot of an ellipse.

In the case of the parametric plot, the generated points are regularly spaced for regularly spaced *t*-values resulting in a smoother plot. Intervals of size 0.05 are used here also.

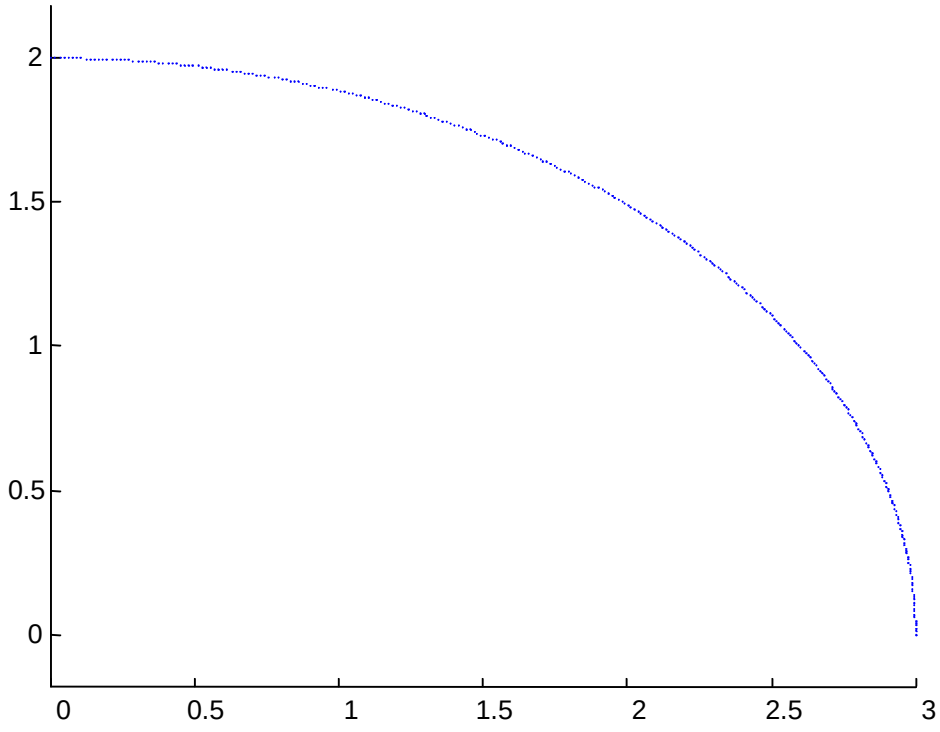


Figure 3-2. Parametric plot of an ellipse.

Generation of plane curves by rolling

In [9], an interesting problem in descriptive geometry is presented, where parametric representation is the start point.

If a "moving curve" C_m rolls, without slipping, along another "fixed curve" C_f, any fixed point Q attached to C_m describes a curve γ which is known as roulette. In the same reference, it is shown how the equations of C_m and C_f which generate a certain curve γ can be derived in parametric form, where their parameter is that of the given curve γ. Examples of roulette are the family of cycloid (epicycloid, hypocylcoid, …) and the family of trochoid (epitrochoid, hypotrochoid, …). It may be valuable to add implicitization analysis to the resulting roulettes.

The famous cycloid curve, which is the tracing of a point on the circumference of a circle rolling on a straight line, is given by the parametric equations:

$x = a(t - \sin t)$, $y = a(1 - \cos t)$, where a is the radius of the rolling circle

Its plot is shown in the next figure.

Later, the cycloid is treated from the point of view of implicitization, thus paving the way for other roulettes to be treated the same way.

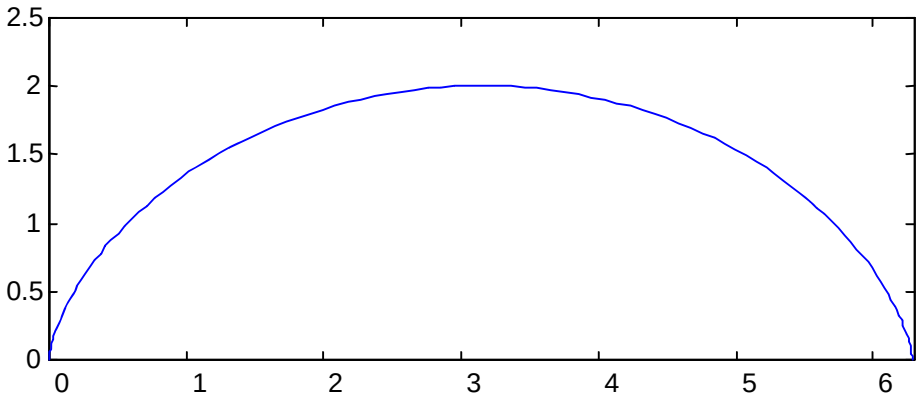


Figure 3-3. The cycloid.

Trigonometric curves [11] [12]

A trigonometric curve is a real plane curve where each coordinate is given parametrically by a trigonometric polynomial, that is, a truncated Fourier series of the form:

$$\sum_{0 \le k \le m} a_k \cos(k\theta) + b_k \sin(k\theta)$$

Trigonometric curves arise in various areas of mathematics, physics and engineering. This class of curves includes numerous classical curves such as limacon of Pascal, cardioid, trifolium, epicycloid, hypocycloid, etc, as special cases. (Note that some of these curves are the roulettes mentioned above in the generation of curves by rolling.) They also arise naturally in numerous areas such as linear differential equations, Fourier analysis, almost periodic functions (under the name of generalized trigonometric polynomials), representation of groups (utilizing its periodicity), electrical circuit analysis (Lissajous curves as often shown on oscilloscopes), fracture mechanics (as the caustic pattern appearing when a fractured material is shown by a laser beam), etc. The class includes all bounded polynomial curves (i.e. images of polynomial parameterization with bounded parameter interval). It is a subset of the class of rational curves (images of rational parameterization).

3.1.3 Variants of implicitization

The most naturally related problem is the reverse problem, namely that of parameterization of curves, i.e. finding a parametric form of an implicit equation [2]. However, the implicitization problem is itself divided into numerous subproblems, some of which far from a complete and satisfactory solution.

Several open problems and achievements in this area are presented in [21]. A brief overview highlights the following research areas:

a) Two-dimensional versus three-dimensional implicitization. A 2-D object is represented by two functions $x(t)$ and $y(t)$, while a 3-D object requires a third function $z(t)$.

b) Curves versus surfaces. Parametric representations of curves require functions in only one parameter. Surfaces are represented by two-parameter functions. The implicitization of surfaces is still a problem, far from having a complete solution [13].

c) Rational versus trigonometric functions. Rational parametric functions are of the form of a ratio of two polynomials such as $x = f(t)/g(t)$. Trigonometric functions are the form of truncated Fourier series presented in the previous section.

3.2 MORE RESULTS FROM ALGEBRA

In the following, some results of algebra are summarized.

3.2.1 Resultants [1] [3]

Let S be a commutative ring with identity. Let $A(x)$ and $B(x) \in S[x]$ be univariate polynomials of respective positive degrees m and n with coefficients in the ring S.

$A(x) = a_m x^m + a_{m-1}x^{m-1} + \ldots + a_0$, $\quad \deg(A) = m$, and

$B(x) = b_n x^n + b_{n-1}x^{n-1} + \ldots + b_0$, $\quad \deg(B) = n$,

- The *Sylvester matrix* of $A(x)$ and $B(x) \in S[x]$, denoted as $syl(A, B)$, is the following matrix of size $(m + n) \times (m + n)$ over S:

$$syl(A, B) = \begin{bmatrix} a_m & a_{m-1} & \cdots & a_0 & & & \\ & a_m & a_{m-1} & \cdots & a_0 & & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & & a_m & a_{m-1} & \cdots & a_0 \\ b_n & b_{n-1} & \cdots & \cdots & b_0 & & \\ & b_n & b_{n-1} & \cdots & \cdots & b_0 & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & b_n & b_{n-1} & \cdots & \cdots & b_0 \end{bmatrix}$$

(n staggered rows of coefficients of A; m staggered rows of coefficients of B)

In particular, the first n rows of the Sylvester matrix correspond to the polynomials $x^{n-1} A(x)$, $x^{n-2}A(x),\ldots, A(x)$, and the last m rows, to $x^{m-1} B(x)$, $x^{m-2}B(x),\ldots, B(x)$.

- The *resultant* of $A(x)$ and $B(x)$, denoted $res(A, B)$, is the determinant of the Sylvester matrix $syl(A, B)$, and thus is an element of S.

Since $syl(B, A)$ can be obtained by $m \cdot n$ row transpositions, we see that

$$\begin{aligned} res(B, A) &= det(syl(B, A)) \\ &= (-1)^{mn} det(syl(A, B)) \\ &= (-1)^{mn} res(A, B). \end{aligned}$$

The following properties will be used for some of the subsequent proofs.

3.2.1.1 ***Properties of resultants***

Lemma. [3]

Let S be a commutative ring with identity, and $A(x)$ and $B(x) \in S[x]$ be univariate polynomials of respective positive degrees m and n with coefficients in the ring S, then there exist polynomials $T(x)$ and $U(x) \in S[x]$ such that:

$$A(x) \cdot T(x) + B(x) \cdot U(x) = \text{res}\,(A, B),$$

where $\deg(T) < \deg(B) = n$ and $\deg(U) < \deg(A) = m$.

Proof.

Consider the Sylvester matrix of A and B:

$$M = syl(\mathrm{A}, \mathrm{B}) = \begin{bmatrix} a_m & a_{m-1} & \cdots & a_0 & & & \\ & a_m & a_{m-1} & \cdots & a_0 & & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & & a_m & a_{m-1} & \cdots & a_0 \\ b_n & b_{n-1} & \cdots & \cdots & b_0 & & \\ & b_n & b_{n-1} & \cdots & \cdots & b_0 & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & b_n & b_{n-1} & \cdots & \cdots & b_0 \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\} n \text{ rows} \\ \left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\} m \text{ rows} \end{matrix}$$

Let us create a new matrix M' from M by following elementary matrix operations:

1. First, multiply the i^{th} column by x^{m+n-i} and add to the last column of M.
2. All but the last column of M' are the same as those of M.

By definition,

$$res(A, B) = det(M) = det(M').$$

We observe that the matrix M' is as follows:

$$syl(A,B)=\left[\begin{array}{cccccc}
a_m & a_{m-1} & \cdots & a_0 & & \sum_{i=0}^{m} a_i x^{n+i-1} \\
 & a_m & a_{m-1} & \cdots & a_0 & \sum_{i=0}^{m} a_i x^{n+i-2} \\
 & & \ddots & \ddots & \ddots & \ddots \\
 & & & a_m & a_{m-1} & \cdots \quad \sum_{i=0}^{m} a_i x^{i} \\
b_n & b_{n-1} & \cdots & \cdots & b_0 & \sum_{i=0}^{n} b_i x^{m+i-1} \\
 & b_n & b_{n-1} & \cdots & \cdots & b_0 \quad \sum_{i=0}^{n} b_i x^{m+i-2} \\
 & & \ddots & \ddots & \ddots & \ddots \\
 & & & b_n & b_{n-1} & \cdots \quad \cdots \quad \sum_{i=0}^{n} b_i x^{i}
\end{array}\right]$$

(n rows; m rows)

$$=\left[\begin{array}{cccccc}
a_m & a_{m-1} & \cdots & a_0 & & x^{n-1}A(x) \\
 & a_m & a_{m-1} & \cdots & a_0 & x^{n-2}A(x) \\
 & & \ddots & \ddots & \ddots & \ddots \\
 & & & a_m & a_{m-1} & \cdots \quad A(x) \\
b_n & b_{n-1} & \cdots & \cdots & b_0 & x^{m-1}B(x) \\
 & b_n & b_{n-1} & \cdots & \cdots & b_0 \quad x^{m-2}B(x) \\
 & & \ddots & \ddots & \ddots & \ddots \\
 & & b_n & b_{n-1} & \cdots & \cdots \quad B(x)
\end{array}\right]$$

(n rows; m rows)

Note that since the last column of the matrix M' is simply

$$[x^{n-1}A(x),....,A(x),x^{m-1}B(x),...,B(x)]^T,$$

We can compute the $det(M')$ explicitly by expanding the determinant with respect to its last column. We then have the following:

$$\begin{aligned}
res(A,B) &= det(M') \\
&= x^{n-1}A(x)\cdot M'_{1,m+n}+...+A(x)\cdot M'_{n,m+n}+x^{m-1}B(x)\cdot M'_{n+1,m+n}+...+B(x)\cdot M'_{m+n,m+n} \\
&= A(x)(M'_{1,m+n}x^{n-1}+...+M'_{n,m+n})+B(x)(M'_{n+1,m+n}x^{m-1}+...+M'_{m+n,m+n}) \\
&= A(x)\cdot T(x)+B(x)\cdot U(x).
\end{aligned}$$

Note that the coefficients of $T(x)$ and $U(x)$ are cofactors of the last column of M', and hence of M, and are ring elements in S. Clearly,

$$\deg(T) \le n\text{-}1 < \deg(B) \text{ and } \deg(U) \le m\text{-}1 < \deg(A).$$

Lemma. [3]

Let S be a unique factorization domain with identity, and $A(x)$ and $B(x)$ be univariate polynomials of positive degrees with coefficients in S, then res(A, B) = 0 iff $A(x)$ and $B(x)$ have a common divisor of positive degree.

Proof.

Assume that $A(x)$ and $B(x)$ have a common divisor $C(x)$ of positive degree. Then,

$$A(x) = C(x) \cdot U(x), \qquad \deg(U) < \deg(A), \text{ and}$$

$$B(x) = C(x) \cdot T(x), \qquad \deg(T) < \deg(B).$$

Therefore, $A(x) \cdot T(x) + B(x) \cdot (-U(x)) = C(x) \cdot T(x) \cdot U(x) - C(x) \cdot T(x) \cdot U(x) = 0$.

Thus, there exists polynomials T, $U \in S[x]$, with $\deg(T) < \deg(B)$ and $\deg(U) < \deg(A)$, such that $A(x) \cdot T(x) + B(x) \cdot U(x) = 0$. And, by the previous lemma, res(A, B) = 0.

