Εθνικο και Καποδιστριακο Πανεπιστημιο Αθηνων
Σχολη Θετικων Επιστημων

ΜΠΛΑ
Προγραμμα Μεταπτυχιακων Σπουδων

Διπλωματική Εργασία:

Real Solving on Algebraic Systems of Small Dimension

Φοιτητής
Διώχνος Δημήτρης
Αριθμός Μητρώου: 200306

Επιβλέπων Καθηγητής
Ιωάννης Εμίρης

Αθήνα
Ιούνιος 2007

Η παρούσα Διπλωματική Εργασία
εκπονήθηκε στα πλαίσια των σπουδών
για την απόκτηση του
Μεταπτυχιακού Διπλώματος Ειδίκευσης
στη
Λογική και Θεωρία Αλγορίθμων και Υπολογισμού
που απονέμει το
Τμήμα Μαθηματικών
του
Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών

Εγκρίθηκε την 8/3/2007 από Εξεταστική Επιτροπή
αποτελούμενη από τους:

Ονοματεπώνυμο	**Βαθμίδα**	**Υπογραφή**
1. Γ. Εμίρης (Επιβλέπων)	Αναπληρωτής Καθηγητής Τμήματος Πληροφορικής και Τηλεπικοινωνιών Ε.Κ.Π.Α.	.....
2. Η. Κουτσουπιάς	Καθηγητής Τμήματος Πληροφορικής και Τηλεπικοινωνιών Ε.Κ.Π.Α.	.....
3. Ε. Ράπτης	Αναπληρωτής Καθηγητής Τμήματος Μαθηματικών Ε.Κ.Π.Α.	.....

Στην οικογένειά μου

Ευχαριστίες

Είμαι πολύ χαρούμενος που είχα την τιμή να συνεργαστώ με τον καθηγητή κ. Γιάννη Εμίρη για τη διπλωματική μου εργασία στο $\mu\prod\lambda\forall$. Τα μαθήματά του καθώς επίσης και οι παρατηρήσεις του στις κατ' ιδίαν συζητήσεις που είχαμε, είτε αυτές αναφέρονταν αποκλειστικά στη πεδίο της διπλωματικής μου είτε όχι, ήταν πάντοτε μια πηγή έμπνευσης και πρόσφεραν χώρο για δημιουργική δουλειά. Επιπλέον, ο χαρακτήρας του θα είναι πάντοτε πρότυπο για εμένα. Δουλεύοντας δίπλα του έμαθα πως μπορεί κανείς να ασχολείται εντατικά με διάφορα προβλήματα και να διαχειρίζεται κρίσεις χωρίς όμως να παραγκωνίζει και την προσωπική του ζωή. Το λίγοτερο που μπορώ να κάνω είναι να τον ευχαριστήσω για κάθε στιγμή συνεργασίας μας.

Επίσης οι καθηγητές κύριοι Ηλίας Κουτσουπιάς και Ευάγγελος Ράπτης μου έκαναν την τιμή να είναι στην τριμελή επιτροπή μου και τους ευχαριστώ θερμά γι' αυτό. Και οι δύο, κυρίως μέσα από τα μαθήματά τους, μου πρόσφεραν με τις συζητήσεις τους μια νέα προοπτική σε θέματα αλγορίθμων και άλγεβρας αντίστοιχα. Σίγουρα η κατανόηση διαφόρων εννοιών δεν θα ήταν το ίδιο εύκολη χωρίς τη συμβολή και των δύο.

Ακόμη, δεν μπορώ παρά να πω ένα μεγάλο ευχαριστώ και στον Ηλία Τσιγαρίδα. Ο Ηλίας τελείωνε το διδακτορικό του την εποχή που ξεκινούσα τη διπλωματική μου σε ένα θέμα το οποίο στην ουσία βασιζόταν σε ένα μεγάλο μέρος στη δουλειά που είχε κάνει ο ίδιος έως τότε. Οι συζητήσεις μας ήταν πάντοτε ένα ανεκτίμητο κράμα μαθηματικών και προγραμματισμού αφού το πάθος του για την ομορφιά που βρίσκεται πίσω από τις εξίσωσεις καθώς επίσης και η δινή προγραμματιστική του εμπειρία μας έφερνε πάντοτε να ασχολούμαστε με την καρδιά των προβλημάτων που αντιμετωπίσαμε. Δίχως τη βοήθεια του Ηλία είναι αμφίβολο αν η διπλωματική μου θα είχε την ίδια μορφή.

Φυσικά δεν μπορώ να παραλείψω και την παρέα τόσο του Ηλία όσο και των υπολοίπων παιδιών στο γραφείο. Δεν είναι λίγες οι φορές που χρειάστηκε να δουλέψουμε ως αργά τη νύχτα στη σχολή και η παρέα τους αλλά και οι παρεμφερείς ανησυχίες τους βοηθούσαν να ξεπεραστούν πιο εύκολα τα όποια εμπόδια αντιμετωπίζαμε. Θέλω λοιπόν να πω κι ένα ευχαριστώ στους Χρήστο Κοναξή και Γιώργο Τζούμα.

Τέλος θα ήθελα να ευχαριστήσω θερμά και τους καθηγητές μου κυρίους Γιάννη Μοσχοβάκη και Κώστα Δημητρακόπουλο που μου έδωσαν την ευκαιρία να φοιτήσω στο $\mu\prod\lambda\forall$. Μέσα από τα μαθήματα των ίδιων αλλά και των υπολοίπων συναδέλφων τους στο $\mu\prod\lambda\forall$ είχα την ευκαιρία να γνωρίσω μια πιο μαθηματική πλευρά της πληροφορικής την οποία δύσκολα πιστεύω πως θα μπορούσα να βρω σε οποιοδήποτε άλλο μεταπτυχιακό πρόγραμμα στην Ελλάδα. Τους είμαι πραγματικά ευγνώμων.

Δημήτρης Διώχνος,
Αθήνα, 14 Ιουνίου 2007.

Περίληψη

Η παρούσα διπλωματική ασχολείται με την ακριβή (exact) επίλυση στους πραγματικούς αριθμούς, καλώς ορισμένων πολυωνυμικών συστημάτων. Το κύριο πρόβλημα είναι η εύρεση όλων των πραγματικών λύσεων του συστήματος και ο υπολογισμός των πολλαπλοτήτων στα σημεία τομής. Για το σκοπό αυτό παρουσιάζονται τρεις αλγόριθμοι και αναλύεται η δυαδική πολυπλοκότητά τους. Οι δύο από τους τρεις αλγορίθμους επιτυγχάνουν ένα φράγμα $\widetilde{\mathcal{O}}_{\mathrm{B}}(\mathrm{N}^{12})$ [DET07a, DET07b], ξεχνώντας λογαριθμικούς παράγοντες, ενώ το προηγούμενο καλύτερο φράγμα ήταν $\widetilde{\mathcal{O}}_{\mathrm{B}}(\mathrm{N}^{14})$, όπου το N φράσσει το βαθμό και το δυαδικό μήκος των συντελεστών των πολυωνύμων εισόδου. Η έξοδος των αλγορίθμων είναι οι ακριβείς συντεταγμένες των σημείων τομής, δηλαδή διατεταγμένα ζεύγη πραγματικών αλγεβρικών αριθμών και δίνονται υπό μορφή διαστημάτων απομόνωσης (isolating interval representation).

Το κύριο εργαλείο είναι οι ακολουθίες υποαπαλοιφουσών (subresultants) και Sturm-Habicht, οι οποίες εξετάζονται σε πολλές μεταβλητές μέσω της τεχνικής της δυαδικής κατάτμησης. Για την επίτευξη των φραγμάτων χρησιμοποιούνται πρόσφατα αποτελέσματα πολυπλοκότητας στην απομόνωση των ριζών πολυωνύμων μιας μεταβλητής. Ακόμη παρουσιάζεται νέο φράγμα στον υπολογισμό προσήμου πολυωνύμου σε δύο μεταβλητές. Επίσης, οι αλγόριθμοι επίλυσης σε δύο μεταβλητές εφαρμόζονται και για τον υπολογισμό της τοπολογίας μιας πραγματικής αλγεβρικής καμπύλης.

Τέλος, όλοι οι αλγόριθμοι έχουν υλοποιηθεί στο MAPLE με πολύ ενθαρρυντικά αποτελέσματα. Η υλοποίηση χρησιμοποιεί αριθμητικά φίλτρα προκειμένου να επιταχύνονται οι υπολογισμοί όταν οι ρίζες είναι καλά διαχωρισμένες. Η δυναμική της βιβλιοθήκης παρουσιάζεται με τη βοήθεια πειραμάτων συγκριτικά με άλλες ευρέως διαδεδομένες βιβλιοθήκες όπως είναι το FGB/RS, τρεις αλγόριθμοι της SYNAPS (STURM, SUBDIV και NEWMAC), και δύο ελαφρά αλλαγμένες μορφές των INSULATE και TOP τα οποία υπολογίζουν την τοπολογία μιας πραγματικής αλγεβρικής καμπύλης.

Abstract

This thesis is concerned with exact real solving of well-constrained, bivariate algebraic systems. The main problem is to isolate all real solutions of the system and determine their intersection multiplicities. Three projection-based algorithms are presented and their asymptotic bit complexity is analyzed. This leads to a bound of $\widetilde{\mathcal{O}}_B(N^{12})$ [DET07a, DET07b], when ignoring polylogarithmic factors, whereas the previous record bound was in $\widetilde{\mathcal{O}}_B(N^{14})$, where N bounds the degree and the bitsize of the input polynomials. The output of the solvers are pairs of real algebraic numbers in isolating interval representation.

The main tool is Sturm-Habicht and subresultant remainder sequences, extended to several variables by the technique of binary segmentation. In order to achieve the bounds, recent advances on the complexity of univariate root isolation are exploited. New bound for the sign evaluation of bivariate polynomials over a pair of real algebraic numbers is also presented. Moreover, the algorithms for bivariate real solving are applied to compute the topology of real plane algebraic curves.

Lastly, all algorithms have been implemented in MAPLE with very encouraging results. The implementation uses numeric filtering to speed up computation when the roots are well-separated. We illustrate it by experiments against well-established libraries such as FGB/RS, 3 SYNAPS solvers (STURM, SUBDIV, and NEWMAC), and modified versions of INSULATE and TOP which compute the topology of real plane algebraic curves.

Contents

List of Tables

List of Algorithms

Chapter 1

Introduction

The problem of well-constrained algebraic system solving is fundamental. However, most of the algorithms treat the general case or consider solutions over an algebraically closed field [Can87, LL91, Emi95, EV99, Mou96, MP97, MP98, Mou99]. This thesis is based on the results presented in [DET07a, DET07b] and focuses on real solving in the bivariate case in order to provide precise complexity bounds and study different algorithms in practice. The work can be considered as an extension to the bivariate solvers presented in [Tsi06]. The idea is that if someone treats specific cases on their own it is possible to obtain better bounds than those provided in the general case. This is important in several applications ranging from nonlinear computational geometry and computer-aided geometric design to real quantifier elimination and robotics. A question of independent interest is to compute the topology of a plane real algebraic curve, which is also studied in this thesis.

The algorithms isolate all common real roots inside non-overlapping rational rectangles, and determine the intersection multiplicity per root. The output is pairs of real algebraic numbers. Three projection-based algorithms are presented and their asymptotic bit complexity is analyzed. Similarly to other works, $\mathcal{O}_{\mathrm{B}}$ means bit complexity and $\widetilde{\mathcal{O}}, \widetilde{\mathcal{O}}_{\mathrm{B}}$ means that we are ignoring polylogarithmic factors. This leads to a bound of $\widetilde{\mathcal{O}}_{\mathrm{B}}(\mathrm{N}^{12})$, whereas the previous record bound was $\widetilde{\mathcal{O}}_{\mathrm{B}}(\mathrm{N}^{14})$ [GVEK96, BPM06], derived from the closely related problem of computing the topology of real plane algebraic curves, where N bounds the degree and the bitsize of the input polynomials. The approach in [GVEK96] depends on Thom's encoding for representing the real roots of a univariate polynomial. Real algebraic numbers in this thesis are represented in isolating interval representation, since it is more intuitive, it is used in applications, and the preliminary experiments that were conducted in this thesis (see also [DET07a, DET07b]) demonstrate that it supports as efficient algorithms as other representations. In [GVEK96] it is stated that "isolating intervals provide worst [sic] bounds". Moreover, it is widely believed that isolating intervals do not produce good theoretical results. [DET07a, DET07b] suggest that isolating intervals should be re-evaluated.

The main tool is Sturm-Habicht and subresultant remainder sequences, ex-

tended to several variables by the technique of binary segmentation. Recent breakthroughs on univariate root isolation are exploited. These have reduced complexity by 1-3 orders of magnitude to $\widetilde{\mathcal{O}}_B(N^6)$ [DSY05, ESY06, EMT07]. Note that the complexity that is achieved by numerical methods [Pan02] is $\widetilde{\mathcal{O}}_B(N^4)$ and hence the gap between the two approaches has narrowed. Hence, new bounds are derived for the sign evaluation of bivariate polynomials over two real algebraic numbers.

1.1 Previous Work

In [KSP05], 2×2 systems are solved and the multiplicities computed under the assumption that a generic shear has been obtained, based on [SF90]. In [Wol02], 2×2 systems of bounded degree were studied, obtained as projections of the arrangement of 3D quadrics. This algorithm is a precursor of ours, see also [ET05], except that matching and multiplicity computation was simpler. In [MP05], a subdivision algorithm is proposed, exploiting the properties of the Bernstein basis, with unknown bit complexity, and arithmetic complexity based on the characteristics of the graphs of the polynomials. For other approaches based on multivariate Sturm sequences the reader may refer to e.g. [Mil92, PRS93].

Determining the topology of a real algebraic plane curve is a closely related problem. The best bound is $\widetilde{\mathcal{O}}_B(N^{14})$ [BPM06, GVEK96]. In [WS05] three projections are used; this is implemented in INSULATE, with which we make several comparisons. Work in [EKW07] is based on Sturm-Habicht sequences and solves the problem of singularities and vertical asymptotes with the Bitstream Descartes method [EKK+05]. For an alternative using Gröbner bases the reader may refer to [CFPR06]. To the best of our knowledge the only result in topology determination using isolating intervals is [AM88], where a $\widetilde{\mathcal{O}}_B(N^{30})$ bound is proved.

We establish a bound of $\widetilde{\mathcal{O}}_B(N^{12})$ using the isolating interval representation. It seems that the complexity in [GVEK96] could be improved to $\widetilde{\mathcal{O}}_B(N^{10})$ using fast multiplication algorithms, fast algorithms for computations of signed subresultant sequences and improved bounds for the bitsize of the integers appearing in computations. To put the bounds that are presented in this thesis and [DET07a, DET07b] into perspective, note that the input is $\mathcal{O}_B(N^3)$, and the bitsize of all output isolation points for univariate solving is $\widetilde{\mathcal{O}}_B(N^2)$, and this is tight.

1.2 Contributions

The main contributions of [DET07a, DET07b] and this thesis are the following: An improved complexity bound for bivariate sign evaluation (theorem 2.37) is established, which helps us derive bounds for root counting in an extension field (lemma 3.7) and for the problem of simultaneous inequalities (corollary 3.10). We study the complexity of bivariate polynomial real solving, using

three projection-based algorithms: a straightforward grid method (theorem 3.1), a specialized RUR approach (theorem 3.5), and an improvement of the latter using fast GCD (theorem 3.6). The first two algorithms also appeared in [Tsi06]. The best bound is $\widetilde{\mathcal{O}}_B(N^{12})$; within this bound, root multiplicities are computed as well. Computing the topology of a real plane algebraic curve is in $\widetilde{\mathcal{O}}_B(N^{12})$ (theorem 3.12). Moreover, a MAPLE package has been developed which allows computations with real algebraic numbers and for implementing our algorithms presented also in [DET07a, DET07b]. It is easy to use and it integrates seminumerical filtering to speed up computations when the roots are well-separated. It guarantees exactness and completeness of results; moreover, the runtimes seem very encouraging. We illustrate it by experiments against well-established C/C++ libraries FGB/RS and SYNAPS. We also examine MAPLE libraries INSULATE and TOP, which compute curve topology. Our software is robust and effective; its runtime is within a small constant factor with respect to the fastest C/C++ library.

1.3 Outline

The thesis is divided as follows. Chapter 2 presents basic tools and complexity results in univariate solving and on operations of Sturm sequences. Having formed a solid background chapter 3 presents the three projection based algorithms that are also presented in [DET07a, DET07b]. Chapter 4 presents extensive experiments that were conducted with the library that was developed in this thesis. Chapter 5 summarizes the most important results and states the goals for future work with Sturm sequences and projection based solvers. Chapter A in the appendix presents the polynomials that were used for testing. Finally, chapter B in the appendix gives an overview of the commands that are provided through the library at the moment. Sample executions of the basic commands are shown and the output is explained. For an up-to-date coverage of the commands available for the library bundled with sample execution and explanation of each command, the reader is urged to visit the homepage of the SLV (*Sturm soLVer*) library: `http://www.di.uoa.gr/~erga/soft/SLV_index.html`.

Chapter 2

Foundations

This chapter is devoted to basic tools and complexity results that will be used by the three algorithms for real solving bivariate systems in chapter 3. The heart of the algorithms relies on Sturm sequences. Therefore most of the results of this chapter deal with or are based on sequences generation and evaluation on rational points. Moreover, Fan-In / Fan-Out techniques are described in section 2.1.7 for completeness, although they are not applied in bivariate solving since at the moment seem not to yield better bounds there.

2.1 Notation and Basic Complexity Results

This section sets the necessary notation and covers basic complexity results mainly on primitive operations with univariate polynomials.

In what follows $\mathcal{O}_B$ means bit complexity and the $\widetilde{\mathcal{O}}_B$-notation means that we are ignoring polylogarithmic factors. For $f \in \mathbb{Z}[y_1, \ldots, y_k, x]$, $\deg(f)$ denotes its total degree, while $\deg_x(f)$ denotes its degree if we consider it as a univariate polynomial with respect to x. $\mathcal{L}(f)$ bounds the bitsize of the coefficients of f (including a bit for the sign). We assume $\mathcal{L}(\deg(f)) = \mathcal{O}(\mathcal{L}(f))$. For $a \in \mathbb{Q}$, $\mathcal{L}(a)$ is the maximum bitsize of numerator and denominator. Let $M(\tau)$ denote the bit complexity of multiplying two integers of bit size at most τ and $M(d, \tau)$ denote the bit complexity of multiplying two univariate polynomials of degrees $\leqslant d$ and coefficient bit size $\leqslant \tau$. Using FFT, $M(\tau) = \mathcal{O}_B(\tau \lg^{c_1} \tau)$, $M(d, \tau) = \mathcal{O}_B(d\tau \lg^{c_2}(d\tau)) = \widetilde{\mathcal{O}}_B(d\tau)$, for suitable constants c_1, c_2. Let $f, g \in \mathbb{Z}[x]$, $\deg(f) = p \geqslant q = \deg(g)$ and $\mathcal{L}(f), \mathcal{L}(g) \leqslant \tau$. $\mathrm{rem}(f, g)$ and $\mathrm{quo}(f, g)$ denote the Euclidean remainder and quotient, respectively. We compute $f_1 \bmod f_2$ in $\widetilde{\mathcal{O}}_B(d(\delta\tau_2 + \tau_1))$, where $d \geqslant \deg(f_i), \delta = \deg(f_1) - \deg(f_2), \mathcal{L}(f_i) = \tau_i$.