3.2.1.2 Homomorphisms and Resultants

Let S and S^* be commutative rings with identities, and $\phi : S \to S^*$ be a ring homomorphism of S into S^*. Note that ϕ induces a ring homomorphism of $S[x]$ into $S^*[x]$, also denoted by ϕ, as follows:

$$\phi : S[X] \to S^*[x]$$

$$: a_m x^m + a_{m-1} x^{m-1} + \ldots + a_0 \mapsto \phi(a_m) x^m + \phi(a_{m-1}) x^{m-1} + \ldots + \phi(a_0).$$

Lemma. [3]

Let $A(x) = \sum_{i=0}^{m} a_i x^i$ and $B(x) = \sum_{i=0}^{n} b_i x^i$ be two univariate polynomials over the ring S with $deg(A) = m > 0$ and $deg(B) = n > 0$.

If

$$\deg(\phi(A)) = m \text{ and } \deg(\phi(B)) = k, \qquad (0 \le k \le n),$$

Then

$$\phi(res(A, B)) = \phi(a_m)^{n-k} \cdot res(\phi(A), \phi(b)).$$

Proof.

Let

$$A^* = \phi(A) = \phi(a_m) x^m + \phi(a_{m-1}) x^{m-1} + \ldots + \phi(a_0), \quad \text{and}$$

$$B^* = \phi(B) = \phi(b_k) x^k + \phi(b_{k-1}) x^{k-1} + \ldots + \phi(b_0).$$

Then, M, the Sylvester matrix of $A(x)$ and $B(x)$, is

$$M = syl(\mathrm{A}, \mathrm{B}) = \begin{bmatrix} a_m & a_{m-1} & \cdots & a_0 & & & \\ & a_m & a_{m-1} & \cdots & a_0 & & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & & a_m & a_{m-1} & \cdots & a_0 \\ b_n & b_{n-1} & \cdots & \cdots & b_0 & & \\ & b_n & b_{n-1} & \cdots & \cdots & b_0 & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & b_n & b_{n-1} & \cdots & \cdots & b_0 \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\} n \text{ rows} \\ \left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\} m \text{ rows} \end{matrix}$$

and M^*, the Sylvester matrix of $A^*(x)$ and $B^*(x)$, is $M^* = syl(A^*, B^*)$

$$= \begin{bmatrix} \phi(a_m) & \phi(a_{m-1}) & \cdots & \phi(a_0) & & & \\ & \phi(a_m) & \phi(a_{m-1}) & \cdots & \phi(a_0) & & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & & \phi(a_m) & \phi(a_{m-1}) & \cdots & \phi(a_0) \\ \phi(b_k) & \phi(b_{k-1}) & \cdots & \cdots & \phi(b_0) & & \\ & \phi(b_k) & \phi(b_{k-1}) & \cdots & \cdots & \phi(b_0) & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & \phi(b_k) & \phi(b_{k-1}) & \cdots & \cdots & \phi(b_0) \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\} k \text{ rows} \\ \left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\} m \text{ rows} \end{matrix}$$

The matrix M^* is obtained from M by the following process:

1. First, the matrix, $\phi(M)$, is computed by replacing the entry a_j by $\phi(a_j)$ (for all $0 \le j \le m$), and by replacing the entry b_j by $\phi(b_j)$ (for all $0 \le j \le n$). By assumption
$$\phi(b_n) = ... = \phi(b_{k+1}) = 0$$
2. Next, from $\phi(M)$, the first (n - k) rows and (n - k) columns are deleted, yielding an $(m + k) \times (m + k)$ matrix equal to M^*.

Thus,

$$\phi(M) = \begin{bmatrix} \phi(a_m) & \phi(a_{m-1}) & \cdots & \phi(a_0) & & & \\ & \ddots & \ddots & \ddots & \ddots & & \\ & & & \phi(a_m) & \phi(a_{m-1}) & \cdots & \phi(a_0) \\ & & 0 & & & M^* & \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}a\\a\\a\end{matrix}}\right\} (n - k) \text{ rows} \\ \left.\vphantom{\begin{matrix}a\\a\\a\end{matrix}}\right\} (m + k) \text{ rows} \end{matrix}$$

Therefore,

$$\phi(res(A,B)) = \phi(\det(syl(A,B)))$$
$$= \det(\phi(M)) = \phi(a_m)^{n-k} \cdot \det(\phi(M^*))$$
$$= \phi(a_m)^{n-k} \cdot \phi(\det(syl(A^*,B^*)).$$
$$= \phi(a_m)^{n-k} \cdot \phi(res(A^*,B^*)).$$

Therefore,

$$\phi(res(A,B)) = \phi(a_m)^{n-k} \cdot \phi(res(A^*,B^*)).$$

- Let S be a commutative ring with an identity, and $(\alpha_1,...,\alpha_r) \in S^T$ be an r-tuple.

Define a ring homomorphism $\phi_{\alpha_1,...,\alpha_r}$, called the *evaluation homomorphism*, as follows:

$$\phi_{\alpha_1,...,\alpha_r} : S[x_1,...,x_r] \to S$$
$$: x_1 \mapsto \alpha_1,$$
$$\vdots$$
$$: x_r \mapsto \alpha_r.$$

Note that, if $F(x_1,...,x_r) \in S[x_1,...,x_r]$, then we shall write $F(\alpha_1,...,\alpha_r)$ for $\phi_{\alpha_1,...,\alpha_r}(F)$. The evaluation homomorphism will be used later in subsequent proofs.

3.2.1.3 ***Multivariable resultants***

Let S be a commutative ring with an identity, and

$$A(x_1,...,x_r) = \sum_{i=0}^{m} A_i(x_1,...,x_{r-1})x_r^i \in S[x_1,...,x_r], \quad \text{and}$$

$$B(x_1,...,x_r) = \sum_{i=0}^{n} B_i(x_1,...,x_{r-1})x_r^i \in S[x_1,...,x_r]$$

be two polynomials in $S[x_1,...,x_r]$ of respective positive degrees m and n in x_r. Then,

$$res_{x_r}(A,B) = \begin{bmatrix} A_m & A_{m-1} & \cdots & A_0 & & & \\ & A_m & A_{m-1} & \cdots & A_0 & & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & & A_m & A_{m-1} & \cdots & A_0 \\ B_n & B_{n-1} & \cdots & \cdots & B_0 & & \\ & B_n & B_{n-1} & \cdots & \cdots & B_0 & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & & B_n & B_{n-1} & \cdots & \cdots & B_0 \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\} n \text{ rows} \\ \left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\} m \text{ rows} \end{matrix}$$

is the resultant of two multivariate polynomials $A(x_1,...,x_r)$ and $B(x_1,...,x_r)$, with respect to x_r.

Lemma. [3]

Let L be an algebraically closed field, and let

$$res_{x_r}(A,B) = C(x_1,...,x_{r-1})$$

be the resultant of the multivariate polynomials,

$$A(x_1,...,x_r) = \sum_{i=0}^{m} A_i(x_1,...,x_{r-1})x_r^i \in L[x_1,...,x_r], \quad \text{and}$$

$$B(x_1,...,x_r) = \sum_{i=0}^{n} B_i(x_1,...,x_{r-1})x_r^i \in L[x_1,...,x_r].$$

with respect to x_r. Then

1. If $(\alpha_1,...,\alpha_r) \in L^r$ is a common zero of $A(x_1,...,x_r)$ and $B(x_1,...,x_r)$, then $C(\alpha_1,...,\alpha_{r-1}) = 0$.
2. Conversely, if $C(\alpha_1,...,\alpha_{r-1}) = 0$, then at least one of the following four conditions holds:
   (a) $A_m(\alpha_1,...,\alpha_{r-1}) = ... = A_0(\alpha_1,...,\alpha_{r-1}) = 0$, or
   (b) $B_n(\alpha_1,...,\alpha_{r-1}) = ... = B_0(\alpha_1,...,\alpha_{r-1}) = 0$, or
   (c) $A_m(\alpha_1,...,\alpha_{r-1}) = B_n(\alpha_1,...,\alpha_{r-1}) = 0$, or
   (d) For some $\alpha_r \in L, (\alpha_1,...,\alpha_r)$ is a common zero of both $A(x_1,...,x_r)$ and $B(x_1,...,x_r)$

Proof.

(1) Since there exist T and U in $L[x_1,...,x_{r-1}]$ such that $A \cdot T + B \cdot U = C$, we have

$$C(\alpha_1,...,\alpha_{r-1}) = A(\alpha_1,...,\alpha_r) \cdot T(\alpha_1,...,\alpha_r) + B(\alpha_1,...,\alpha_r) \cdot U(\alpha_1,...,\alpha_r)$$
$$= 0,$$

as $A(\alpha_1,...,\alpha_r) = B(\alpha_1,...,\alpha_r) = 0$, by assumption.

(2) Next, assume that $C(\alpha_1,...,\alpha_{r-1}) = 0$, but that conditions (a), (b), and (c) are not satisfied. Then there are two cases to consider:

1. $A_m(\alpha_1,...,\alpha_{r-1}) \neq 0$, and for some $k(0 \leq k \leq n), B_k(\alpha_1,...,\alpha_{r-1}) \neq 0$
   (k is assumed to be the largest such index).
2. $B_n(\alpha_1,...,\alpha_{r-1}) \neq 0$, and for some $k(0 \leq k \leq m), A_k(\alpha_1,...,\alpha_{r-1}) \neq 0$
   (k is assumed to be the largest such index).

Since *res*(B, A) = $(-1)^{mn}$ *res*(A, B) = ± C, cases (1) and (2) are symmetric, and without any loss of generality, we may only deal with the first case.

Let $\phi = \phi_{\alpha_1,\ldots,\alpha_{r-1}}$ be the evaluation homomorphism defined earlier.

Thus,

$$\begin{aligned} 0 &= \phi(c) \\ &= \phi(res(A,B)) \\ &= \phi(A_m)^{n-k} \cdot res(\phi(A),\phi(B)), \end{aligned}$$

and $res(\phi(A),\phi(B)) = 0,$ since $\phi(A_m) = A_m(\alpha_1,\ldots,\alpha_{r-1}) \neq 0,$

If $k = 0$, then $res(\phi(A),\phi(B)) = \phi(B_0)^m = B_0(\alpha_1,\ldots,\alpha_{r-1})^m \neq 0,$ (by assumption).

Hence $k > 0$ and $\phi(A)$ and $\phi(B)$ are of positive degree and have a common divisor of positive degree, say $D(x_r)$. Since L is algebraically closed, $D(x_r)$ has at least one zero, say α_r. Therefore,

$$A(\alpha_1,\ldots,\alpha_{r-1},x_r) = D(x_r)\cdot\tilde{A}(x_r), \text{ and}$$

$$B(\alpha_1,\ldots,\alpha_{r-1},x_r) = D(x_r)\cdot\tilde{B}(x_r),$$

and $(\alpha_1,\ldots,\alpha_{r-1},\alpha_r)$ is a common zero of A and B.

The following properties of linear systems of equations will be used for some of the subsequent proofs.

3.2.2 Homogeneous systems of linear equations [27]

- The quantities $Q_1, Q_2, \ldots, Q_n$ are said to be *linearly dependent* if there exists a set of constants $c_1, c_2, \ldots, c_n$, at least one of which is different from zero, such that the equation

$$c_1Q_1 + c_2Q_2 + \ldots + c_nQ_n = 0$$

holds identically.

- The quantities $Q_1, Q_2, \ldots, Q_n$ are said to be *linearly independent* if they are not linearly dependent, i.e., if the only linear equation of the form

$$c_1Q_1 + c_2Q_2 + \ldots + c_nQ_n = 0$$

which they satisfy identically has $c_1 = c_2 = \ldots = c_n = 0$

Given a system of the form

$$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1$$

$$a_{11}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2$$

.....

$$a_{m1}x_1 + a_{m2}x_2 + ... + a_{mn}x_n = b_m$$

where m, the number of equations, is not necessarily equal to n, the number of unknowns. If at least one of the m quantities b_i is different from zero, the system is said to be nonhomogeneous. If $b_i = 0$ for all values of i, the system is said to be homogeneous. If we define the matrices

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \qquad X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

The system can be written in the compact matrix form

$$AX = B \qquad (3.1)$$

In this form, the matrix A is known as the *coefficient matrix* of the system, and the matrix $[AB]$ obtained by adjoining the column matrix B to the coefficient matrix A as an $(n + 1)^{st}$ column is known as the *augmented matrix* of the system.

Before proceeding to the question of the existence and determination of solutions of (3.1), it shall first be proved several important theorems about such solutions on the assumption that they exist.

Theorem. [27]

If X_1 and X_2 are two solutions of the homogeneous matrix equation $AX = 0$, then for all values of the constants c_1 and c_2, the vector $c_1X_1 + c_2X_2$ is also a solution of the system of equations $AX = 0$.

Proof. By direct substitution we have

$$\begin{aligned} A(c_1X_1 + c_2X_2) &= A(c_1X_1) + A(c_2X_2) = 0 \\ &= c_1(AX_1) + c_2(AX_2) \\ &= c_1 \cdot 0 + c_2 \cdot 0 \\ &= 0 \end{aligned}$$

where the coefficients of c_1 and c_2 vanish because, by hypothesis, both X_1 and X_2 are solutions of $AX = 0$. Hence $c_1X_1 + c_2X_2$ also satisfies $AX = 0$, as asserted.

Theorem. [27]

If k is the maximum number of linearly independent solution vectors of the system $AX = 0$, and if $X_1, X_2,\ldots, X_k$ are k particular linearly independent solution vectors, then any solution vector of $AX = 0$ can be expressed in the form.

$$c_1X_1 + c_2X_2 + \ldots + c_kX_k$$

where the c_i's are scalar constants.

Proof. Let k be the maximum number of linearly independent solution vectors of the equation $AX = 0$; let $X_1, X_2,\ldots, X_k$ be a particular set of k linearly independent solution vectors; and let X_{k+1} be any solution vector. If X_{k+1} is one of the vectors in the set $\{ X_1, X_2,\ldots, X_k \}$, the assertion of the theorem is obviously true. If X_{k+1} is not a member of the set $\{ X_1, X_2,\ldots, X_k \}$, then $X_1, X_2,\ldots, X_k , X_{k+1}$ cannot be linearly independent since, by hypothesis, k is the maximum number of linearly independent solution vectors of $AX = 0$. Hence, the X's must satisfy a linear equation of the form

$$c_1X_1 + c_2X_2 + \ldots + c_kX_k + c_{k+1}X_{k+1} \qquad (3.2)$$

in which at least one c is different from zero. In fact, $c_{k+1} \neq 0,$ for otherwise equation (3.2) would reduce to

$$c_1X_1 + c_2X_2 + \ldots + c_kX_k = 0$$

with at least one of the c_i's different from zero, contrary to the hypothesis that $X_1, X_2,\ldots, X_k$ are linearly independent. But if $c_{k+1} \neq 0,$ it is clearly possible to solve equation (3.2) for X_{k+1} and express it in the form asserted by the theorem.

Because of the property guaranteed by the preceding theorem, a general linear combination of the maximum number of linearly independent solution vectors of $AX = 0$ is usually referred to as a *complete solution* of $AX = 0$.

Theorem. [27]

A system of m simultaneous linear equations in n unknowns, $AX = B$, has a solution if and only if the coefficient matrix A and the augmented matrix $[AB]$ have the same rank, r. When solutions exist, the maximum number of independent arbitrary constants in any general solution, i.e., the maximum number of linearly independent solution vectors of the related homogeneous system $AX = 0$, is $n - r$.

The proof is obtained by applying the well-known Gaussian elimination procedure [27].

3.3 IMPLICITIZATION ALGORITHMS

To this end, we can describe some given implicitization algorithms from the literature and some newly suggested ones.

3.3.1 Implicitization of rational curves

The brief overview of resultants, presented above, highlighted a major property they have. Namely, the resultant of two polynomials vanishes if and only if they have a common root. Resultants are thus used to determine whether or not two polynomials have a common root, without explicitly solving for the roots of the equations. Sometimes, actually solving for the common roots might be very expensive or even impossible when the number of common roots is infinite [13].

To see how resultants arise naturally in the implicitization of rational curves, consider a rational curve

$$X = x(t) / w(t), \qquad Y = y(t) / w(t),$$

where $x(t)$, $y(t)$ and $w(t)$ are polynomials. To obtain the implicit representation $f(X, Y) = 0$, introduce two auxiliary polynomials in t

$$X \cdot w(t) - x(t), \qquad Y \cdot w(t) - y(t).$$

By the above mentioned property, the resultant of these two polynomials vanishes if and only if they have a common root. This happens if and only if there is a common parameter t such that $X \cdot w(t) - x(t) = 0$ and $Y \cdot w(t) - y(t) = 0$ or $X = x(t) / w(t)$ and $Y = y(t) / w(t)$, i.e. the point (X, Y) is on the curve. So setting the resultant to zero yields the implicit equation of the parametric curve [10] [13].

Classical resultants are typically represented as determinants whose entries are polynomials in the coefficients of the original polynomials in the system. For two univariate polynomials, there are two well-known determinant representations: *Sylvester* resultants [3] [13] and *Bézout* resultants [10] [13]. The Sylvester resultant of two degree-n polynomials (already mentioned in the previous section) is a $2n \times 2n$ determinant constructed using Sylvester's dialytic method. This determinant has lots of zero entries, and the non-zero entries are simply the coefficients of the two original polynomials. The Bézout resultant generated from Cayley's expression, is a more compact $n \times n$ determinant whose entries are more complicated polynomials in the coefficients of the two original polynomials [13].

Base points

The major drawback of the resultant method for implicitization is that resultants vanish identically in the presence of base points. A rational curve or surface is said to have a base point if its numerators and denominators have a common factor, thus vanish simultaneously at some parameter value. Tom Sederberg [10] introduced the method of *moving algebraic curves and surfaces* to solve the implicitization problem for rational curves and surfaces with base points [13].

In this section, both methods (resultants and moving lines) are reviewed and investigated. In the next section, we have devised an algorithm for the method of moving lines, implemented it using Matlab and tested it for practical examples.

3.3.1.1 ***The resultant method*** [10]

The Sylvester resultant of two polynomials has already been viewed in the previous section. The construction of the Bézout resultant is a bit more complicated but yields a more compact $n \times n$ matrix as opposed to the $2n \times 2n$ Sylvester matrix.

Consider two degree-n polynomials $f(t)$ and $g(t)$

$$f(t) = a_n t^n + \ldots + a_1 t + a_0 , \qquad g(t) = b_n t^n + \ldots + b_1 t + b_0.$$

Let

$$f_k(t) = a_n t^k + \ldots + a_{n-k} , \qquad g_k(t) = b_n t^k + \ldots + b_{n-k} ,$$

$$p_{k+1}(t) = g_k(t) f(t) - f_k(t) g(t), \; k = 0, \ldots, n - 1.$$

Since

$$f(t) = f_k(t) t^{n-k} + a_{n-k-1} t^{n-k-1} + \ldots + a_1 t + a_0$$

$$g(t) = g_k(t) t^{n-k} + b_{n-k-1} t^{n-k-1} + \ldots + b_1 t + b_0$$

it follows that $p_1(t), \ldots, p_n(t)$ are polynomials of degree $n - 1$ in t. The Bézout resultant of $f(t)$ and $g(t)$ is the determinant of the coefficients of $p_1(t), \ldots, p_n(t)$. Explicit formulas for the entries of the Bézout resultant show that they are quadratic in the coefficients of $f(t)$ and $g(t)$. These are given in [13] by:

$B_{i,j} = \sum_{\alpha \le k \le \beta} (a_{i-k} b_{j+1+k} - b_{i-k} a_{j+1+k})$, where $\alpha = \max(0, i - j)$ and $\beta = \min(i, n - 1 - j)$.

In [10], it is proved that the Sylvester and Bézout resultant are equivalent up to sign, that is, det(Bézout) = ± det(Sylvester). Hence, these resultants can be used more efficiently in the problem of implicitization. The Bézout resultant is used in the method of moving lines [10].

3.3.1.2 ***The moving lines method*** [10]

Given a rational curve, $X = x(t) / w(t)$, $Y = y(t) / w(t)$, where

$$x(t) = a_n t^n + \ldots + a_1 t + a_0 \, , \; y(t) = b_n t^n + \ldots + b_1 t + b_0 \, , \; w(t) = d_n t^n + \ldots + d_1 t + d_0 \, .$$

By cross multiplication, two degree-n polynomials in t are formed.

$$X \cdot w(t) - x(t) = (Xd_n - a_n)t^n + \ldots + (Xd_1 - a_1)t + (Xd_0 - a_0)$$

$$Y \cdot w(t) - y(t) = (Yd_n - b_n)t^n + \ldots + (Yd_1 - b_1)t + (Yd_0 - b_0)$$

The Bézout resultant of $X \cdot w(t) - x(t)$ and $Y \cdot w(t) - y(t)$ can be obtained as:

$$f_k(t) = (Xd_n - a_n)t^k + \ldots + (Xd_{n-k} - a_{n-k})$$

$$g_k(t) = (Yd_n - b_n)t^k + \ldots + (Yd_{n-k} - b_{n-k})$$

$$p_{k+1}(t) = g_k(t)(Xw(t) - x(t)) - f_k(t)(Yw(t) - y(t)), \; k = 0, \ldots, n - 1.$$

It follows that $p_1(t), \ldots, p_n(t)$ are polynomials of degree-n in t as already shown in the analysis of Bézout resultants. Their coefficients are linear in x and y since these are sums of terms of the form

$$(Xd_i - a_i)(Yd_j - b_j) - (Xd_j - a_j)(Yd_i - b_i) = (b_i d_j - b_j d_i)X + (a_j d_i - a_i d_j)Y.$$

Thus, the p_k's can be written as

$$p_1(t) = L_{1,n-1}(X, Y)t^{n-1} + \ldots + L_{1,1}(X, Y)t + L_{1,0}(X, Y)$$

$$\ldots\ldots$$

$$p_n(t) = L_{n,n-1}(X, Y)t^{n-1} + \ldots + L_{n,1}(X, Y)t + L_{n,0}(X, Y)$$

where the functions $L_{ij}(X, Y)$ are linear in X, Y. The determinant of these coefficients $R(X, Y) = \det(L_{ij}(X, Y))$ is the Bézout resultant of the polynomials $Xw(t) - x(t)$ and $Yw(t) - y(t)$. Thus, the implicit equation is $R(X, Y) = 0$.

Moving lines

The polynomials $p_k(t)$ whose coefficients appear in the Bézout resultant can be written in two forms:

1. as above: $L_{n-1}(X, Y)t^{n-1} + \ldots + L_1(X, Y)t + L_0(X, Y)$,
2. or by collecting the coefficients of X and Y: $A(t)X + B(t)Y + C(t)$,

where $A(t)$, $B(t)$, $C(t)$ are polynomials of degree $n - 1$ in t.

For each value of t, the second form can be seen as an implicit form of a line in the xy-plane. Sederberg [10] calls such a line a *moving line*. A moving line $A(t)X + B(t)Y + C(t)$ is said to follow a rational curve $X = x(t) / w(t)$, $Y = y(t) / w(t)$ if it vanishes on the curve, i.e., if the line $A(t)x(t) / w(t) + B(t)y(t) / w(t) + C(t) = 0$, or equivalently, if

$$A(t)x(t) + B(t)y(t) + C(t)w(t) = 0, \text{ for all values of } t.$$

In what follows, the method of moving lines in implicitization is explained.

Methods of moving lines

Case I: No base points.

Given a rational curve $X = x(t) / w(t)$, $Y = y(t) / w(t)$ of degree n, it is required first to seek all moving lines

$$L_{n-1}(X, Y)t^{n-1} + \ldots + L_1(X, Y)t + L_0(X, Y) = 0 \qquad (3.3)$$

of degree n - 1 that follow the curve. Since each coefficient $L_j(X, Y)$ is linear in X, Y, equation (3.3) can be rewritten as

$$(A_{n-1}X + B_{n-1}Y + C_{n-1})t^{n-1} + \ldots + (A_1X + B_1Y + C_1)t + (A_0X + B_0Y + C_0) = 0 \qquad (3.4)$$

In equation (3.4), there are $3n$ unknowns-A_k, B_k, C_k, $k = 0, \ldots, n$-1. We can generate $2n$ homogeneous linear equations in these $3n$ unknowns by substituting for X and Y the rational functions $x(t) / w(t)$ and $y(t) / w(t)$ and multiplying through by $w(t)$. This yields the equation

$$(A_{n-1}\,x(t) + B_{n-1}\,y(t) + C_{n-1}w(t))t^{n-1} + \ldots + (A_0\,x(t) + B_0\,y(t) + C_0w(t)) = 0 \qquad (3.5)$$

where the left hand side is a polynomial in t of degree $2n$-1. For equation (3.4) to represent a moving line that follows the rational curve, this polynomial must be identically zero. The vanishing of this polynomial leads to $2n$ homogeneous linear equations in these $3n$ unknowns, which in matrix form can be written as

$$[x \;\; y \;\; w \ldots t^{n-1}x \;\; t^{n-1}y \;\; t^{n-1}w] \cdot [A_0 \;\; B_0 \;\; C_0 \ldots A_{n-1} \;\; B_{n-1} \;\; C_{n-1}]^{\mathrm{T}} = 0,$$

where the rows of the coefficient matrix $[x \;\; y \;\; w \ldots t^{n-1}x \;\; t^{n-1}y \;\; t^{n-1}w]$ are indexed by the powers of t and the columns are the coefficients of the polynomials t^kx, t^ky, t^kw, $k = 0, \ldots, n$-1

A homogeneous linear system of $2n$ equations in $3n$ unknowns has at least n linearly independent solutions. Let

$$\begin{aligned} p_1(t) &= L_{1,n-1}(X, Y)t^{n-1} + \ldots + L_{1,1}(X, Y)t + L_{1,0}(X, Y) = 0 \\ &\ldots\ldots \qquad (3.6) \\ p_n(t) &= L_{n,n-1}(X, Y)t^{n-1} + \ldots + L_{n,1}(X, Y)t + L_{n,0}(X, Y) = 0 \end{aligned}$$

be n linearly independent solutions, and form the matrix $R(X, Y) = (L_{ij}(X, Y))$. Since $L_{ij}(X, Y)$ is linear in X and Y, $det\{R(X, Y)\}$is a polynomial of degree n in X and Y. Also, $det\{R(X, Y)\}$= 0 when X, Y is on the rational curve because by equation (3.6) when X, Y is on the curve, the columns of $R(X, Y)$are dependent. Thus, $det\ \{R(X, Y)\}$= 0 is a good candidate for the implicit equation of the rational curve, since $det\ \{R(X, Y)\}$has the correct degree and vanishes on the curve. It is next formally proved that as long as we choose the moving lines $p_1(t), \ldots, p_n(t)$ to be linearly independent, $det\{R(X, Y)\}$= 0 is indeed the implicit equation of the rational curve, provided that there are no base points.