2.1.1 Evaluation at a point $x_0 \in \mathbb{Z}$ using Horner's rule

Given $f = \sum_{i=1}^{d} c_i x^i$ such that $\mathcal{L}(c_i) \leqslant \tau$ and $x_0 \in \mathbb{N}$ such that $\mathcal{L}(x_0) = \sigma$ we can perform the evaluation with Horner's rule by applying only d multiplications

plus $d+1$ additions, since Horner's rule implies

$$c_0 + x_0 \cdot (c_1 + x_0 \cdot (\ldots (c_{d-1} + x_0 \cdot c_d) \ldots)).$$

More formally, the evaluation $\mathtt{Eval}_{\mathtt{Horner}}(f, x_0)$ can be described with the following recursive equations:

$$\begin{aligned} \mathtt{Eval}_{\mathtt{Horner}}(f, x_0) &= h(f, d+1, x_0) \\ h(f, i+1, x_0) &= c_{d-i} + x_0 \cdot h(f, i, x_0) \\ h(f, 0, x_0) &= c_d \end{aligned}$$

The running time is asymptotically bounded by the d multiplications, hence it is $\widetilde{\mathcal{O}}_B(d \max\{d\sigma, \tau\})$. The bitsize of the result is $\mathcal{L}(f(x_0)) = O(d\sigma + \tau)$.

2.1.2 Pseudo-Division and Pseudo-Remainder

In cases where the coefficients of the polynomials do not belong to a field, then classical division has to be redefined. This redefinition is the following:

Definition 2.1. *Let* $f, g \in \mathbb{Z}[x]$. *There exist unique polynomials* $Q, R \in \mathbb{Z}[x]$ *such that*

$$\mathit{lead}(g)^{\delta-1} f = Qg + R$$

and $\deg(R) < \deg(g)$ *and* $\delta = \max\{1, \deg(f) - \deg(g)\}$. Q *and* R *are called* pseudo-quotient *and* pseudo-remainder *respectively.* Q *is denoted as* pquo *and* R *as* prem.

Note that for simplicity in the above definition $f, g \in \mathbb{Z}[x]$. However, the definition still applies for $f, g \in \mathbb{K}[x]$, for any commutative ring $\mathbb{K}$.

Lemma 2.2. *Let* $f, g \in \mathbb{Z}[x]$ *such that* $\deg(f) = m \geqslant n = \deg(g)$ *and* $\mathcal{L}(f) = \sigma$, $\mathcal{L}(g) = \tau$. *Then* $\mathcal{L}(\mathsf{prem}(f, g)) = \mathcal{L}(\mathsf{pquo}(f, g)) = \widetilde{\mathcal{O}}(\delta\tau + \sigma)$, *where* $\delta = m - n$.

Proof. Let $f = \sum_{i=0}^{m} a_i x^i$ and $g = \sum_{i=0}^{n} b_i x^i$ and consider

$$M = \begin{bmatrix} b_n & b_{n-1} & \ldots & b_0 & & & & \\ & b_n & \ldots & b_1 & b_0 & & & \\ & & \ddots & & & \ddots & & \\ & & & b_n & b_{n-1} & \ldots & b_1 & b_0 \\ a_m & a_{m-1} & \ldots & a_{m-n} & a_{m-n-1} & \ldots & a_1 & a_0 \end{bmatrix}$$

The dimension is $(m-n+2) \times (m+1)$. The coefficients of the remainder can be computed as determinants of certain sub-matrices of M [BPM06, Yap00]. By Bareiss' algorithm, this also holds for pseudo-remainders.

The coefficient of x^{n-1-j} in the remainder, where $n-1 \geqslant j \geqslant 0$, is the determinant of matrix M_j, formed by taking the first $m-n+1$ columns of M and column $m-n+2+j$.

Using Hadamard's inequality on the rows

$$|c_j| = |\det M_j| \leqslant ((\delta+2)\,2^{\tau})^{\delta+1} \cdot 2^{\sigma}(\delta+2):$$

The coefficients of $\mathrm{pquo}(f,g)$ can be computed as principal minors of

$$\begin{bmatrix} a_m & a_{m-1} & \dots & a_{m-n} & a_{m-n-1} & \dots & a_1 & a_0 \\ b_n & b_{n-1} & \dots & b_0 & & & & \\ & b_n & \dots & b_1 & b_0 & & & \\ & & \ddots & & & \ddots & & \\ & & & b_n & b_{n-1} & \dots & b_1 & b_0 \end{bmatrix}$$

□

2.1.3 Greatest Common Divisor

Let $\mathcal{K}$ be a unique factorization domain. For example $\mathcal{K} = \mathbb{Z}$ or $\mathcal{K} = \mathbb{Z}[y]$. Then we can define the *greatest common divisor* of two polynomials $f, g \in \mathcal{K}[x]$ as the polynomial of maximal degree that divides the two polynomials f and g. This is denoted as $\gcd(f,g)$.

The greatest common divisor is of extreme importance in what follows. It allows us to compute the number of common roots between two different polynomials as well as compute the number of different roots of a polynomial f.

Given $f, g \in \mathcal{K}[x]$ such that $\deg(f) = d_1, \deg(g) = d_2, \mathcal{L}(f) \leqslant \tau$ and $\mathcal{L}(g) \leqslant \tau$, we can compute the gcd in $\widetilde{\mathcal{O}}_B(d_1 d_2 \tau)$. The bitsize of its coefficients is $\mathcal{O}(\max\{d_1, d_2\}\tau)$. Due to the importance of the gcd in the computations that arise in the subsequent algorithms, more details will be discussed at the appropriate points.

2.1.4 Resultant

Another significant tool that forms the basis for all algorithms that will be discussed in chapter 3 is the *resultant* of two polynomials.

Theorem 2.3. *Given* $f, g \in \mathcal{K}[x]$ *such that* $f = \sum_{i=1}^{n} a_i x^x$ *and* $g = \sum_{j=1}^{m} b_j x^j$ *with* $a_n b_m \neq 0$, *there is a unique (up to sign) irreducible polynomial* $\mathtt{res}(f,g) \in \mathcal{K}[a_n, \dots, a_0, b_m, \dots, b_0]$ *which is zero iff* f, g *have a common factor. It is homogeneous and* $\deg(\mathtt{res}(f,g)) = \deg(f) + \deg(g) = n + m$. *This polynomial is called resultant.*

Theorem 2.4. *Given* $f, g \in \mathcal{K}[x]$ *we can compute the resultant* $\mathtt{res}(f,g)$ *via the Sylvester matrix* $Syl(f,g)$. *More specifically, we have:*

$$\mathtt{res}(f,g) = \det(Syl(f,g)).$$

Definition 2.5 (Sylvester Matrix)**.** *The Sylvester matrix of* f *and* g *is the*

$(n+m)\times(n+m)$ *matrix defined as follows:*

$$Syl(f,g)=\begin{pmatrix} a_n & a_{n-1} & a_{n-2} & \cdots & \cdots & a_0 & & & \\ & a_n & a_{n-1} & a_{n-2} & \cdots & \cdots & a_0 & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & a_n & a_{n-1} & a_{n-2} & \cdots & \cdots & a_0 \\ b_m & b_{m-1} & b_{m-2} & \cdots & b_0 & & & & \\ & b_m & b_{m-1} & b_{m-2} & \cdots & b_0 & & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & & b_m & b_{m-1} & b_{m-2} & \cdots & b_0 \end{pmatrix}$$

Note that we can compute the resultant in time $\widetilde{\mathcal{O}}_B(nm\tau)$, where τ is a bound on the bitsize of the coefficients of f and g.

2.1.5 Discriminant

Definition 2.6. *Let* $f=\sum_{i=0}^{d}a_ix^i\in\mathcal{K}[x], d\geqslant 2$ *and* $\rho_1,\ldots,\rho_d\in\mathcal{K}$ *its roots. The* discriminant *of* f *is the quantity*

$$\mathsf{disc}(f)=a_n^{2d-2}\prod_{1\leqslant i<j\leqslant d}(\rho_i-\rho_j)^2$$

In order to compute the discriminant we need the following theorem:

Theorem 2.7. *Given* $f=\sum_{i=0}^{d}a_ix^i\in\mathcal{K}[x], d\geqslant 2$ *we have:*

$$\mathtt{res}(f,f')=\pm a_n\mathsf{disc}(f)=(-1)^{\frac{n(n-1)}{2}}a_n\mathsf{disc}(f)$$

A useful remark that will be used later on in section 3.1.1 is the following:

Remark 2.8 (Multiplicities)**.** *The definition of the discriminant does not require only different roots of the polynomial. Therefore, if a polynomial has a multiple root, then the discriminant must be zero (0). Equivalently,* $\mathtt{res}(f,f')=0$ *if and only if* f *has multiple roots. If* $f\in\mathbb{Z}[x]$ *does not contain multiple roots, then* $\mathsf{disc}(f)\geqslant 1$.

An upper bound on the discriminant can be obtained with the following theorem:

Theorem 2.9. *Let* $f=\sum_{i=0}^{d}a_ix^i\in\mathbb{Z}[x]$. *Then:*

$$|\mathsf{disc}(f)|\leqslant d^d\mathcal{M}(f)^{2(d-1)}\leqslant d^d\|f\|_2^{2(d-1)},$$

where $\|f\|_2=\sqrt{\sum_{i=0}^{d}a_i^2}$ *and* $\mathcal{M}(f)$ *is the Mahler bound of* f *which will be described in the following paragraph.*

2.1.6 Mahler Bound

Definition 2.10. *Let* $A = \sum_{i=0}^{d} a_i x^i \in \mathbb{C}[x] \setminus \mathbb{C}$ *such that*

$$A = \sum_{i=0}^{d} a_i x^i = a_d \prod_{i=1}^{d} (X - \rho_i)$$

with $a_d \neq 0$. *Then, the* Mahler bound *of* A, *denoted as* $\mathcal{M}(A)$ *is*

$$\mathcal{M}(A) = |a_d| \prod_{j=1}^{d} \max\{1, |\rho_j|\}$$

The following identities are presented without proof:

Proposition 2.11. *Given* $A, B \in \mathbb{C}[x] \setminus \mathbb{C}$, *such that Mahler bound is defined and* $\deg(A) = d$ *and* $n \in \mathbb{N}^*$, *the following hold:*

A. $\mathcal{M}(x^d A(\frac{1}{x})) = \mathcal{M}(A(x))$.

B. $\mathcal{M}(A \cdot B) = \mathcal{M}(A) \cdot \mathcal{M}(B)$.

C. $\mathcal{M}(A(x^k)) = \mathcal{M}(A(x))$.

2.1.7 Classical Fan-In / Fan-Out

Consider the setting of the previous paragraph but this time assume we want to evaluate f over a set $\mathfrak{J}$ of natural numbers such that $|\mathfrak{J}| \leqslant d$. The obvious technique would be to apply $|\mathfrak{J}|$ times Horner's rule, thereby evaluating f over all $|\mathfrak{J}|$ numbers in $\widetilde{\mathcal{O}}_B(|\mathfrak{J}| d \max\{d\sigma, \tau\}) = \widetilde{\mathcal{O}}_B(\max\{d^3\sigma, d^2\tau\})$. However, we can do better. This is a classic result [EP99] called Fan-In / Fan-Out and is based on an extension of the following lemma.

Lemma 2.12. *Given* $a, b, c \in \mathbb{N}$, $(a \mod (bc)) \mod b = a \mod b$.

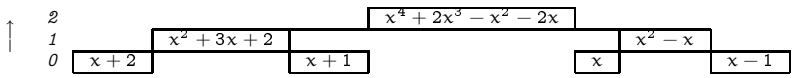
Proof. $a \mod (bc) = k \rightarrow a = jbc + k$. $a \mod b = m \rightarrow a = ib + m$. Therefore, $k = ib + m - jbc \rightarrow k \mod b = m$. □

Corollary 2.13. *The following holds:*

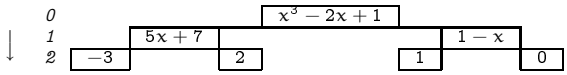
$$p(x) \mod (x - x_i) = \left[p(x) \mod \prod_{j \in \mathfrak{J}} (x - x_j) \right] \mod (x - x_i), \quad i \in \mathfrak{J} \subsetneq \mathbb{N}.$$

Example 2.1. *Let* $f(x) = x^3 - 2x + 1$ *and assume we want to evaluate the function on* $\{-2, -1, 0, 1\}$. *The evaluation consists of two parts.*

Fan-In: *In the first part we generate products of the form* $\prod_j (x - x_j)$. *Thus we start with the polynomials that have as roots the required integers; in this case we have* $(x + 2)$, $(x + 1)$, x, $(x - 1)$. *In the following step we take two polynomials each time and generate their product, thereby obtaining this time* $(x^2 + 3x + 2)$ *and* $(x^2 - x)$. *Finally, we compute the product* $p = (x^2 + 3x + 2)(x^2 - x)$ *of these two polynomials that occurred. Working this way we generated a binary tree bottom-up as shown below:*



Fan-Out: *The next step consists of computing the required remainders. This time we traverse the tree top-down and store the results of each level. Therefore on the first level we must compute the pseudo-remainder* $\mathrm{prem}(f,p)$. *Note that* f *remains intact since* $\deg(p) = 4 > 3 = \deg(f)$. *On each subsequent level we work similarly; i.e. we compute the pseudo-remainder between the polynomial of the previous level and the polynomial that is directed by the fan-in tree at the respective position. Hence on level* 2 *we compute* $\mathrm{prem}\left(x^3-2x+1, x^2+3x+2\right) = 5x+7$ *and* $\mathrm{prem}\left(x^3-2x+1, x^2-x\right) = -x+1$. *Finally, on the third level we compute the pseudo-remainders* $\mathrm{prem}(5x+7, x+2) = -3$, $\mathrm{prem}(5x+7, x+1) = 2$, $\mathrm{prem}(-x+1, x) = 1$, *and* $\mathrm{prem}(-x+1, x-1) = 0$. *The whole process and the new binary tree that has been formed is shown below:*



Note that the requested values have been computed; i.e. $f(-2) = -3, f(-1) = 2, f(0) = 1,$ *and* $f(1) = 0$.

Theorem 2.14 (Fan-In / Fan-Out)**.** *Let* $f \in \mathbb{Z}[x]$ *such that* $\deg(f) \leqslant d$ *and* $\mathcal{L}(f) \leqslant \tau$. *We can evaluate* f *at* $n \leqslant 2^m \leqslant d+1$ *natural numbers of bitsize* σ *in time* $\widetilde{\mathcal{O}}_B(d^2\sigma + d\tau)$.

Proof. For simplicity assume that d is of the form $2^m - 1$ and $n = d+1$.

Fan-In: Generating $\prod_j(x-x_j)$. We form a binary tree with leaves the polynomials $(x-x_j)$, nodes the various polynomials that arise as products from the nodes of the previous level, and root the polynomial $\prod_{0\leqslant i\leqslant d}(x-x_j)$. If we start counting from 0 at the leaf-level, then at level j we have $\frac{n}{2^j}$ polynomials of degree 2^j. Moreover, the bitsize of these polynomials is $\widetilde{\mathcal{O}}((j+1)\sigma)$. The cost for computing the $\frac{n}{2^j}$ polynomials of degree 2^j, where $j = 1, 2, \ldots, \lg n$, is $\widetilde{\mathcal{O}}_B((n/2^j)M(2^{j-1}, j\sigma)) = \widetilde{\mathcal{O}}_B(nj\sigma)$, since $M(a,b) = \widetilde{\mathcal{O}}_B(ab)$ denotes the cost of multiplying two polynomials of degree bounded by a and coefficient bitsize bounded by b. Overall, the complexity for this step is $\mathcal{O}_B(n\sigma\lg^2 n) = \widetilde{\mathcal{O}}_B(n\sigma) = \widetilde{\mathcal{O}}_B(d\sigma)$.

Fan-Out: Subdivision. We now traverse the tree in the inverse direction. Moving from level j to $j+1$ $(j = 0, \ldots, (\lg n - 1))$ we have to perform 2^{j+1} divisions with remainder with polynomials of degree $\frac{n}{2^j}$ and respective bitsize $\beta_j = \mathcal{O}((\lg n + 1 - j)\sigma)$. Let's have a look on the degrees of the resulting polynomials (remainders). At root level the given polynomial f does not change. Dividing this polynomial with polynomials of degree $\frac{n}{2}$ we obtain polynomials of degree $\frac{n}{2}-1$. Similarly, at level j (again counting from 0 at root level) we have polynomials of degree $\frac{n}{2^j}-1$, $j = 1, \ldots, \lg n$. Now, let's examine their respective bitsizes. Moving from level 0 to level 1 we perform division on polynomials with bitsize $b_0 = \tau$ (bound for coefficients of f) and $\beta_1 = \mathcal{O}(\sigma\lg n)$. Thus, the result has bitsize $b_1 = \mathcal{O}(\delta_1\beta_1 + b_0) = \mathcal{O}(\delta_1\sigma\lg n + \tau)$, where $\delta_1 = (n-1) - \frac{n}{2} = \frac{n}{2} - 1$. Similarly, when reaching level j we have performed division with polynomials of bitsize b_{j-1} and β_j which results in $b_j = \mathcal{O}(\delta_j\beta_j + b_{j-1})$. Note that β_j is defined as $\beta_j = \mathcal{O}((\lg n + 1 - j)\sigma),\ j = 0, \ldots, \lg n$. Also, $\delta_j = \frac{n}{2^j} - 1,\ j = 1, \ldots, \lg n$.

Hence for all $j = 0, \ldots, \lg n$, $b_j \leqslant \mathcal{O}(n\sigma \lg n + \tau) = \widetilde{\mathcal{O}}(n\sigma + \tau)$. As a consequence, the running time complexity for this phase is:

$$\begin{aligned} T &= \sum_{j=0}^{\lg n - 1} 2^{j+1} \mathcal{O}_B((\frac{n}{2^j} - 1)(\frac{n}{2^j})(n\sigma \lg n + \tau)) \\ &= \mathcal{O}_B((n\sigma \lg n + \tau) \sum_{j=0}^{\lg n - 1} \frac{n}{2^j}) \\ &= \mathcal{O}_B((n\sigma \lg n + \tau) 2(n-1)) \\ &= \widetilde{\mathcal{O}}_B(n^2\sigma + n\tau) \end{aligned}$$

since we perform 2^{j+1} divisions per level j and the degrees are $(\frac{n}{2^j} - 1)$ and $\frac{n}{2^j}$ and the bitsizes are bounded by $\mathcal{O}(n\sigma \lg n + \tau)$. □

2.2 Representing Real Algebraic Numbers

We choose to represent a real algebraic number $\alpha \in \mathbb{R}_{alg}$ by the *isolating interval* representation. because it is more intuitive, it facilitates geometric applications, and turns out to be as efficient as other representations. It includes a square-free polynomial which vanishes on α and a (rational) interval containing α and *no other root*. More particularly, assume that α is the unique root of the square-free polynomial f in the interval $\mathfrak{I} = [I_L, I_R]$, where $I_L, I_R \in \mathbb{Q}$. We denote this with the following representation:

$$\alpha \simeq [f, \mathfrak{I}] = [f, [I_L, I_R]].$$

Remark 2.15 (Bolzano)**.** *Note that since f is square-free and $\mathfrak{I}$ is an isolating interval of the unique root α of f in the interval, then f satisfies the Bolzano criterion in the interval, i.e. $f(I_L) \cdot f(I_R) < 0$.*

Remark 2.16 (Derivative)**.** *Moreover, it is easy to compute the* sign *of the derivative of f when evaluating at $x = \alpha$. Since f is square-free, there exists an interval $\mathfrak{J} \subseteq \mathfrak{I}$ containing α such that f is monotone. Moreover, since the square-free f satisfies the Bolzano criterion in $\mathfrak{I}$ it does not change sign on the intervals $[I_L, \alpha)$ and $(\alpha, I_R]$ by construction. Therefore,*

$$sign\big(f'(\alpha)\big) = sign\Big(sign\big(f(I_R)\big) - sign\big(f(I_L)\big)\Big).$$