Proof

Both the Sylvester and the Bézout resultants will be used in this proof. First recall that the rows of the Bézout resultant for the polynomials $xw(t) - x(t)$, $yw(t) - y(t)$ are moving lines of degree $n - 1$ that follow the curve. Since when there are no base points the determinant of the Bézout matrix is the implicit equation of the rational curve, it is known that these rows must be linearly independent; otherwise this determinant would be identically zero. To prove that the method of moving lines always works, it is required to show that when there are no base points, there are never more than n linearly independent moving lines of degree n - 1 that follow a degree n rational curve. Consider then the system of $2n$ homogeneous linear equations in $3n$ unknowns which is to be solved to find the moving lines that follow the curve. In matrix form these equations are:

$$[x \;\; y \;\; w \ldots t^{n-1}x \;\; t^{n-1}y \;\; t^{n-1}w] \cdot [A_0 \;\; B_0 \;\; C_0 \ldots A_{n-1} \;\; B_{n-1} \;\; C_{n-1}]^{\mathrm{T}} = 0.$$

Let $C = [x \;\; y \;\; w \ldots t^{n-1}x \;\; t^{n-1}y \;\; t^{n-1}w]$ be the coefficient matrix. To prove that there are never more that n linearly independent moving lines that follow a degree n rational curve, we must show that C has rank $2n$. To do so, let $x^*(t) = x(t) + Xt^{2n}$, $y^*(t) = y(t) + Yt^{2n}$, $w^*(t) = w(t) + t^{2n}$, and let S denote the $3n$ x $3n$ matrix $S = [x^* \;\; y^* \;\; tx^* \;\; ty^* \;\; \ldots \; t^{n-1}x^* \;\; t^{n-1}y^* \;\; w^* \;\; tw^* \ldots \; t^{n-1}w^*]$. Note that C is a submatrix of S with its columns rearranged. But $det(S)$ is the Sylvester resultant of $xw(t) - x(t)$ and $yw(t) - y(t)$. To prove this assertion, subtract x-column ($t^k x^*$) from column ($t^k x^*$) and y-column ($t^k x^*$) from column ($t^k x^*$), $k = 0, \ldots, n\text{-}1$. This procedure leaves the determinant unchanged, kills off the x's and y's in the rows below the row that is indexed by t^{2n-1}, and produces the matrix (the * represents arbitrary elements):

$$\begin{pmatrix} (2n \times 2n)\ \text{Sylvester matrix of} & * \\ xw(t) - x(t)\ \text{and}\ yw(t) - y(t) & \\ & \\ (n \times 2n)\ 0 - \text{matrix} & (n \times n)\ \text{Identity matrix} \end{pmatrix}$$

whose determinant is indeed just the Sylvester resultant of $xw(t) - x(t)$ and $yw(t) - y(t)$. Since there are no base points, $xw(t) - x(t)$ and $yw(t) - y(t)$ do not have common roots for arbitrary values of x and y. therefore the Sylvester resultant of $xw(t) - x(t)$ and $yw(t) - y(t)$ is not identically zero. Hence $det(S) \neq 0$, so the rows of S are linearly independent. Hence the rows of C must also be linearly independent, so C has rank $2n$. Since C has full rank, the number of linearly independent solutions of equation (3.4) (i.e. the number of linearly independent moving lines that follow the curve) is exactly n. Thus the rows of the Bézout resultant span the solution space. Hence there is a constant nonsingular matrix M for which we have $R(X, Y)$

= $M \times$ Bézout Matrix. Since M is nonsingular, $det(R(X, Y)) = 0$ if det(Bézout Matrix) = 0. Hence $det(R(X, Y)) = 0$ is indeed the implicit equation of the rational curve.

Case II: Base points present.

Suppose then that $X = x(t) / w(t)$, $Y = y(t) / w(t)$ is a degree n rational curve with r base points. Then $x(t)$, $y(t)$, $w(t)$ have a common factor $c(t)$ of degree r; that is,

$$x(t) = c(t)x^*(t) \quad y(t) = c(t)y^*(t) \quad w(t) = c(t)w^*(t)$$

To apply the method of moving lines, we need to know how many moving lines of degree n - 1 follow this rational curve. A moving line of degree $n - 1$ follows a degree n rational curve when the left-hand side of equation (3.5) is identically zero. Canceling out the common factor $c(t)$, we obtain

$$(A_{n-1}\, x^*(t) + B_{n-1}\, y^*(t) + C_{n-1}w^*(t))t^{n-1} + \ldots + (A_0\, x^*(t) + B_0\, y^*(t) + C_0 w^*(t)) = 0$$

where the left-hand side is a polynomial of degree $2n - r - 1$. The vanishing of this polynomial leads to $2n - r$ homogeneous linear equations in the $3n$ unknowns, A_k, B_k, C_k, for $k = 0, \ldots, n\text{-}1$. Now a linear system of $2n - r$ homogeneous equations in $3n$ unknowns has at least $n + r$ linearly independent solutions. Let

$$p_1(t) = L_{1,\,n-1}(X, Y)t^{n-1} + \ldots + L_{1,1}(X, Y)t + L_{1,0}(X, Y) = 0$$

$$\ldots\ldots$$

$$p_{n+r}(t) = L_{n+r,\,n-1}(X, Y)t^{n-1} + \ldots + L_{n+r,\,1}(X, Y)t + L_{n+r,\,0}(X, Y) = 0$$

be $n + r$ linearly independent solutions, and form the $(n + r) \times n$ matrix $R(X, Y) = (L_{ij}(X, Y))$. It is required to show how to use the matrix $R(X, Y)$ to recover the implicit equation of the rational curve. The key idea is to apply Gaussian elimination.

Since $p_k(t)$ follows the rational curve, the line $p_k(0) = L_{k,0}(X, Y)$ must pass through the point $(x(0) / w(0), y(0) / w(0))$. Therefore the lines $L_{k,0}(X, Y)$, $k = 1, \ldots, n+r$, are concurrent, so any three of these lines are linearly dependent. Thus by Gaussian elimination we can zero out all but the first two elements in the last column of $R(X, Y)$. Since the rows of this new matrix are linear combinations of moving lines that follow the curve, these rows are again moving lines that follow the curve. If we remove the first two rows and last column of this matrix, then the remaining moving lines have the form

$$\Lambda_k(t) = \Lambda_{k,n-1}(X, Y)t^{n-1} + \ldots + \Lambda_{k,1}(X, Y)t.$$

Factoring out t, we obtain an $(n + r - 2) \times (n - 1)$ matrix of $n + r - 2$ moving lines of degree $n - 2$ that follow the curve. Repeating this Gaussian elimination r times, we arrive at $n - r$ moving lines of degree $n - r - 1$ that follow that curve.

Let $S(X, Y)$ be the $(n - r) \times (n - r)$ matrix generated from $R(X, Y)$ by Gaussian elimination. Then $det(S(X, Y)) = 0$ is the implicit equation of the rational curve.

Proof

By construction, the rows of $R(x, y)$ are linearly independent. Since $S(x, y)$ is generated from $R(x, y)$ by Gaussian elimination, the rows of $S(x, y)$ must also be linearly independent. Therefore, $\det(S(x, y)) = 0$ must be the implicit equation of the rational curve given by $X = x^*(t) / w^*(t)$, $Y = y^*(t) / w^*(t)$ (with no base points), and hence too the implicit equation of the original rational curve (with base points).

It seems natural to give a brief note on complexity analysis of the above mentioned implicitization algorithms.

3.3.1.3 Complexity analysis

The analysis of an algorithm is a measure that relates the size of the input to the time and space this algorithm requires to execute (and deliver the output) on a particular machine. To obtain standard measures, several hypothetical machines, on which programs are run, have been used, such as the Turing machine or RAM machine [28].

In the analysis of implicitization algorithms, however, there is fixed cost of computing the determinant of a certain matrix, the resultant. So, what interests us most is the time required to compute the entries of the matrix and, sure before that, the size of the matrix.

The following is an analysis of the methods explained earlier.

1. Sylvester resultant method

The Sylvester matrix takes constant time to build up, since its entries are simply coefficients of polynomials used without any further processing. However, for n degree parametric input, the matrix is $2n \times 2n$.

2. Bézout resultant method [13]

A closed formula for the ij^{th} entry of the Bézout resultant has already been given at the beginning of this section by

$$B_{i,j} = \sum_{\alpha \le k \le \beta} (a_{i-k} b_{j+1+k} - b_{i-k} a_{j+1+k}), \text{ where } \alpha = \max(0, i - j) \text{ and } \beta = \min(i, n - 1 - j).$$

As a result of symmetry, only the entries $B_{i,j}$ for which $i \le j$ need to be computed. We have the formulas:

$$B_{i,j} = \sum_{0 \le k \le i} (a_{i-k} b_{j+1+k} - b_{i-k} a_{j+1+k}), \quad i + j \le n - 1,$$

$$B_{i,j} = \sum_{0 \le k \le n-1-j} (a_{i-k} b_{j+1+k} - b_{i-k} a_{j+1+k}), \quad i + j > n - 1.$$

Each summand requires two multiplications and on addition. When n is odd, the total number of multiplications to compute all the $B_{i,j}$'s, $i \le j$, is:

$2\cdot\sum_{0\le i\le (n-1)/2}(n-2i)\cdot(i+1) + 2\cdot\sum_{(n+1)/2\le j\le n-1}(n-j)\cdot(2j-n+1) = (2n^3+9n^2+10n+3)/12$,

and the total number of additions is:

$\sum_{0\le i\le (n-1)/2}(n-2i)\cdot(2i+1) + \sum_{(n+1)/2\le j\le n-1}(2n-2j-1)\cdot(2j-n+1) = (2n^3+3n^2+4n+3)/12$.

Similar results hold when n is even. Therefore the method requires $O(n^3)$ multiplications and additions. In [13], M. Zhang has developed a faster technique that requires only $O(n^2)$ additions and multiplications.

3. Moving line method

As shown in the description of the method, the entries of the determinant are computed from the solution of a $2n \times 3n$ system of equations. This is known to be $O(n^3)$ operations. The moving line method power does not lie in a reduced complexity. It succeeds to solve the implicitization problem in the presence of base points. It can be generalized also to the *moving conics* method and can be also applied to the implicitization of parametric surfaces in the presence of base points [10] [13].

In the following, other implicitization methods are described.

3.3.2 Implicitization of generalized trigonometric polynomials curves

Consider a plane curve given parametrically by a generalized trigonometric polynomial, that is $x = \sum_{1\le k\le n} a_k \cos k\theta$, $y = \sum_{1\le k\le n} a_k \sin k\theta$. It is required to obtain an implicitization of the curve, i.e. an equation in x and y which captures all the points on the curve, and, if any, only finitely many more points.

A direct approach that makes use of algorithms already developed for the implicitization of rational curves is as follows:

1) Rewrite $\cos k\theta$ and $\sin k\theta$ as polynomials in $\cos\theta$ and $\sin\theta$. Using De Moivre's theorem of complex numbers, such expressions are found to be:

$$\cos k\theta = \cos^k\theta - \frac{k(k-1)}{2!}\cos^{k-2}\theta\sin^2\theta + \frac{k(k-1)(k-2)(k-3)}{4!}\cos^{k-4}\theta\sin^4\theta - \cdots$$

$$\sin k\theta = k\cos^{k-1}\theta\sin\theta - \frac{k(k-1)(k-2)}{3!}\cos^{k-3}\theta\sin^3\theta + \cdots$$

2) Parameterize $\cos\theta$ and $\sin\theta$ by the usual rational parameterization of a circle:

$$x = \cos\theta = \frac{1-t^2}{1+t^2} \quad , \quad y = \sin\theta = \frac{2t}{1+t^2}$$

3) Implicitize the rational functions obtained from the preceding step by the implicitization algorithm for rational functions.

A much simpler and computationally cheaper method, which takes advantage of the special structure of the problem, is devised in [11].

In the subsequent chapter, we devise an algorithm for this method, implement it on Matlab and apply it for some practical examples.

Consider a curve given parametrically by

$$x = \sum_{k=1}^{n} a_k \cos k\theta \quad , \quad y = \sum_{k=1}^{n} a_k \sin k\theta \qquad (3.7)$$

where $a_1,...,a_n,\theta \in R$ and $a_n \neq 0$.

Thus, $x + iy = \sum_{1\le k\le n} a_k e^{ik\theta}$. Renaming $e^{i\theta}$ with z, a polynomial F is obtained. Then taking the complex conjugate of F and multiplying by z^n, another polynomial G results:

$$F = -(x+iy) + \sum_{k=1}^{n} a_k z^k \quad , \quad G = -(x-iy)z^n + \sum_{k=1}^{n} a_k z^{n-k}$$

where $F, G \in C[x, y, z]$, C being the field of complex numbers.

Let $H \in C[x, y]$ be defined by: $H = \mathrm{res}_z(F, G)$, that is, H is the determinant of the matrix:

$$\begin{bmatrix}
a_n & a_{n-1} & \cdots & a_1 & a_0 & & & \\
 & a_n & a_{n-1} & \cdots & a_1 & a_0 & & \\
 & & \cdots & \cdots & \cdots & \cdots & \cdots & \\
 & & & a_n & a_{n-1} & \cdots & a_1 & a_0 \\
\bar{a}_0 & a_1 & \cdots & a_{n-1} & a_n & & & \\
 & \bar{a}_0 & a_1 & \cdots & a_{n-1} & a_n & & \\
 & & \cdots & \cdots & \cdots & \cdots & \cdots & \\
 & & & \bar{a}_0 & a_1 & \cdots & a_{n-1} & a_n
\end{bmatrix},$$

where $a_0 = -(x + iy)$ and $\bar{a}_0 = -(x - iy)$, where the upper bar stands for complex conjugation.

The polynomial H gives an implicitization of the curve defined by the equations (3.7). More in detail:

- $H = 0$ captures all the points on the curve. (Containment)
- $H = 0$ captures, if any, only finitely many more points. (Finite exceptions)
- H is a real polynomial, that is $H \in R[x, y]$. (Real)

Proof

Let

$$S = \{(x, y) \in R^2 \mid (\exists\, \theta \in R) \text{ s.t. } (x = \sum_{k=1}^{n} a_k \cos k\theta) \wedge (y = \sum_{k=1}^{n} a_k \sin k\theta)\} \text{ (parametric)},$$

$$T = \{\ (x, y) \in R^2 \mid H(x, y) = 0\ \}. \text{ (implicit)}.$$

1) Containment ($S \subseteq T$): H captures all the points on the curve.

Let $(p, q) \in S$. Then it is required to show that $(p, q) \in T$.

First note that from the definition of S, there must exist $t \in R$ such that

$$p = \sum_{k=1}^{n} a_k \cos kt$$

$$q = \sum_{k=1}^{n} a_k \sin kt.$$

By using the elementary fact: $\cos t + i \sin t = e^{it}$, we obtain

$$p + iq = \sum_{k=1}^{n} a_k e^{ikt}$$

$$p - iq = \sum_{k=1}^{n} a_k e^{-ikt}.$$

By moving the left hand sides to the right and by multiplying e^{int} to the second equation, we obtain

$$0 = -(p + iq) + \sum_{k=1}^{n} a_k e^{ikt}$$

$$0 = (p - iq)e^{\text{int}} + \sum_{k=1}^{n} a_k e^{-i(n-k)t}.$$

Note that the right hand sides are the polynomials F and G evaluated at p, q, e^{it}. Let r denote e^{it}. Then, we have

$$F(p, q, r) = 0$$

$$G(p, q, r) = 0$$

By the properties of multivariable resultants, we immediately have

$$H(p, q) = 0.$$

Thus, $(p, q) \in T$.