2.3 Polynomial Remainder Sequences

We now return to the problem of computing the gcd of two polynomials. In applications the interest relies on the roots (or the factorization) of the gcd. Hence, it is acceptable to compute the gcd up to similarity. An easy way of computing the gcd of two polynomials $f, g \in \mathbb{Z}[x]$ would be to perform successive pseudo-divisions (see section 2.1.2), thereby producing the sequence of (remainder) polynomials

$$R_0 = f, R_1 = g, R_2 = \mathsf{prem}(f, g), \ldots, R_k = \mathsf{prem}(R_{k-2}, R_{k-1}),$$

such that $\mathrm{prem}(R_{k-1}, R_k) = 0$. Therefore, any sequence that *mimics* the above remainder sequence is called *Polynomial Remainder Sequence* (*PRS* for short). Moreover, if the degree at each step drops by 1, we call the sequence *regular* (or *normal*). The variations of the possible sequences rely on different computations of the parameters k, λ of the following identity

$$\lambda_i f = Qg + kR.$$

Hence, apart from the Pseudo-Euclidean PRS which is rarely, if ever, used in practice, other PRSs are the Primitive-PRS which maintains primitive polynomials at each step, as well as the most commonly used in practice Subresultant PRS and Sturm-Habicht PRS. The Subresultant and the Sturm-Habicht sequences avoid the costly computation of the gcd at each step (contrary to the Primitive-PRS) and the bitsize of the coefficients of the polynomials that arise in the sequence increases *only* linearly contrary to the Pseudo-Euclidean PRS where the bitsize of the coefficients exhibits exponential growth. The following example illustrates this fact:

Example 2.2. *Given*

$$\begin{aligned} f &= x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5 \\ g &= 3x^6 + 5x^4 - 4x^2 - 9x + 21 \end{aligned}$$

we obtain the following Remainder Sequences:
Euclidean:

$$-15x^4 + 3x^2 - 9$$

$$15795x^2 + 30375x - 59535$$

$$1254542875143750x - 1654608338437500$$

$$12593338795500743100931141992187500$$

Primitive Part:

$$-5x^4 + x^2 - 3$$

$$13x^2 + 25x - 49$$

$$4663x - 6150$$

$$1$$

Subresultant:

$$15x^4 - 3x^2 + 9$$

$$65x^2 + 125x - 245$$

$$9326x - 12300$$

$$260708$$

The reader may refer to [Yap00] for a detailed treatment of the Polynomial Remainder Sequences. We now switch to the class of Polynomial Remainder Sequences that are actually used in real solving and are the heart of all computations that are described in this thesis.

2.3.1 Signed Polynomial Remainder Sequences

Definition 2.17. *[LR01] The* signed polynomial remainder sequence *of* f *and* g *is denoted by* $\mathrm{sPRS}(f,g)$ *and is a polynomial sequence* similar *to the*

$$R_0 = f, R_1 = g, R_2 = -\operatorname{prem}(f,g), \ldots, R_k = -\operatorname{prem}(R_{k-2}, R_{k-1}),$$

where $\operatorname{prem}(R_{k-1}, R_k) = 0$. *The* quotient sequence *contains the* $\{Q_i\}_{0\leqslant i\leqslant k}$, *where* $Q_i = \operatorname{pquo}(R_i, R_{i+1})$, *and the* quotient boot *is* $(Q_0, \ldots, Q_{k-1}, R_k)$.

There is a huge bibliography on signed polynomial remainder sequences (c.f [BPM06, vzGG03, Yap00] and references therein). Here, we consider signed subresultant sequences, which contain polynomials similar to the polynomials in the signed polynomial remainder sequence; see [vzGL03] for a unified approach to subresultants. We consider the signed Subresultant sequences and Sturm-Habicht sequences ($\mathrm{SR}(f,g)$ and $\mathrm{StHa}(f,g)$ respectively) of f, g, which contain polynomials proportional to the polynomials in $\mathrm{sPRS}(f,g)$ since they achieve better bounds on the coefficient bitsize as it was demonstrated in example 2.2 and have good specialization properties since they are defined through determinants. To be more specific, we will consider subresultant sequences, or Sturm-Habicht sequences, in the case where g is the derivative of f. The reader may refer to [GVLRR89, BPM06] for information. We recall here the main results regarding the computation and the evaluation of such sequences. By $\mathrm{sr}(f,g)$ the sequence of the principal subresultant coefficients, by $\mathrm{SQ}(f,g)$ the corresponding quotient boot, and by $\mathrm{SR}(f,g;a)$ the evaluated sequence over $a \in \mathbb{Q}$. If the polynomials are multivariate, then the aforementioned sequences are considered w.r.t. variable x, except if explicitly stated otherwise.

Proposition 2.18. *[LR01, LRSED00, Rei97] Assuming* $p \geqslant q$, $\mathrm{SR}(f,g)$ *is computed in* $\widetilde{\mathcal{O}}_B(p^2q\tau)$ *and* $\mathcal{L}(\mathrm{SR}_j(f,g)) = \mathcal{O}(p\tau)$. *For any* f, g, *their quotient boot, any polynomial in* $\mathrm{SR}(f,g)$, *their resultant, and their* gcd *are computed in* $\widetilde{\mathcal{O}}_B(pq\tau)$.

Lemma 2.19. *[LR01, Rei97] Let* $p \geqslant q$. *We can compute* $\mathrm{SR}(f,g;a)$, *where* $a \in \mathbb{Q}\cup\{\pm\infty\}$ *and* $\mathcal{L}(a) = \sigma$, *in* $\widetilde{\mathcal{O}}_B(pq\tau + q^2\sigma + p^2\sigma)$. *If* $f(a)$ *is known, then the bound becomes* $\widetilde{\mathcal{O}}_B(pq\tau + q^2\sigma)$.

Proof. Let $\mathrm{SR}_{q+1} = f$ and $\mathrm{SR}_q = g$. For the moment we forget SR_{q+1}. We may assume that SR_{q-1} is computed, since the cost of computing one element of SR is the same as that of computing $\mathrm{SQ}(f,g)$ (proposition 2.18).

We follow Lickteig and Roy [LR01]. For two polynomials A, B of degree bounded by D and bit size bounded by L, we can compute $\mathrm{SR}(A,B)(a)$, where $\mathcal{L}(a) \leqslant L$, in $\widetilde{\mathcal{O}}_B(M(D,L))$. In our case $D = \mathcal{O}(q)$ and $L = \mathcal{O}(p\tau + q\sigma)$, thus the total costs is $\widetilde{\mathcal{O}}_B(pq\tau + q^2\sigma)$.

It remains to compute the evaluation $\mathrm{SR}_{q+1}(a) = f(a)$. This can be done using Horner's scheme in $\widetilde{\mathcal{O}}_B(p\max\{\tau, p\sigma\})$. Thus, the whole procedure has complexity

$$\widetilde{\mathcal{O}}_B(pq\tau + q^2\sigma + p\max\{\tau, p\sigma\}),$$

where the term $p\tau$ is dominated by $pq\tau$. □

If $\mathcal{L}(f) = \tau_f \neq \tau_g = \mathcal{L}(g)$ then $\tau = \max\{\tau_f, \tau_g\}$ in the previous theorem.

When $q > p$, $\mathrm{SR}(f, g)$ is $f, g, -f, -(g \bmod (-f)) \ldots$, thus $\mathrm{SR}(f, g; a)$ starts with a sign variation irrespective of $\mathrm{sign}(g(a))$. If only the sign variations are needed, there is no need to evaluate g, so prop. 2.19 yields $\widetilde{\mathcal{O}}_B(pq\tau + p^2\sigma)$.

Definition 2.20. *Let* L *denote a list of real numbers.* $\mathtt{VAR}(L)$ *denotes the number of (possibly modified, see e.g. [BPM06, GVLRR89]) sign variations.*

Corollary 2.21. *For any* f, g, $\mathtt{VAR}(\mathrm{SR}(f, g; a))$ *is computed in* $\widetilde{\mathcal{O}}_B(pq\tau + \min\{p, q\}^2\sigma)$*, provided* $\mathrm{sign}(f(a))$ *is known.*

2.4 Univariate Polynomials

The heart of the algorithms that are presented in chapter 3 relies on real solving univariate polynomials. Starting with some bounds, Sturm's algorithm will be briefly presented in the sequel, as well as the algorithm for determining the sign of a polynomial when evaluated at a real algebraic number. Finally, some bounds on root separation are going to be presented. These are critical on the complexity analysis of all the subsequent algorithms .

2.4.1 Bounding Roots

Let $f = \sum_{i=0}^{d} a_i x^i \in \mathbb{Z}[x]$. Then we can obtain a bound for the roots of f with the following lemmas.

Maximal Bounds

Maximal bounds for roots are presented first.

Lemma 2.22. *[Cauchy, Mignotte] Let* α *be a root of* f*. Then*

$$|\alpha| \leqslant 1 + \max\left(\left|\frac{a_{d-1}}{a_d}\right|, \ldots, \left|\frac{a_0}{a_d}\right|\right).$$

Note that the bound in lemma 2.22 is invariant if we multiply the polynomial by a constant. However, it behaves badly under the transformation $x \mapsto \frac{x}{2}$, which only changes the roots by a factor of 2, but may change the bound by a factor of 2^d. The following bounds do not have this effect.

Lemma 2.23. *[Cauchy] Let* α *be a root of* f*. Then*

$$|\alpha| \leqslant \max\left(\left|\frac{da_{d-1}}{a_d}\right|, \left|\frac{da_{d-2}}{a_d}\right|^{\frac{1}{2}}, \ldots, \left|\frac{da_0}{a_d}\right|^{\frac{1}{d}}\right).$$

Lemma 2.24. *[Zassenhaus] Let* α *be a root of* f*. Then*

$$|\alpha| \leqslant 2\max\left(\left|\frac{a_{d-1}}{a_d}\right|, \left|\frac{a_{d-2}}{a_d}\right|^{\frac{1}{2}}, \ldots, \left|\frac{a_0}{a_d}\right|^{\frac{1}{d}}\right).$$

Complexity: All the above bounds can be computed in time $\widetilde{\mathcal{O}}_B(d\tau)$.

Minimal Bounds

The above lemmas 2.22, 2.23, 2.24 can be used for obtaining a bound on the minimal absolute value of a root of a polynomial (assuming that the constant coefficient is not zero - which is easy to check). The idea and algorithm for computing a bound comes from the following remark:

Remark 2.25. *Let* $f = \sum_{i=0}^{d} a_i x^i \in \mathbb{Z}[x]$ *and assume that* $\alpha \neq 0$ *is a root of* f. *Then it holds that*

$$\sum_{i=0}^{d} a_i \alpha^i = 0. \tag{2.1}$$

Consider the reciprocal polynomial $g(x) = x^d f(\frac{1}{x}) = \sum_{i=0}^{d} a_i x^{d-i}$. *Then by equation (2.1) and the hypothesis it follows that* $\frac{1}{\alpha}$ *is a root of* g. *Hence, obtaining an upper bound on the roots of* g *we actually obtain a lower bound on the roots of* f.

2.4.2 Root Isolation and Sturm's algorithm

Algorithm 1 presents Sturm's algorithm for computing isolating intervals of the roots of a polynomial $f \in \mathbb{Z}[x]$. For simplicity, the input polynomial f is square-free.

Proposition 2.26. *[DSY05, ESY06, EMT07] Let* $f \in \mathbb{Z}[x]$ *have degree* p *and bitsize* τ_f. *We compute the isolating interval representation of its real roots and their multiplicities in* $\widetilde{\mathcal{O}}_B(p^6 + p^4\tau_f^2)$. *The endpoints of the isolating intervals have bitsize* $\mathcal{O}(p^2 + p\,\tau_f)$ *and* $\mathcal{L}(f_{red}) = \mathcal{O}(p + \tau_f)$.

There is no need to evaluate f_{red} over the interval's endpoints because its sign is known; moreover, $f_{red}(a)f_{red}(b) < 0$.

2.4.3 Univariate sign determination

Corollary 2.27. *[BPM06, EMT07] Given a real algebraic number* $\alpha \cong (f, [a, b])$, *where* $\mathcal{L}(a) = \mathcal{L}(b) = \mathcal{O}(p\tau_f)$, *and* $g \in \mathbb{Z}[x]$, *such that* $\deg(g) = q, \mathcal{L}(g) = \tau_g$, *we compute* $\operatorname{sign}(g(\alpha))$ *in bit complexity* $\widetilde{\mathcal{O}}_B(pq\max\{\tau_f, \tau_g\} + p\min\{p, q\}^2\tau_f)$.

Proof. Assume that α is not a common root of f and g in $[a, b]$, then it is known that

$$\operatorname{sign} g(\alpha) = [\mathtt{VAR}(\mathrm{SR}(f, g; a)) - \mathtt{VAR}(\mathrm{SR}(f, g; b))]\operatorname{sign}(f'(\alpha)).$$

Actually the previous relation holds in a more general context, when f *dominates* g, see [Yap00] for details. Notice that $\operatorname{sign}(f'(\alpha)) = \operatorname{sign}(f(b)) - \operatorname{sign}(f(b))$, which is known from the real root isolation process. The complexity of the operation is dominated by the computation of $\mathtt{VAR}(\mathrm{SR}(f, g; a))$ and $\mathtt{VAR}(\mathrm{SR}(f, g; b))$, i.e. we compute SQ and evaluate it on a and b.

Algorithm 1: STURM::UNIVARIATE.

Input: A square-free polynomial $f \in \mathbb{Z}[x]$

Output: A list of isolating intervals of the roots of f.

```
1  S ⟵ SR(f, f′);
2  N ⟵ VAR(S; −∞) − VAR(S; +∞);
3  if N = 0 then return ∅;
4  if N = 1 then return [−∞, +∞];
5  M ⟵ Maximal_Bound(f);
6  Intervals ⟵ ∅;
7  Stack ⟵ {[−M, M, VAR(S, −∞), VAR(S, +∞)]};
8  while Stack ≠ ∅ do
9      [a, b, V_a, V_b] ⟵ Pop(Stack);
10     c = (a+b)/2;
11     if f(c) = 0 then
12         Intervals ⟵ Intervals ∪ {[c, c]};
13         M ⟵ Minimal_Bound(Subst(x = x − c, f)/x);
14         V_{c+} ⟵ VAR(S; c + M);
15         V_{c−} ⟵ V_{c+} − 1;
16         if V_a = V_{c−} + 1 then Intervals ⟵ Intervals ∪ {[a, c − M]};
17         if V_a > V_{c−} + 1 then Push(Stack, {[a, c − M, V_a, V_{c−}]});
18         if V_b = V_{c+} − 1 then Intervals ⟵ Intervals ∪ {[c + M, b]};
19         if V_b < V_{c+} − 1 then Push(Stack, {[c + M, b, V_{c+}, V_b]});
20     else
21         V_c ⟵ VAR(S; c);
22         if V_a = V_c + 1 then Intervals ⟵ Intervals ∪ {[a, c]};
23         if V_a > V_c + 1 then Push(Stack, {[a, c, V_a, V_c]});
24         if V_b = V_c − 1 then Intervals ⟵ Intervals ∪ {[c, b]};
25         if V_b < V_c − 1 then Push(Stack, {[c, b, V_c, V_b]});
26 return Intervals ;
```

As explained above, there is no need to evaluate the polynomial of the biggest degree, i.e. the first (and the second if $p < q$) of $\mathrm{SR}(f, g)$ over a and b. Thus the complexity is that of corollary 2.21, viz.

$$\widetilde{\mathcal{O}}_B(pq \max\{\tau_f, \tau_g\} + \min\{p, q\}^2 p\, \tau_f)$$

Thus the complexity of the operation is two times the complexity of the evaluation of the sequence over the endpoints of the isolating interval.

If α is a common root of f and g, or if f and g are not relative prime, then their gcd, which is the last non-zero polynomial in $\mathrm{SR}(f, g)$ is not a constant. Hence, we evaluate SR on a and b, we check if the last polynomial is not a constant and if it changes sign on a and b. If this is the case, then $\mathrm{sign}(g(\alpha)) = 0$. Otherwise we proceed as above. □

Algorithm 2: UNIVARIATE-SIGN AT.

Input: A Polynomial $g \in \mathbb{Z}[x]$ and $\alpha \cong (f, [a, b]) \in \mathbb{R}_{alg}$.

Output: The sign of $g(\alpha)$.

```
1 PRS ⟵ SR(f, g);
2 ν_L ⟵ VAR(SR(PRS; a));
3 ν_R ⟵ VAR(SR(PRS; b));
4 return Sign([ν_L − ν_R] · [f′(α)]);
```

2.4.4 Bounding root separation

Theorem 2.28 (Davenport-Mahler-Mignotte). [TE06] *Let* $A \in \mathbb{Z}[X]$, *with* $\deg(A) = d$ *and* $\mathcal{L}(A) = \tau$, *where* $A(0) \neq 0$. *Let* Ω *be any set of* k *pairs of indices* (i, j) *such that* $1 \leqslant i < j \leqslant d$ *and let the non-zero (complex) roots of* A *be* $0 < |\gamma_1| \leqslant |\gamma_2| \leqslant \cdots \leqslant |\gamma_d|$. *Then*

$$2^k \mathcal{M}(A)^k \geqslant \prod_{(i,j)\in\Omega} |\gamma_i - \gamma_j| \geqslant 2^{k - \frac{d(d-1)}{2}} \mathcal{M}(A)^{1-d-k} \sqrt{\mathsf{disc}(A)}.$$

Proposition 2.26 expresses the state-of-the-art in univariate root isolation. It relies on the previous propositions for fast computation of polynomial sequences, and extensions of the Davenport-Mahler bound on aggregate root separation. The following lemma, derived from Davenport-Mahler's bound, is crucial.

Lemma 2.29 (Aggregate separation). *Given* $f \in \mathbb{Z}[x]$, *the sum of the bitsize of* all *isolating points of the real roots of* f *is* $\mathcal{O}(p^2 + p\,\tau_f)$.

Proof. Let there be $r \leqslant p$ real roots. The isolating point between two consecutive real roots α_j, α_{j+1} is of magnitude at most $\frac{1}{2}|\alpha_j - \alpha_{j+1}| := \frac{1}{2}\Delta_j$. Thus their product is $\frac{1}{2^r}\prod_j \Delta_j$. Using the Davenport-Mahler-Mignotte bound by theorem 2.28, $\prod_j \Delta_j \geqslant 2^{-\mathcal{O}(p^2+p\tau_f)}$ and we take logarithms. □

Corollary 2.30 (Intermediate Points). *Given the list of real roots of* f *in isolating interval representation, we compute rational numbers between them in* $\widetilde{\mathcal{O}}_B(p^2 + p\tau_f)$.