2) Finite exceptions (T - S is a finite set): H captures, if any, only finitely many more points.

Now one might wonder whether $S = T$. Unfortunately this is not true in general, as illustrated by the following counter example:

$$x = 5/2\cos\theta - \cos 2\theta \quad , \quad y = 5/2\sin\theta - \sin 2\theta.$$

The associated polynomial H is

$$-\frac{21}{4} - \frac{33}{4}x^2 - \frac{33}{4}y^2 + x^4 + 2x^2y^2 + y^4 + \frac{25}{2}x.$$

A simple calculation shows that the point (1, 0) is on the curve T given by $H = 0$, but it is not on the curve S given by the parametric equations. In fact, in this particular case, we have

$$T - S = \{(1, 0)\}.$$

Observe that the set {(1,0)} is a finite set. Thus one naturally wonders whether $T - S$ is always a finite set. It is, indeed and the following gives a technical method to check whether H introduces new points or not.

Let $p(r) = \sum_{k=1}^{n} a_k((\bar{r}r)^{k-1} + \cdots + \bar{r}r + 1)r^{n-k}$. Then, if the equation $p(r) = 0$ hasn't any roots of nonunit magnitude (i.e. hasn't any r such that $\bar{r}r \neq 1$), the implicitization method doesn't introduce new points. The proof is in [11].

Again for the previous example, $n = 2$ and $p(r) = a_2((\bar{r}r) + 1) + a_1 \cdot (1)r$. Letting r being real, we obtain the equation $-r^2 - 1 + 5/2r = 0$, which has nonunit magnitude roots, thus, introducing new points.

3) Real ($H \in R[x, y]$): H has only real coefficients in spite of the fact that its definition involves imaginary unit number i.

Let F and G be two polynomials in $C[x_1, \ldots, x_v]$ such that

$$F = \sum_{k=0}^{n} F_k x_v^k \quad , \quad G = \sum_{k=0}^{n} \overline{F}_k x_v^{n-k}$$

where $F_k \in C[x_1, \ldots, x_v]$ and $\overline{F}_k$ is the complex conjugate of F_k (in the sense that the coefficients of F_k are replaced by their complex conjugates). Let $H = res_{x_v}(F, G)$. Note that it is dealt here only with a particular case where

$$\nu = 3, x_1 = x, x_2 = y, x_3 = z$$

and

$$F_k = \begin{cases} a_k \text{ for } k \geq 1 \\ -(x + iy) \text{ for } k = 0 \end{cases}$$

But the proved result holds in general.

From the definition of resultant, $H = res_{x_v}(F, G)$, we have:

$$H = \det\begin{bmatrix} F_n & \cdots & \cdots & F_0 & & \\ & \cdots & \cdots & \cdots & \cdots & \\ & & F_n & \cdots & \cdots & F_0 \\ \overline{F}_0 & \cdots & \cdots & \overline{F}_n & & \\ & \cdots & \cdots & \cdots & \cdots & \\ & & \overline{F}_0 & \cdots & \cdots & \overline{F}_n \end{bmatrix}$$

By flipping the matrix left-right (which can be done by n swapping of columns), we obtain

$$H = \det\begin{bmatrix} & & F_0 & \cdots & \cdots & F_n \\ & \cdots & \cdots & \cdots & \cdots & \\ F_0 & \cdots & \cdots & F_n & & \\ & & \overline{F}_n & \cdots & \cdots & \overline{F}_0 \\ & \cdots & \cdots & \cdots & \cdots & \\ \overline{F}_n & \cdots & \cdots & \overline{F}_0 & & \end{bmatrix} (-1)^n$$

By flipping the matrix top-down (which can be done by n swapping of rows), we obtain

$$H = \det\begin{bmatrix} \overline{F}_n & \cdots & \cdots & \overline{F}_0 & & \\ & \cdots & \cdots & \cdots & \cdots & \\ & & \overline{F}_n & \cdots & \cdots & \overline{F}_0 \\ F_0 & \cdots & \cdots & F_n & & \\ & \cdots & \cdots & \cdots & \cdots & \\ & & F_0 & \cdots & \cdots & F_n \end{bmatrix} (-1)^{2n}$$

Since the conjugate operation commutes with determinant operation, we have

$$H = \det\begin{bmatrix} F_n & \cdots & \cdots & F_0 & & \\ & \cdots & \cdots & \cdots & \cdots & \\ & & F_n & \cdots & \cdots & F_0 \\ \overline{F}_0 & \cdots & \cdots & \overline{F}_n & & \\ & \cdots & \cdots & \cdots & \cdots & \\ & & \overline{F}_0 & \cdots & \cdots & \overline{F}_n \end{bmatrix}$$

Thus we finally have $H = \overline{H}$, hence H is real.

Remark

There is no overhead in computing the entries of the $2n \times 2n$ Sylvester resultant used in the algorithm. In [11], this method is compared to a classical solution method using a Gröbner basis approach. The comparison is performed using the running time it takes for solving certain problems on the same machine, operating system and using the same software package. The ratio is 5 min to 0.3 sec for the new technique of course.

Newly suggested implicitization methods, namely, fixed near implicitization and multiple near implicitization are described in what follows.

3.3.3 Near implicitization

For computational problems, people usually build models using polynomials systems. For example, many computer aided design (CAD) and ocmputer aided manufacturing (CAM) systems use Bézier-Bernstein splines to model mechanical components and assemblies [13]. Even if a model is not a polynomial system, often it may be reduced to a polynomial system or approximated to a polynomial system. For instance, when a system involves transcendental functions, we may not be able to solve the system directly. Instead we try to replace these functions or approximate them with polynomials, say with finite Taylor expansions. The reason for these reductions and approximations is that a lot is known about working with polynomial systems.

In this section, we devised what we have called the *near implicitization method*. Parametric functions for which a direct method is not available are approximated using Taylor's expansion. The resulting polynomial is then implicitized using the algorithm developed for rational functions.

We had to compromise between the accuracy of the approximation and the execution time of the algorithm. This has led us to develop two variants of the method: fixed and multiple near implicitizations.

3.3.3.1. Fixed near implicitization

In this method, Taylor expansion is evaluated around a fixed point. Accuracy is achieved by expanding more terms. This method is quite simple to implement and to apply to different curves without modifying the algorithm. It has the disadvantage of generating large determinants.

For n degree rational parametric equations, an $n \times n$ determinant is required to obtain the implicit equation. This is a quadratic increase in the determinant size paid for the increase in the number of terms of the Taylor's expansion. In addition, it leads to implicit equations with a large number of terms, less efficient in their manipulation.

3.3.3.2. Multiple near implicitization

We have to overcome the problem of large determinants, present in the previous method, by using what we called multiple near implicitization. Here, Taylor expansion uses fewer terms (thus lower degrees), but is evaluated at several points along the curve to be implicitized. The end result is a collection of implicit equations, each valid for an interval of the curve. The price paid for increased accuracy is an added number of determinants to be evaluated. However, this is much less severe than the overhead with the previous method, since all determinants are of size $n \times n$, yielding smaller implicit equations.

A bound checking routine is then added against which the (x, y) values are checked to direct them to the correct implicit equation to be used.

Sample runs at the end of the next section illustrate the comparison between the two methods for different curves.

Chapter 4: IMPLEMENTATION

In this chapter, we devise algorithms for all methods of implicitization mentioned in the previous chapter, namely, algorithms for the implicitization of parametric rational curves (in the absence and in the presence of base points), and the implicitization of curves parameterized by generalized trigonometric polynomials. We have also devised algorithms for our newly introduced method of near implicitization.

In addition, we have implemented all the previous algorithms using the software package Matlab. Starting with version 5.1, Matlab has added a toolbox for symbolic computation. We have benefited from this feature in our implementations, together with the original capability of the software package to deal with matrices and determinants efficiently. We have devised and implemented three *M-files* (the name Matlab uses for its functions), which when added together may form a Matlab toolbox for implicitization.

4.1 IMPLICITIZATION OF RATIONAL CURVES

4.1.1 Algorithm

The following algorithm computes the implicit equation of a curve defined parametrically by rational functions. It uses the method of moving lines presented in the previous chapter. It also combines both the cases of absence and presence of base points.

Algorithm 4.1: Rational implicitization.

Input: $x(t), y(t), w(t) \in R[t]$, n (degree of input)

Output: $S(x, y) \in R[x, y]$ such that $S(x, y) = 0$ is the implicit equation of the parametric equations: $x = x(t) / w(t)$, $y = y(t) / w(t)$.

x := coefficients of $x(t)$; y := coefficients of $y(t)$; w := coefficients of $w(t)$;

construct the $2n \times 3n$ coefficient matrix A by staggering the vectors v, y, w column-wise;

for i := 1, 2, …, k **do**

$v(i)$:= vectors spanning the solution space of $Av = 0$;

num_of_base_points := $n - k$;

remove base points and create the $k \times k$ resultant matrix S with coefficients from $v(i)$;

return det(S);

4.1.2 Implementation

This note is intended to reveal some of the mathematical and programming features of the next function.

1. The function accepts input polynomials in symbolic format (e.g. 3*t^2 + 1) rather than the usual Matlab convention of vector of coefficients (e.g. [3 0 1]). This is a more user-friendly format and is less error prone.
2. Variable number of input arguments is supported. A missing denominator value defaults to 1.
3. Numeric to symbolic conversion is performed within the function thus enabling the user to enter constants (numeric) for any of the input arguments.
4. The same function has provision for dealing with the case of existence of base points. Their presence is automatically detected and the implicit equation for the reduced parametric equations is returned.
5. The output is in symbolic format.

(M-File 1)

```
function [R] = implicitrat(xt, yt, wt)
%IMPLICITRAT      Implicit equation.
%    Implicitrat(xt, yt, wt) returns the implicit equation
%    of a parametrically defined one.
%    R(x,y)=0 is the implicit symbolic equation.
%    xt, yt and wt are the symbolic polynomials.
%    If wt=1, it need not be input. Enter only xt and yt
%    [ x=x(t)/w(t) , y=y(t)/w(t) ] is the rational parametric form.

t = sym('t');           % define symbolic variable
% Checking number of input parameters
if (nargin < 2) error('Not enough input arguments'); end
if (nargin == 2) wt = 0*t + 1; end

% Checking type of input parameters
if isnumeric(xt) xt = 0*t + xt; end
if isnumeric(yt) yt = 0*t + yt; end
if isnumeric(wt) wt = 0*t + wt; end

% Convert symbolic polynomials to coefficients vector
xn = sym2poly(xt);
```

```
yn = sym2poly(yt);
w  = sym2poly(wt);

n = max([length(xn)  length(yn)  length(w)])-1; % degree of parametric form
% right align vectors by padding with leading zeros
for i = 1:(n+1-length(xn)) xn = [0 xn]; end;
for i = 1:(n+1-length(yn)) yn = [0 yn]; end;
for i = 1:(n+1-length(w )) w  = [0  w]; end;

xn = flipLR(xn)'; % flip left/right
yn = flipLR(yn)'; % (least significant first)
w  = flipLR(w)';  % then transpose

A = [xn yn w];    % coefficients matrix whose columns are tx, ty and tw
z = [];           % vector that will be used to add zero rows to matrix A

for i = 2:n
  xn = [0;xn]; yn=[0;yn]; w=[0;w];  % shift down one row
  z = [z zeros(1,3)];
  A = [A;z];                        % append a zero row
  A = [A xn yn w];
end;

format rat;             % Rational format
S = null(A,'r');        % Null space of Ax=0 w.r.t. rational basis

x = sym('x'); y = sym('y');   % define symbolic variables

% Fill the resultant with moving lines
bp = length(S(1,:)) - length(S(:,1))/3;   % Number of base points if any
for i = 1:n-bp
  for j = 1:n-bp
      dr(i,j) = S(3*(j-1)+1 , i)*x + S(3*(j-1)+2 , i)*y + S(3*(j-1)+3 , i);
  end
end;

R=det(dr);

format;                       % Back to default format
% end of implicit function
```

4.1.3 Sample runs

Sample run 1-a: (Circle)

```
» t=sym('t');
» x=1-t^2;
» y=2*t;
» w=1+t^2;
» pretty(x)
                                      2
                                   1 - t
» pretty(y)
                                      2 t
» pretty(w)
                                      2
                                   1 + t
» imp=implicit(x,y,w);
» pretty(imp)
                                    2        2
                                   x  - 1 + y
```

Sample run 1-b: (Circle, with base points)

Here, two base points are added by multiplying numerators and denominators by $(t + 2)(t + 5)$

```
» xt=-t^4-7*t^3-9*t^2+7*t+10;
» yt=2*t^3+14*t^2+20*t;
» wt=t^4+7*t^3+11*t^2+7*t+10;

» r=implicitrat(xt,yt,wt);
» pretty(r)
                                    2        2
                                   x  - 1 + y
```

Sample run 1-c: (Circle, with base points)

Base points obtained by multiplying numerators and denominators by $(1 + t)(1 - t)$

```
» xt=1-2*t^2+t^4;
» yt=2*t-2*t^3;
» wt=1-t^4;

» pretty(implicitrat(xt,yt,wt))
                                    2        2
                                   x  - 1 + y
```

Sample run 2-a:

```
» x=t^2-1;
» y=t^3-t;
```

```
» pretty(x)
                                    2
                                   t  - 1
» pretty(y)
                                    3
                                   t  - t
» pretty(implicit(x,y))
                                 2    3    2
                                y  - x  - x
```

Sample run 2-b: (Previous example with base points)

Numerators and denominators multiplied by $(t^2 + 1)$.

```
» xt=t^4-1;
» yt=t^5-t;
» wt=t^2+1;

» pretty(implicitrat(xt,yt,wt))
                                   2    3    2
                                  y  - x  - x
```

Sample run 3-a: (Tacnode)

```
» x=t^3-6*t^2+9*t-2;
» y=t^2-4*t+4;
» w=2*t^4-16*t^3+40*t^2-32*t+9;
» pretty(x)
                                3      2
                               t  - 6 t  + 9 t - 2

» pretty(y)
                                  2
                                 t  - 4 t + 4

» pretty(w)
                            4        3        2
                         2 t  - 16 t  + 40 t  - 32 t + 9

» pretty(implicit(x,y,w))
                       2         3           2         4         4
                  3/4 y  - 3/2 y  - 9/4 y x  + 3/4 y  + 3/2 x
```

Sample run 3-b: (Tacnode with base points)

Numerators and denominators multiplied by $(t^2 - 1)$.

```
» xt=t^5-6*t^4+8*t^3+4*t^2-9*t+2;
» yt=t^4-4*t^3+3*t^2+4*t-4;
» wt=2*t^6-16*t^5+38*t^4-16*t^3-31*t^2+32*t-9;

» pretty(implicitrat(xt,yt,wt))
```

```
                2         4         4         3         2
- 9/4 y x  + 3/4 y  + 3/2 x  - 3/2 y  + 3/4 y
```

4.2 IMPLICITIZATION OF GENERALIZED TRIGONOMETRIC POLYNOMIALS CURVES

4.2.1 Algorithm

The following algorithm computes the implicit equation of a curve defined parametrically by generalized trigonometric functions, using the method mentioned in the previous chapter.

Algorithm 4.2: Implicitization of generalized trigonometric polynomial curves.

Input: $a_1, a_2, \ldots, a_n \in R$ such that $x(t) = \sum a_k \cos kt$, $y(t) = \sum a_k \sin kt \in R[t]$.

Output: $S(x, y) \in R[x, y]$ such that $S(x, y) = 0$ is the implicit equation of $[x(t), y(t)]$.

$a_0 = -(x + iy)$; $a_0' = -(x - iy)$;

create matrix S row-wise from the coefficients $a_0, a_0', a_1, \ldots, a_n$ as described in the previous chapter;

return det(S);

4.2.2 Implementation

The function has the following features:

1. The input is in the compact vector of coefficients form.
2. The function uses the built-in imaginary unit *i*.
3. The output is in symbolic format.

(M-File 2)

```
function [R] = implicitrig(a)
% Implicitrig Implicit equation
%    Implicitrig(a) returns the implicit equation of a generalized
%    trigonometric parametric one
%    R(x,y) = 0 is the implicit symbolic equation
%    [a] is a vector of the coefficients of the parametric
%    trigonometric polynomial, such that:
%    x(t) = a1 cos(t) + a2 cos(2t) + ... + an cos(nt)
%    y(t) = a1 sin(t) + a2 sin(2t) + ... + an sin(nt)

%define symbolic variables x and y and symbolic imaginary unit i
syms x y i;
```

```
n = length(a);          % degree of trigonometric polynomial

a = flipLR(a);          % flip left/right - most significant first
a = [a -(x+i*y)]; % add a0
for l = 1:n-1
   a = [a 0];           % add trailing zeros
end;

H = a;                  % H is the matrix of the resultant
                  %  initialized with first row
for l = 2:n
   a(end) = [];         % rotate right by deleting last zero
   a = [0 a];           % then adding a leading one
   H = [H;a];           % add resulting row to matrix
end;

a = flipLR(a);          % flip left/right
a(1) = -(x-i*y) ; % first conjugate row
H = [H;a];        % add to the resultant matrix

for l = 2:n
   a(end) = [];         % rotate right by deleting last zero
   a = [0 a];           % then adding a leading one
   H = [H;a];           % add resulting row to matrix
end;

format rat;       % rational format
R = expand( det(H) );   % fully expand determinant and return the resultant
format;                 % back to default

% End of file
```

4.2.3 Sample runs

Sample run 1:

$x = 5/2 \cos\theta - \cos 2\theta, \qquad y = 5/2 \sin\theta - \sin 2\theta$

```
» a=[5/2 -1]

a =
     5/2          -1
```

```
» R=implicitrig(a);
» pretty(R)
                                    2          2    4       2  2    4
            - 21/4 + 25/2 x - 33/4 x  - 33/4 y  + x  + 2 x  y  + y
```

Sample run 2:

$x = \cos 2\theta + 2/3 \cos 3\theta$, $y = \sin 2\theta + 2/3 \sin 3\theta$

```
» a=[0 1 2/3];
» R=implicitrig(a);
» pretty(R)
```

```
  80           55  2   55  2          4          2  2          4      2  4
- --- + 8/9 x - -- x  - -- y  + 10/3 x  + 20/3 x  y  + 10/3 y  - 3 y  x
  729          27       27

          2  4    6    6
    - 3 x  y  - y  - x
```

4.3 NEAR IMPLICITIZATION

4.3.1 Algorithms

The following are the algorithms for both methods of near implicitization that we devised in the previous chapter.

Algorithm 4.3: Fixed near implicitization.

Input: $x(t), y(t) \in R[t], n, t_0$.

Output: $S(x, y) \in R[x, y]$ such that $S(x, y) = 0$ is the approximate implicit equation of $[x(t), y(t)]$ near t_0.

$x :=$ Taylor expansion of $x(t)$ about t_0 using n terms;

$y :=$ Taylor expansion of $y(t)$ about t_0 using n terms;

$S(x, y) :=$ **call** algorithm 4.1 with $x(t) = x$ and $y(t) = y$;

return S;

Algorithm 4.4: Multiple near implicitization.

Input: $x(t), y(t) \in R[t], n, t_1, t_2, \ldots, t_k$.

Output: $S_1(x, y), S_2(x, y), \ldots, S_k(x, y) \in R[x, y]$ such that $S_i(x, y) = 0$ are the approximate implicit equations of $[x(t), y(t)]$ in the neighborhoods of $t_1, t_2, \ldots, t_k$ respectively.

for $i := 1, 2, \ldots, k$ **do**

$S_i(x, y)$:= **call** algorithm 4.3 with $x(t)$, $y(t)$, n and t_i ;

return S_i , i = 1, 2, …, k.

4.3.2 Implementation

The functions have the following features:

1. Variable number of input arguments supported.
2. Default values for non-initialized arguments are automatically assigned based on mathematically reasonable values.

Fixed near implicitization

(M-File 3)

```
function [R] = implicitall(xt,yt,acc,on)
%IMPLICITALL Implicit equation of a parametric one of any form
%   implicitall(xt,yt,acc,on): returns the approximate symbolic implicit
%   equation based on Taylor approximation of the parametric form
%   R(x,y): is the approximate implicit symbolic equation
%   xt, yt: are the symbolic input parametric functions for x(t), y(t)
%   acc: is the required degree of the Taylor expansion of x(t) and y(t)
%   on: is the point around which the expansion occurs

% Check number of input arguments and initialize default values
if (nargin < 2) error('Not enough input arguments')
  elseif (nargin == 2) acc = 5; on = 0;
  elseif (nargin == 3) on = 0;
end

% Taylor expansion of parametric equation
x = Taylor(xt, acc, on);
y = Taylor(yt, acc, on);

% Expand x and y as polynomials in t
x = expand(x)
y = expand(y)

R = implicitrat(x,y);         % Call rational implicitization function

% End of file
```

Multiple near implicitization

(M-File 4)

```
function [R] = multimplicit(xt,yt,acc,on)
% MULTIMPLICIT Implicit equation of a parametric one of any form
%   multimplicit(xt,yt,acc,on): returns the approximate symbolic implicit
%   equation based on Taylor approximation of the parametric form
%   R(x,y): is the vector of approximate implicit symbolic equations
%   xt, yt: are the symbolic input parametric functions for x(t), y(t)
%   acc: is the required degree of the Taylor expansion of x(t) and y(t)
%   on: is the vector of points around which the expansions occurs

% Check number of input arguments and initialize default values
if (nargin < 2) error('Not enough input arguments')
  elseif (nargin == 2) acc = 5; on = 0;
  elseif (nargin == 3) on = 0;
end

for i = 1:length(on)
  R(i) = implicitall(xt,yt,acc,on(i));
end

% End of file
```

4.3.3 Sample runs

Fixed near implicitization

Sample run 1-a: (Cycloid with degree 4)

$$x = \theta - \sin\theta, \quad y = 1 - \cos\theta, \quad 0 \le \theta \le 2\pi.$$

Taylor expansion around $\theta = \pi$, with highest degree = 4.

```
» syms t
» R=implicitall(t-sin(t),1-cos(t),4,3.14);
» pretty(R)

840772896868853105110332506199351152032106330580561  2
------------------------------------------------- x
374144419156711147060143317175368453031918731001856

        222060141882289628844436048817423471426354383723
    + ------------------------------------------------- x y
      467680523945888933825179146469210566289898413752 32
```

$$
\begin{aligned}
&\frac{94349903129012137342236522654340624640553}{467680523945888933825179146469210566289898413752 32}\,x^{3} \\
-\;&\frac{710887720654449943750293668731389762512962755}{187072209578355573530071658587684226515959365500928}\,x^{2}\,y \\
+\;&\frac{697427947588052897403629461176834255238653356 9}{292300327466180583640736966543256603931186508595 2}\,x\,y^{2} \\
-\;&\frac{243694215173712468735282594055177}{811296384146066816957890051440 64}\,y^{2} \\
+\;&\frac{253213383059728352696519837206 9}{507060240091291760598681282150 4}\,y^{3} \\
-\;&\frac{183281580378020347912916485518689 77}{129807421463370690713262408230502 4}\,x \\
+\;&\frac{607845660161387512457903284233}{405648192073033408478945025720 32}\,y + \frac{6979280796728801}{1125899906842624}
\end{aligned}
$$

The exact and approximate plots are shown in the figure below.

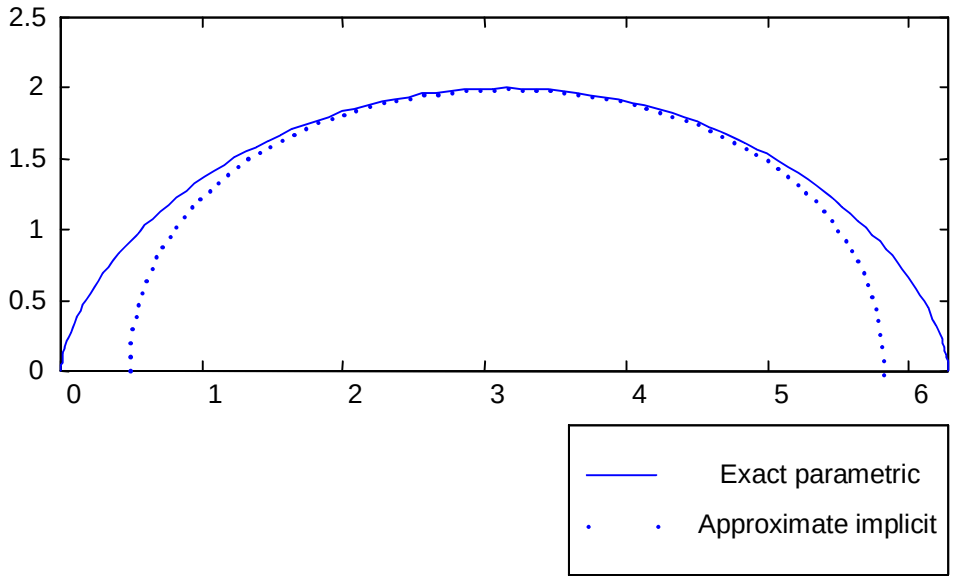


Figure 4.1. The cycloid: exact and approximated to 4 terms of Taylor's expansion.

<u>Sample run 1-b:</u> (Cycloid with degree 6)

$$x = \theta - \sin\theta, \quad y = 1 - \cos\theta, \quad 0 \le \theta \le 2\pi.$$

Taylor expansion around $\theta = \pi$, with highest degree = 6.

```
                                                                        /
877755463362566215985033238927840791046694867920280867741759276978892316 9  /
                                                                      /
    151771007205135083665582961470587414581438034300948400097797844510851 89\

                 4
    728165691392 x  y - 551127200166221713277292771154195728615795616337997\

                            /
    2601383308133155603371065  /  758855036025675418327914807352937070290719\
                          /

                                            3  2
    01715047420004889892225542594864082845696 x  y  - 67898412767778623520 7\

                                                                     /
    9417958927006746693726662934487100287661499706535848670990049  /  37942\
                                                                  /
    75180128377091639574036764685364535950857523710002444946112771297432041\

           3
    422848 x  y - 1387773707465711521660523588205896049021182975169282575 43\

                            /
    949983042745151251408050 3  /  3705346855594118253554271520278013051304 6\
                          /

                                         2  2
    395093004980492626426882532201484775 2 x  y

      122840217792264524670540780028298556512253439497946713237619251    3
    - --------------------------------------------------------------- x y  -
      514220174162876888173427869549172032807104958010493707296440 32

    6989807976769566764895012802523330323979359959073777064225784217673 79/3\
    794275180128377091639574036764685364535950857523710002444946112771297 43\

               5
    2041422848 x  + 558054806557452014103306014305539145986473824911801140 9\

                               /
    04380438682284537160939852987 7  /  94856879503209427290989350919117134 1\
                             /

                                                   3
    1339877143809275006112365281928243580103557 12 x

      200234172506554630840441624438498487068727499115  3
    + ------------------------------------------------ y
        8920298079412249256614287309059344602392166 4
```

```
  3021306491322073892898520810227111911291250851164772031464439334 1    2
+ ------------------------------------------------------------------- x y
   1285550435407192220433569673872930082017762395026234268241100 8


  1472308267573653446951892335935607831063970816932824732243670 7     4
- ------------------------------------------------------------------- x y
  1028440348325753776346855739098344065614209916020987414592880 64


  6414566412974927848045468637837460918122413391  5
+ ---------------------------------------------- y
  3568119231764899702645714923623737840956866 56


  2984215628479857376447088660411958183589814613839672780388526261852 3  2
- ------------------------------------------------------------------- x
    1645504557321206042154969182557350504982735865633579863348609024


  5815405845345275828394628833551720348267633475201260819925946 5
- ------------------------------------------------------------- x y
  1606938044258990275541962092341162602522202993782792835301376


  1686332463351875464449608055911041951612217433 1  2
+ ---------------------------------------------- y
    5575186299632655785383929568162090376495104


  7775276768207427785730365114298450892535126797 79
- ------------------------------------------------ x
   3568119231764899702645714923623737840956866 56


  3788329194915960219982727031134376216435727
+ ---------------------------------------- y + 54069229193764370880\
  1393796574908163946345982392040522594123776


                                                                  /
9188652155499320956116259920348932461739338373932732797 19  /  118571099\
                                                             /

3790117841137366886488964176417484642976159375764045660241030447512944 6\


   2  3
4 x  y  - 3552687238741526362270862949651776278262129923396665874171913\


                          /
913644991570562720302189  /  7588550360256754183279148073529370729071 90\
                         /


                                        4
1715047420004889892225542594864082845696 x


  2676747894140305417418001522445174167948124754 3  4
+ ---------------------------------------------- y  + 499952341654287\
   8920298079412249256614287309059344602392166 4


                                                                    /
821029427020567066249764779483300058378409625552450209524337552433  /
                                                                   /

2964277484475294602843417216222410441043711607440398439410114150602576 1\
```

```
           2     16819648904504317010473341480 7
187823616 x  y - ------------------------------
                 15474250491067253436239052 8
```