2.5 Multivariate polynomials

We discuss multivariate polynomials, using binary segmentation [Rei97]. An alternative approach could be [Klo95]. Let $f, g \in (\mathbb{Z}[y_1, \ldots, y_k])[x]$ with $\deg_x(f) = p \geqslant q = \deg_x(g)$, $\deg_{y_i}(f) \leqslant d_i$ and $\deg_{y_i}(g) \leqslant d_i$. Let $d = \prod_{i=1}^k d_i$ and $\mathcal{L}(f), \mathcal{L}(g) \leqslant \tau$. The y_i-degree of every polynomial in $SR(f, g)$, is bounded by $\deg_{y_i}(\mathtt{res}(f, g)) \leqslant (p + q)d_i$. Thus, the homomorphism $\psi : \mathbb{Z}[y_1, \ldots, y_k] \to \mathbb{Z}[y]$, where

$$y_1 \mapsto y,\ y_2 \mapsto y^{(p+q)d_1}, \ldots, y_k \mapsto y^{(p+q)^{k-1}d_1\cdots d_{k-1}},$$

allows us to decode $\mathtt{res}(\psi(f), \psi(g)) = \psi(\mathtt{res}(f, g))$ and obtain $\mathtt{res}(f, g)$. The same holds for every polynomial in $SR(f, g)$. Now $\psi(f), \psi(g) \in (\mathbb{Z}[y])[x]$ have

$y-$degree $d = (p+q)^{k-1} d_1 \cdots d_k$ since, in the worst case, f or g hold a monomial such as $y_1^{d_1} y_2^{d_2} \ldots y_k^{d_k}$.

Thus, $\deg_y(\mathtt{res}(\psi(f), \psi(g))) < (p+q)^k d$.

Proposition 2.31. *[Rei97] We can compute* $\mathrm{SQ}(f, g)$*, any polynomial in* $\mathrm{SR}(f, g)$*, and* $\mathtt{res}(f, g)$ *in* $\widetilde{\mathcal{O}}_B(q(p+q)^{k+1} d\tau)$.

Lemma 2.32. $\mathrm{SR}(f, g)$ *is computed in* $\widetilde{\mathcal{O}}_B(q(p+q)^{k+2} d\tau)$.

Proof. Every polynomial in $\mathrm{SR}(f, g)$ has coefficients of magnitude bounded by $2^{c(p+q)\tau}$, for a suitable constant c, assuming $\tau > \lg(d)$. Consider the map $\chi : \mathbb{Z}[y] \mapsto \mathbb{Z}$, such that $y \mapsto 2^{\lceil c(p+q)\tau \rceil}$, and let $\phi = \psi \circ \chi : \mathbb{Z}[y_1, y_2 \ldots, y_k] \to \mathbb{Z}$. Then $\mathcal{L}(\phi(f)), \mathcal{L}(\phi(g)) \leqslant c(p+q)^k d\tau$. Now apply proposition 2.18. □

Theorem 2.33. *We can evaluate* $\mathrm{SR}(f, g)$ *at* $x = \alpha$*, where* $a \in \mathbb{Q} \cup \{\infty\}$ *and* $\mathcal{L}(a) = \sigma$*, in* $\widetilde{\mathcal{O}}_B(q(p+q)^{k+1} d \max\{\tau, \sigma\})$.

Proof. By proposition 2.31 $\mathrm{SQ}(f, g)$ can be computed and then evaluated it over a using binary segmentation in $\widetilde{\mathcal{O}}_B(q(p+q)^{k+1} d\tau)$. For this we need to bound the bitsize of the resulting polynomials.

The polynomials in $\mathrm{SR}(f, g)$ have total degree in $y_1, \ldots, y_k$ bounded by $(p+q)\sum_{i=1}^{k} d_i$ and coefficient bitsize bounded by $(p+q)\tau$. With respect to x, the polynomials in $\mathrm{SR}(f, g)$ have degrees in $\mathcal{O}(p)$, so substitution $x = a$ yields values of size $\widetilde{\mathcal{O}}(p\sigma)$. After the evaluation we obtain polynomials in $\mathbb{Z}[y_1, \ldots, y_k]$ with coefficient bitsize bounded by $\max\{(p+q)\tau, p\sigma\} \leqslant (p+q)\max\{\tau, \sigma\}$.

Consider $\chi : \mathbb{Z}[y] \to \mathbb{Z}$, such that $y \mapsto 2^{\lceil c(p+q)\max\{\tau,\sigma\} \rceil}$, for a suitable constant c. Apply the map $\phi = \psi \circ \chi$ to f, g. Now, $\mathcal{L}(\phi(f)), \mathcal{L}(\phi(g)) \leqslant cd(p+q)^k \max\{\tau, \sigma\}$. By proposition 2.19, the evaluation costs $\widetilde{\mathcal{O}}_B(q(p+q)^{k+1} d \max\{\tau, \sigma\})$. □

We obtain the following corollaries for $f, g \in (\mathbb{Z}[y])[x]$, such that $\deg_x(f) = p$, $q = \deg_x(g)$, $\deg_y(f), \deg_y(g) \leqslant d$ and $\mathcal{L}(f), \mathcal{L}(g) \leqslant \tau$.

Corollary 2.34. *We compute* $\mathrm{SR}(f, g)$ *in* $\widetilde{\mathcal{O}}_B(pq(p+q)^2 d\tau)$*. For any polynomial* $\mathrm{SR}_j(f, g)$ *in* $\mathrm{SR}(f, g)$*,* $\deg_x(\mathrm{SR}_j(f, g)) = \mathcal{O}(\max\{p, q\})$*,* $\deg_y(\mathrm{SR}_j(f, g)) = \mathcal{O}(\max\{p, q\}d)$*, and also* $\mathcal{L}(\mathrm{SR}_j(f, g)) = \mathcal{O}(\max\{p, q\}\tau)$.

Corollary 2.35. *We compute* $\mathrm{SQ}(f, g)$*, any polynomial in* $\mathrm{SR}(f, g)$*, and* $\mathtt{res}(f, g)$ *in* $\widetilde{\mathcal{O}}_B(pq \max\{p, q\} d\tau)$.

Corollary 2.36. *We compute* $\mathrm{SR}(f, g\,;\, a)$*, where* $a \in \mathbb{Q} \cup \{\infty\}$ *and* $\mathcal{L}(a) = \sigma$*, in* $\widetilde{\mathcal{O}}_B(pq \max\{p, q\} d \max\{\tau, \sigma\})$*. For the polynomials* $\mathrm{SR}_j(f, g\,;\, a) \in \mathbb{Z}[y]$*, except for* f, g*, we have* $\deg_y(\mathrm{SR}_j(f, g\,;\, a)) = \mathcal{O}((p+q)d)$ *and* $\mathcal{L}(\mathrm{SR}_j(f, g\,;\, a)) = \mathcal{O}(\max\{p, q\}\tau + \min\{p, q\}\sigma)$.

2.5.1 Bivariate sign evaluation

We now reduce the computation of the sign of $f \in \mathbb{Z}[x, y]$ over $(\alpha, \beta) \in \mathbb{R}^2_{alg}$ to that over several points in $\mathbb{Q}^2$. Let $\deg_x(f) = \deg_y(f) = n_1$, $\mathcal{L}(f) = \sigma$ and $\alpha \cong (A, [a_1, a_2])$, $\beta \cong (B, [b_1, b_2])$, where $A, B \in \mathbb{Z}[X]$, $\deg(A) = \deg(B) = n_2$,

Algorithm 3: BIVARIATE-SIGN AT.

Input: $\alpha = [A, [A_L, A_R]]$, $\beta = [B, [B_L, B_R]]$ and $f \in \mathbb{R}[x, y]$; A, B square-free.

Output: $\text{sign}(f(\alpha, \beta))$

```
1  PRS ⟵ SR(A, f, x);
2  PRS_1 ⟵ (PRS(A_L));
3  for i ⟵ 1 to |PRS| do
4      PRS_1[i] = SignAt(PRS_1[i], β);
5  V_L ⟵ VAR(PRS_1);
6  PRS_2 ⟵ (PRS(A_L));
7  for i ⟵ 1 to |PRS| do
8      PRS_2[i] = SignAt(PRS_2[i], β);
9  V_R ⟵ VAR(PRS_2);
10 return sign((V_L − V_R) · A'(α));
```

$\mathcal{L}(A) = \mathcal{L}(B) = \sigma$. We assume $n_1 \leqslant n_2$, which is relevant below. Algorithm 3 presents the whole procedure in pseudocode, see [Sak89], and generalizes the univariate case, e.g. [EMT07, Yap00] (algorithm 2). One has to obtain the sign variations of $SR(A, F;\, a_1)$, respectively $SR(A, F;\, a_2)$ and substract them. After the evaluation of the sequence on a_1, respectively a_2, polynomials in $\mathbb{Z}[y]$ occur. In order to obtain the sign variations, the sign of these polynomials over β has to be computed. This is performed with algorithm 2. For A, respectively B, we assume that we know their values on a_1, a_2, respectively b_1, b_2.

Theorem 2.37 (Bivariate sign_at)**.** *Let* $f \in \mathbb{Z}[x, y]$ *such that* $\deg_x(f) = \deg_y(f) = n_1$ *and* $\mathcal{L}(f) = \sigma$ *and two real algebraic numbers* $\alpha \cong (A, \mathcal{I}_\alpha) = [a_1, a_2]$, $\beta \cong (B, \mathcal{I}_\beta) = [b_1, b_2]$ *where* $A, B \in \mathbb{Z}[X]$, $\deg(A) = \deg(B) = n_2$, $\mathcal{L}(A) = \mathcal{L}(B) = \sigma$ *and* $\mathcal{I}_\alpha, \mathcal{I}_\beta \in \mathbb{Q}^2$*. Then, one computes the sign of* f *evaluated over* α *and* β *with complexity* $\widetilde{\mathcal{O}}_B(n_1^2\, n_2^3\, \sigma)$*, assuming that* $n_1 \leqslant n_2$.

Proof. First, we compute $SQ_x(A, f)$ so as to evaluate $SR(A, f)$ on the endpoints of α, in $\widetilde{\mathcal{O}}_B(n_1^2 n_2^2 \sigma)$ (corollary 2.35).

We compute $SR(A, f;\, a_1)$. The first polynomial in the sequence is A, but we already know its value on a_1. This computation costs $\widetilde{\mathcal{O}}_B(n_1^2\, n_2^3\, \sigma)$ by corollary 2.36 with $q = n_1$, $p = n_2$, $d = n_1$, $\tau = \sigma$, and $\sigma = n_2\sigma$, where the latter corresponds to the bitsize of the endpoints. After the evaluation we obtain a list L_1, which contains $\mathcal{O}(n_1)$ polynomials, say $h \in \mathbb{Z}[y]$, such that $\deg(h) = \mathcal{O}(n_1 n_2)$. To bound the bitsize, notice that the polynomials in $SR(f, g)$ are of degrees $\mathcal{O}(n_1)$ with respect to x and of bitsize $\mathcal{O}(n_2\sigma)$. After we evaluate on a_1, $\mathcal{L}(f) = \mathcal{O}(n_1 n_2 \sigma)$.