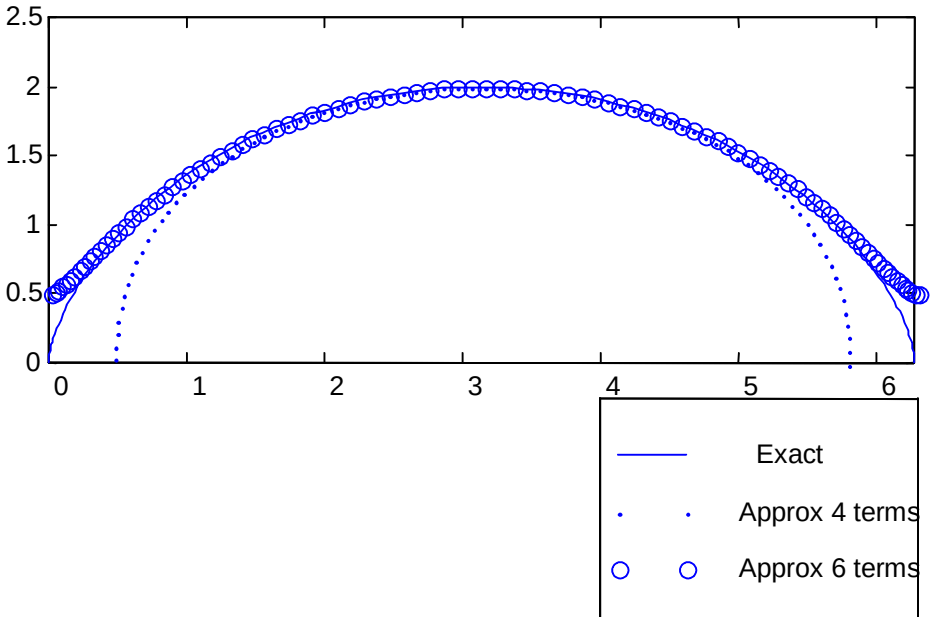


Figure 4.2. The cycloid: exact and approximated to 4 and 6 terms of Taylor's expansion.

Sample run 2: (Archimedean spiral)

$$x = \theta\cos\theta, \qquad y = \theta\sin\theta, \qquad 0 \le \theta \le \pi.$$

Taylor expansion around $\theta = \pi/2$, with highest degree = 4.

```
» syms t
» R=implicitall(t*cos(t),t*sin(t),4,3.14/2);
» pretty(R)
```

```
21670896373576208318673715969195377796624348503 1  2
------------------------------------------------- x
91343852333181432387730302044767688728495783936

       872965203834573551975394664454016004089608551 9
     + ------------------------------------------------- x y
       114179815416476790484662877555959610910619729 92
```

$$+\frac{865512221973315284830465955496075051208832171293}{182687704666362864775460604089535377456991567872}x^3$$

$$+\frac{135622129864887069635961170422116550756720753309}{182687704666362864775460604089535377456991567872}x^2y$$

$$+\frac{442737877306548123949536640316486348197333532 7}{114179815416476790484662877555959610910619729 92}xy^2$$

$$+\frac{205361304062612982716731424655369}{405648192073033408478945025720 32}y^2$$

$$+\frac{273855667187939895638258241778 5}{405648192073033408478945025720 32}y^3$$

$$+\frac{560939800037787854051166128133 73}{202824096036516704239472512860 16}x$$

$$-\frac{175860795400626713133792848425681}{202824096036516704239472512860 16}y+\frac{1951784264959249}{2251799813685248}$$

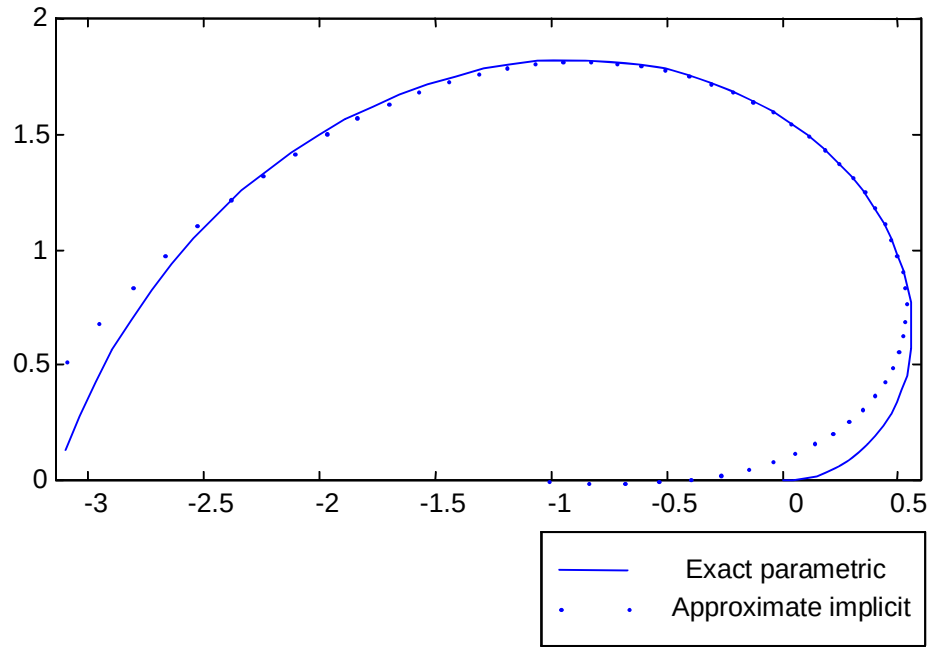


Figure 4.3. The Archimedean spiral: exact and approximated to 4 terms of Taylor's expansion.

Multiple near implicitization

Sample run 1: (Cycloid)

$$x = \theta - \sin\theta, \quad y = 1 - \cos\theta, \quad 0 \leq \theta \leq 2\pi.$$

Taylor expansions around $\theta = \pi/2$ and $3\pi/2$, with highest degree = 4 in both cases.

```
» R=multimplicit(t-sin(t),1-cos(t),4,[3.14/2,3*3.14/2]);
```

The first curve is plotted in the interval $0 \leq \theta \leq \pi$ and the second in $\pi \leq \theta \leq 2\pi$. The following plot is obtained using Matlab.

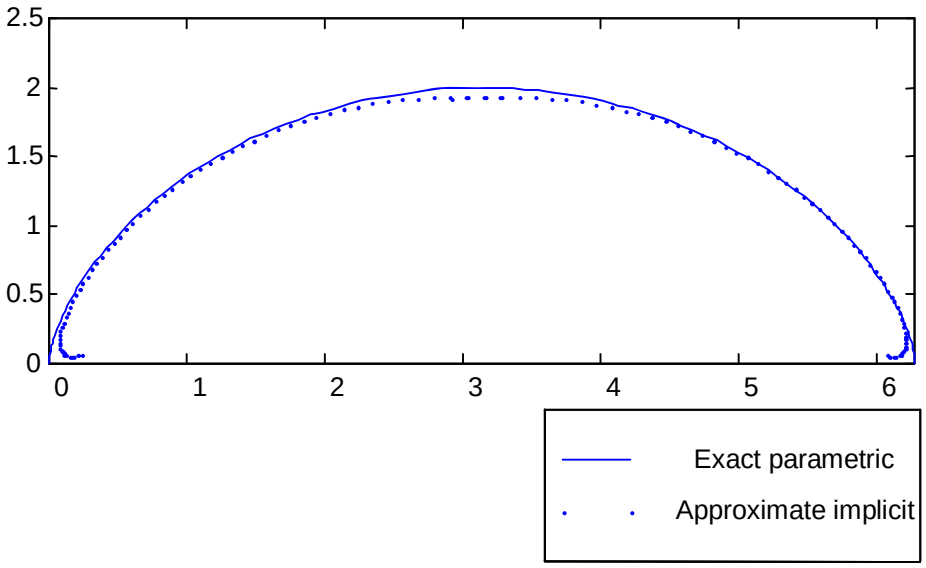


Figure 4.4. The cycloid implicitized around π/2 and 3π/2 with 4 terms of Taylor's expansion.

Sample run 2: (Archimedean spiral)

$$x = \theta\cos\theta, \quad y = \theta\sin\theta, \quad 0 \leq \theta \leq \pi.$$

Taylor expansion around $\theta = \pi/4$ and $3\pi/4$, with highest degree = 4 in both cases.

```
» R=multimplicit(t*cos(t),t*sin(t),4,[3.14/4,3*3.14/4]);
```

The first curve is plotted in the interval $0 \leq \theta \leq \pi/2$ and the second in the interval $\pi \leq \theta \leq 3\pi/4$. The following plot is obtained using Matlab.

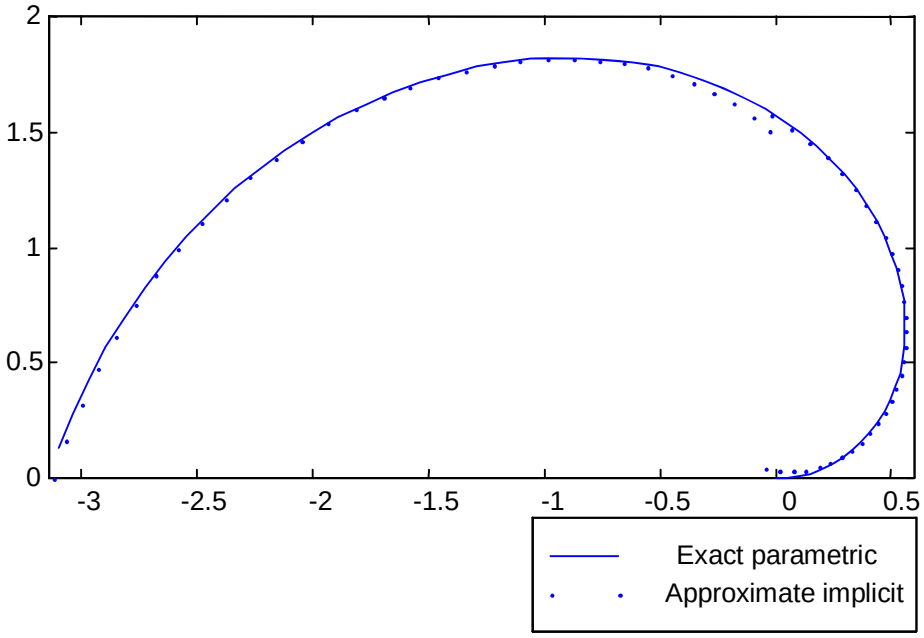


Figure 4.5. The Archimedean spiral implicitized around $\pi/4$ and $3\pi/4$ with 4 terms of Taylor's expansion.

Chapter 5: CONCLUSION AND FUTURE WORK

"Science is what we understand well enough to explain to a computer. Art is everything else we do. During the past several years, an important part of mathematics has been transformed from an Art to a Science. No longer do we need to get a brilliant insight in order to evaluate sums of binomial coefficients, and many similar formulas that arise frequently in practice. We can now follow a mechanical procedure and discover the answers quite systematically." [Donald Knuth, 1995].

The main purpose of this thesis has been to convey this idea. First, we have briefly reviewed a wide variety of computer algebra problems and the tools used to automate their solution. Then, one specific problem, namely that of implicitization of parametric functions, is investigated in depth. We have searched as many implicitization methods as the literature provides, devised algorithms for these methods and implemented them using the Matlab software package.

Additionally, we have tackled the problem when a direct method is not available to deal with the parametric functions. Our method, which we called *near implicitization*, approximates the problem in order to be able to handle it. We have proposed two ways to attack the problem, compromising between accuracy and running time.

In brief, our algorithms and implementations have the following features:

- We have tried to cover practically all forms of parametrically defined plane curves. Efficiency was a major concern, but simplicity of the output results was also considered.
- Implementation on a popular software package such as Matlab benefits from its large "toolbox" and makes our M-files (Matlab functions) available to a wide area of users.
- Most of our implementations accept symbolic input and deliver symbolic results. This feature illustrates the basic topic of the thesis: computer algebra.

Future work and open points

Our emphasis in the present work was on the implicitization of plane curves. Suggested work than can be added to ours is proposed hereafter.

- Adding a third dimension and promoting to the implicitization of 3D curves.
- Implementation of more efficient methods, where applicable, such as the method of moving conics.
- The problem of implicitization of surfaces is still far from a complete solution. This is an open point where research is taking place.

REFERENCES

[1] Joachim von zur Gathen and Jürgen Gerhard: Modern Computer Algebra. Cambridge University Press, 1999.

[2] Franz Winkler: Polynomial Algorithms in Computer Algebra. Springer-Verlag, 1996.

[3] Bhubaneswar Mishra: Algorithmic Algebra. Springer-Verlag, 1993.

[4] Franz Winkler: Advances and Problems in Algebraic Computation. Proceedings of the Vienna Conference, June 1999.

[5] Hesham El-Sayed: An Interactive System for Symbolic Computation. MSc thesis, Computer Science Dept., Eng. Faculty, Alexandria Univ., 1990.

[6] David M. Burton: Abstract and Linear Algebra. Addison-Wesley Publishing Co., 1972.

[7] John F. Humphreys: A Course in Group Theory. Oxford University Press, 1996.

[8] Anthony N. Michel and Charles J. Herget: Mathematical Foundations in Engineering and Science. Prentice-Hall Inc.

[9] Ahmed H. El Sherif: Generation of Plane Curves by Rolling. 1989.

[10] Tom Sederberg, Ron Goldman and Hang Du: Implicitizing Rational Curves by the Method of Moving Algebraic Curves. J. Symbolic Computation (23), 1997.

[11] Hoon Hong: Implicitization of Curves Parameterized by Generalized Trigonometric Polynomials. Research Institute for Symbolic Computation, Linz, Austria, 1995.

[12] Hoon Hong and Joseph Schicho: Algorithms for Trigonometric Curves (Simplification, Implicitization, Parameterization). Research Institute for Symbolic Computation, Linz, Austria, 1997.

[13] Ming Zhang: Topics in Resultants and Implicitization, Ph.D. thesis, Department of Computer Science, Rice University, 2000.

[14] R. N. Goldman, E. W. Chionh, and M. Zhang: Efficient Implicitization of Rational Surfaces by Moving Planes. Proceedings of the ASCM, 2000.

[15] R. N. Goldman, E. W. Chionh, and M. Zhang: Fast Computation of the Bezout and Dixon Resultant Matrices. Submitted to Journal of Applied & Computational Mathematics.

[16] R. N. Goldman, D. Cox, and M. Zhang: On the Validity of Implicitization by Moving

Quadrics for Rational Surfaces with No Base Points. J. Symbolic Computation (29), 2000.

[17] R. N. Goldman, E. W. Chionh, and M. Zhang: On a Relationship between the Moving Line and Moving Conic Coefficient Matrices. Computer Aided Geometric Design 16, 1999.

[18] R. N. Goldman, E. W. Chionh and M. Zhang: Transformations and Transitions from the Sylvester to the Bézout Resultant. Technical report TR99, 1999.

[19] Sandra Licciardi and Teo Mora: Implicitization of Hypersurfaces and Curves by the Primbasissatz and Basis Conversion. Presented at the Meeting Algebraic Algorithms for Geometric Reasoning, Linz, 1992.

[20] J. Rafael Sendra and Franz Winkler: Parameterization of Algebraic Curves Over Optimal Field Extensions. J. Symbolic Computation (23), 1995.

[21] Joseph Schicho: The Parameterization Problem for Algebraic Surfaces. Research Institute for Symbolic Computation, Linz, Austria, 1999.

[22] Dimitrios Poulakis and Evaggelos Voskos: On the Practical Solution of Genus Zero Diophantine Equations. J. Symbolic Computation (30), 2000.

[23] Erik Hillgarter: The Classification Problem for Point Symmetries of 2^{nd} Order PDE's in one Dependent and two Independent Variables, 1999.

[24] Günter Czichowski: Lie Theory of Differential Equations and Computer Algebra. Seminar Sophus Lie, 1991.

[25] E. Tournier: Computer Algebra and Differential Equations. Cambridge University Press, 1994.

[26] Wolfram Koepf: The Identification Problem for Transcendental Functions. Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB), 1995.

[27] C. Ray Wylie, William R. Kenan: Advanced Engineering Mathematics. McGraw-Hill, fourth edition, 1975.

[28] Catherine C. McGeoch: Experimental Analysis of Algorithms. Notices of the American Mathematical Society (48), 2001.

[29] High Performance Numeric Computation and Visualization Software, Matlab User's Guide, 1992.

APPENDIX A: Computer Algebra Systems

The major purpose of a *Computer Algebra System* is to manipulate a formula symbolically using the computer. For example, expanding, factorizing, root finding, or simplifying an algebraic polynomial, are some of the common uses of such systems. However, many systems have gone far beyond that and also offer other functionalities like numerical calculation, graphics, and simulations.

Many Computer Algebra Systems are currently available in the market. In what follows, a brief overview of some of these is presented. For more information, most of the systems presented have web sites, some publish journals regularly, or even organize conferences and workshops.

Axiom

AXIOM is a powerful computer algebra system which provides a complete environment for anyone needing to manipulate and solve mathematical formulae. Its application is wide-ranging, from pure mathematics research through branches of physics, chemistry, biology and engineering to financial modelling and cryptography.

Macsyma

Macsyma is a powerful mathematical software package, while remaining ease of use. It contains facilities for algebra, linear algebra, calculus, numerical analysis, on-line help system, versatile graphics, and system utilities. Macsyma includes 1,300 executable demonstrations easily accessible at many points in the help system. Also hypertext descriptions of 2,900 topics, a browser with 900 topics and commands, and 900 type-in command templates.

Recent mathematical improvements include enhanced speed in solving linear and algebraic equations, stochastic mathematics, better evaluation of special functions, and enhanced tensor analysis. It is smarter about using the algebraic signs of expressions to simplify results.

MAGMA

Magma is a radically new system designed to solve computationally hard problems in algebra, number theory, geometry and combinatorics. Magma provides a mathematically

rigorous environment for computing with algebraic and geometric objects. The basic design principles are:

- Algebraic structures and their morphisms as first class objects.
- Language design reflecting the structure of modern algebra.
- Kernel implementation of important structures to achieve outstanding performance.
- A strong typing scheme derived from modern algebra whereby types correspond to algebraic structures (magmas) to provide mathematical rigor and clarity.
- Seamless integration of algorithmic and database knowledge.
- Structures supported range across group theory (finitely presented groups, blackbox groups, abelian groups, soluble groups, permutation groups, matrix groups, finitely presented semigroups, and characters of finite groups), rings (the integers with optimized arithmetic, residue class rings, univariate and multivariate polynomial rings, invariate rings of finite groups, valuation rings), fields (finite fields, quadratic fields, local fields, cyclotomic fields, number fields, rational function fields, and the rationals), algebras (group algebras, matrix algebras, finitely presented algebras, associative algebras, and algebras defined by structure constants), power and Laurent series,vector spaces, modules, lattices, algebraic geometry (elliptic curves, zero-dimensional varieties), graphs, incidence structures and designs, finite planes, enumerative combinatorics, and coding theory.

Maple

Maple V is a powerful mathematical problem-solving and visualization system used world-wide in education, research, and industry. Its principal strength is its symbolic problem solving algorithms. Maple V also has a programming language (like Pascal) which allows to extend the library of 2500+ functions.

Mathematica

Industry leading fully integrated environment for technical and scientific computing. Quick, accurate, numeric and symbolic computations. Automatic arbitrary precision control for numeric calculations. Creates fully interactive and customizable technical documents including editable/evaluable typeset quality formulas, 2-D and 3-D graphics, sound, and animations.

REDUCE

REDUCE is an interactive program designed for general algebraic computations of interest to mathematicians, scientists, and engineers.

MuPAD

MuPAD is a system for symbolic and numeric computation, parallel mathematical programming, and mathematical visualization. It is intended to be a 'general purpose' computer algebra system. Programming in MuPAD's own programming language is supported by a comfortable source code debugger. The concept of dynamic modules enables the user to integrate C/C++ functions as well as complete software packages into MuPAD. MuPAD has easy-to-use language constructs for parallel programming. A pre-release version for parallel programming exists for SGI, Sequent, and Sun multiprocessor machines. Window-based user interfaces for MuPAD exist for the X-Window-System, Apple Macintosh, and Windows 95/NT.

DERIVE

DERIVE is a symbolic mathematics package which includes: exact arithmetic, factoring, partial fraction expansion, equation solving, 2D and 3D plots, derivatives, integrals, differential equations, Taylor and Fourier Series, Laplace Transforms, vectors and matrices. One drawback of DERIVE is that it cannot be programmed efficiently to perform a given sequence of operations. However, the lack of this facility is compensated by the ease-of-use which the menu-driven nature of the package provides, as opposed to a command-driven computer algebra package, such as MuPad or Reduce.

A more detailed list of available systems can be found at: SAL Computer Algebra Systems - [`http://chpc06.ch.unito.it/linux/A/1/`]. Here is an overview:

One) General purpose systems:

Axiom: A computer algebra system with powerful symbolic solver.

bernina: Interactive program with interface to a computer algebra library.

Computer Algebra Kit: Collection of small programs for computer algebra.

CASA: computer algebra software for constructive algebraic geometry in Maple.

CoCoA: a special-purpose system for doing Computations in Commutative Algebra.

DrMath: online symbolic math and computer algebra system.

FELIX: computations in and with algebraic structures and substructures.

FORM: symbolic manipulation for large mathematical problems.

GAMS: a high-level modeling system for mathematical programming problems.

GAP: computational discrete algebra emphasized on group theory.

GiNaC: open framework for symbolic computation within C++.

GRG: a computer algebra program for differential geometry, gravitation and field theory.

GRTensorII: computer algebra package for differential geometry.

HartMath: a computer algebra program written in Java.

Kalamaris : mathematical package similar to Mathematica.

Interpreter for symbolic manipulation: a simple computer algebra system in C++ or Java.

JACAL: symbolic simplification and manipulation of equations.

lundin.Symbolic Math: Java class library for symbolic differentiation and evaluation.

Macaulay 2: algebraic geometry and commutative algebra.

Macsyma: a powerful and general purpose mathematical tools.

MAGMA: a system for algebra, number theory, geometry and combinatorics.

Maple: numerical computation, symbolic algebra, functions, graphics, programming.

MAS: experimental computer algebra system.

Mathematica: numerical computation, symbolic algebra, functions, graphics, programming.

Mathomatic: artificially intelligent algebraic manipualtor program.

MAXIMA: computer algebra system for symbolic and numerical computations.

Symaxx: a frontend for Maxima computer algebra system.

Mock-Mma: simple mock-up of Mathematica in Common Lisp.

MuPAD: symbolic and numeric computation, parallel programming and visualization.

Punimax: computer algebra system for symbolic and numerical computations.

Reduce: general algebraic computations.

Redten: symbolic algebra package for tensor objects.

Ricci: symbolic tensor computations for differential geometry.

Risa/Asir: experimental computer algebra system with programming libraries.

SACLIB: library of C programs for computer algebra.

SAML: a simple algebraic math library.

SimLab: CLISP code for computer algebra substrate and triangulations of 2D areas.

Singular: a computer algebra system for computing information about singularities.

SISYPHOS: computing in modular group algebras of p-groups.

Software from Symbolic Computation Group: symbolic computation in physics.

text_deriv: a package designed to do simple symbolic derivatives.

Yacas: a small and highly flexible computer algebra language.

Two) Specialized systems:

Algae: a high-level interpreted language for numerical analysis.

AUTOLEV: a symbol manipulation program for analyzing the dynamics of mechanical systems.

CAM C++ Class Libraries: graphics, matrix/vector and symbolic function classes.

FOAM: computer algebra program for high energy physics.

GNU Calc: an advanced calculator that runs as part of the GNU Emacs environment.

meditor: a text editor with some symbolic computation capabilities.

Pari/GP: formal computations on recursive types at high speed (CA system in number theory).

SHEEP: a computer algebra system designed primarily for general relativity.

SIMATH: computer algebra system for number theory.

Simple Lisp: a simple lisp interpreter with symbolic math library for tensors.

Tmath: a Tcl interface to Matlab and Mathematica.

APPENDIX B: History of Symbolic Computation

The primary motivation for the initial development of digital computers was to solve numeric problems. Because of this tendency for using a computer to solve numeric problems, there has been a major shift in the technology of applied mathematics. Prior to the existence of digital computers, numerical analysis was a very tedious subject. With the emergence of computers, numerical analysis method developed much more and analytic techniques tended to be ignored. [5]

Even in our days, many mathematicians think there is a division of labor between man and computer: a person applies the appropriate algebraic transformations to the problem at hand and, finally, arrives at a program which then can be left to a "number crunching" computer. However, this is an error. Computers are general-purpose machines that can also be used for transforming, combining and computing symbolic algebraic expressions. In other words, computer can not only deal with numbers, but also with abstract symbols representing mathematical formulas. This fact has been realized much later and is only now gainig acceptance among mathematicians and engineers. [2]

One cannot exactly determine when the field of computer algebra began, but at least some cornerstones can be outlined as follows:

- The field originated from the needs of physics researchers.
- The first programs dealing with formulas were written by physicists in order to save them from performing long, tedious, error prone calculations by hand.
- 1955 - First programs for formula derivation.
- 1965 - First general-purpose computer systems working with algebraic expressions.
- 1975 - A new research field emerged with its own conferences, journals, etc.
- 1990 - General spreading of computer algebra systems into almost all branches of science.

[Cf.: Richard Liska, `http://www-troja.fjfi.cvut.cz/~liska/ca/`]

Chronology of computer algebra systems

Brian Evans, [`evans@eedsp.gatech.edu`] has compiled the following chronology of computer algebra systems.

System name	Year	Related systems	e-mail address, tel.
ALPAK	1964	*ALTRAN*	(Bell Labs)
ALTRAN	1968		(Bell Labs)
FORMULA (Algol)			
FORMAC		*FORMAC (PL/I)*	(IBM)
FORMAC (PL/I)			(IBM)
MATHLAB (DECUS)	1968	*MACSYMA*	(DEC)
CAMAL			
REDUCE	1968		reduce-netlib@rand.org
MACSYMA	1970	*Symbolics Macsyma, VAXIMA, DOE-Macsyma, ALJABR, ParaMacs*	(See Below)
SchoonShip	1971		archive.umich.edu (FTP)
muMath	1979	*Derive*	
VAXIMA	1980		(312) 972-7250
SMP	1982	*Mathematica*	NOT ON MARKET
Symbolics MACSYMA	1983		macsyma-service@symbolics.com
DOE-Macsyma	1984	*ALJABR*	gcook@llnl.gov
Maple	1985		wmsi@daisy.waterloo.edu
MathCAD	1985(?)	*Mathcad*	1-800-MATHCAD
Powermath	1985		NOT ON MARKET
REDUCE/PC	1986		reduce-netlib@rand.org
Derive	1988		(Soft Warehouse Inc.)
Mathematica	1988		info@wri.com
Theorist	1988		(415) 543-2252 (Prescience Corp)
PARI	1988(?)		ftp to math.ucla.edu
FORM	1989		form@can.nl
MACSYMA/PC	1989		macsyma-service@symbolics.com
ALJABR	1991		aljabr@fpr.com
Mathcad	1991		1-800-MATHCAD
SymbMath	1991		chen@deakin.oz.au
Axiom	1991		(708) 971-2337
ParaMacs	1991		lph@paradigm.com
SIMATH	1992		marc@math.uni-sb.de