For each $h \in L_1$ we compute its sign over β and count the sign variations. We could apply directly corollary 2.27, but we can do better. If $\deg(h) \geqslant n_2$ then $SR(B, h) = (B, h, -B,\ g = -\text{prem}(h, -B), \dots)$. We start the evaluations at g: it is computed in $\widetilde{\mathcal{O}}_B(n_1^2 n_2^3 \sigma)$ (proposition 2.18), $\deg(g) = \mathcal{O}(n_2)$ and $\mathcal{L}(g) = \mathcal{O}(n_1 n_2 \sigma)$. Thus, we evaluate $SR(-B, g; a_1)$ in $\widetilde{\mathcal{O}}_B(n_1 n_2^3 \sigma)$, by

corollary 2.27, with $p = q = n_2$, $\tau_h = \sigma$, $\tau = n_1 n_2 \sigma$. If $\deg(h) < n_2$ the complexity is dominated. Since we perform $\mathcal{O}(n_1)$ such evaluations, all of them cost $\widetilde{\mathcal{O}}_B(n_1^2 n_2^3 \sigma)$.

We repeat for the other endpoint of α, subtract the sign variations, and multiply by $\operatorname{sign}(A'(\alpha))$, which is known from the process that isolated α. If the last sign in the two sequences is alternating, then $\operatorname{sign}(f(\alpha, \beta)) = 0$. □

Chapter 3

Real Solving of Bivariate Systems

This chapter studies algorithms and their complexity for real solving the system $f = g = 0$, for given $f, g \in \mathbb{Z}[x, y]$. For simplicity, let f, g be relatively prime polynomials. This hypothesis is not restrictive because it can be verified and if it does not hold, it can be imposed within the same asymptotic complexity. The main idea is to project the roots on the x and y axes, to compute the coordinates of the real solutions and somehow to match them. The difference between the algorithms is the way they match solutions. Projection is performed with resultants and signed polynomial remainder sequences. The output of the algorithms is a list with pairs of real algebraic numbers and, if possible, the multiplicities of the solutions. In what follows d bounds the total degree of f and g and σ bounds the bitsize of their respective coefficients.

3.1 The grid algorithm

The first algorithm that is studied is named GRID and is straightforward, see also [ET05, Wol02]. The first step consists of computing the $x-$ and $y-$coordinates of the real solutions, as real roots of the resultants $\mathtt{res}_x(f, g)$ and $\mathtt{res}_y(f, g)$. Then, matching is performed using the algorithm SIGN_AT (th. 2.37) by testing all rectangles in this grid. The output is a list of pairs of real algebraic numbers represented in isolating interval representation. The algorithm also outputs rational axis-aligned rectangles, guaranteed to contain a single root of the system. Algorithm 4 presents the solver in pseudocode.

Surprisingly, the first time that the algorithm's complexity was studied seems to be [DET07a, DET07b]. The disadvantage of the algorithm is that exact implementation of SIGN_AT (algorithm 3) is not efficient. However, its simplicity makes it attractive. The algorithm requires no genericity assumption on the input. Moreover, a generic shear that brings the system to generic position in order to compute the multiplicities within the same complexity bound will be discussed.

Algorithm 4: STURM::GRID.

Input: $f \in \mathbb{Z}[x,y]$, $g \in \mathbb{Z}[x,y]$.

Output: A list S of solutions $(x,y) \in \mathbb{R}_{alg} \times \mathbb{R}_{alg}$.

```
1 X ⟵ Solve(Resultant(f, g, y));
2 Y ⟵ Solve(Resultant(f, g, x));
3 S ⟵ ∅;
4 foreach x ∈ X do
5     foreach y ∈ Y do
6         if BivSignAt(f, x, y) = 0 and BivSignAt(g, x, y) = 0 then
7             S ⟵ S ∪ (x, y);
```

The algorithm allows the use of heuristics. In particular, we may exploit easily computed bounds on the number of roots, such as the Mixed Volume or count the roots with a given abscissa α by lemma 3.7.

Theorem 3.1. *Isolating all real roots of system* $f = g = 0$ *using* GRID *has complexity* $\widetilde{\mathcal{O}}_B(d^{14} + d^{13}\sigma)$, *provided* $\sigma = O(d^3)$.

Proof. First we compute the resultant of f and g w.r.t. variable y, i.e. R_x. The complexity is $\widetilde{\mathcal{O}}_B(d^4\sigma)$, using corollary 2.35. Notice that $\deg(R_x) = \mathcal{O}(N^2)$ and $\mathcal{L}(R_x) = \mathcal{O}(N\,\sigma)$. Applying proposition 2.26 help us isolate its real roots in time $\widetilde{\mathcal{O}}_B(d^{12} + d^{10}\sigma^2)$ and store them in list L_x. This complexity shall be dominated. The same procedure is done on y axis and the roots are stored this time in list L_y.

The representation of the real algebraic numbers that we have computed contains the square-free part of R_x, or R_y. In both cases the bitsize of the polynomial is $\mathcal{O}(d^2 + d\,\sigma)$ [BPM06, EMT07]. Moreover, the isolating intervals have endpoints of size $\mathcal{O}(d^4 + d^3\,\sigma)$.

Let r_x, respectively r_y be the number of real roots of the corresponding resultants. Both are bounded by $\mathcal{O}(d^2)$. We form all possible pairs of real algebraic numbers from lists L_x and L_y and check for every such pair if both f and g polynomials vanish. This check is performed with bivariate SIGN_AT function. Applying theorem 2.37 with $n_1 = d$, $n_2 = d^2$ and $\sigma = d^2 + d\sigma$ the cost of each evaluation is: $\widetilde{\mathcal{O}}_B(d^{10} + d^9\sigma)$. Overall, we have $r_x\, r_y = O(d^4)$ different pairs. □

The above algorithm suffices for bivariate solving. Another interesting problem is the the multiplicity of a root $(\alpha,\beta) \in \mathbb{R}_{alg} \times \mathbb{R}_{alg}$ of the system. Refer to [BK86, sec.II.6] for its definition as the exponent of factor $(\beta x - \alpha y)$ in the resultant of the (homogenized) polynomials, under certain assumptions.

The algorithm reduces to bivariate sign determination and does not require bivariate factorization. For this purpose, resultant is used, since it allows for multiplicities to "project". Previous work includes [GVEK96, SF90, WS05]. The sum of multiplicities of all roots (α,β_j) equals the multiplicity of $x = \alpha$ in the respective resultant. It is possible to apply a shear transform to the coordinate

frame so as to ensure that different roots project to different points on the x-axis.

3.1.1 Deterministic shear and counting multiplicities

One can determine an adequate (horizontal) shear such that

$$R_t(x) = \mathtt{res}_y\left(f(x+ty,y), g(x+ty,y)\right), \tag{3.1}$$

when $t \mapsto t_0 \in \mathbb{Z}$, has simple roots corresponding to the projections of the common roots of the system $f(x,y) = g(x,y) = 0$ and the degree of the polynomials remains the same. Notice that this shear does not affect inherently multiple roots, which exist independently of the reference frame. $R_{red} \in (\mathbb{Z}[t])[x]$ is the squarefree part of the resultant, as an element of UFD $(\mathbb{Z}[t])[x]$, and its discriminant, with respect to x, is $\Delta \in \mathbb{Z}[t]$. Then t_0 must be such that $\Delta(t_0) \neq 0$.

Example 3.1. *Take the circle* $f = (x-1)^2 + y^2 - 1$, *and the double line* $g = y^2$ *with two double roots* $(0,0), (2,0)$. *We shall project roots on the* y*-axis under the vertical shear:*

$$f(x, y+tx) = x^2(1+t^2) + 2x(ty-1) + y^2,\ g(x, y+tx) = y^2 + 2txy + (tx)^2.$$

Then, $R(y) = y^2[y^2(t^4+1) + 2t^3y + t^2]$. *The square-free part is* $R_{red}(y) = y[y^2(t^4+1) + 2t^3y + t^2]$ *and* $\Delta(t) = t^6(t^4+1)(3t^4+4)$. *Clearly, one must avoid the value* $t = 0$, *but any other integer is valid.*

Lemma 3.2. *Computing* $t_0 \in \mathbb{Z}$, *such that the corresponding shear is sufficiently generic, has complexity* $\widetilde{\mathcal{O}}_B(d^{10} + d^9\sigma)$.

Proof. Suppose t_0 is such that the degree does not change. It suffices to find, among d^4 integer numbers, one that does not make Δ vanish; note that all candidate values are of bitsize $\mathcal{O}(\log d)$.

We perform the substitution $(x,y) \mapsto (x+ty, y)$ to f and g and we compute the resultant w.r.t. y in $\widetilde{\mathcal{O}}_B(d^5\sigma)$, which is a polynomial in $\mathbb{Z}[t,x]$, of degree $\mathcal{O}(d^2)$ and bitsize $\widetilde{\mathcal{O}}(d\sigma)$. We consider this polynomial as univariate in x and we compute first its square-free part, and then the discriminant of its square-free part. Both operations cost $\widetilde{\mathcal{O}}_B(d^{10} + d^9\sigma)$ and the discriminant is a polynomial in $\mathbb{Z}[t]$ of degree $\mathcal{O}(d^4)$ and bitsize $\widetilde{\mathcal{O}}(d^4 + d^3\sigma)$.

We can evaluate the discriminant over all the first d^4 positive integers, in $\widetilde{\mathcal{O}}_B(d^8 + d^3\sigma)$, using the multipoint evaluation algorithm. Among these integers, there is at least one that is not a root of the discriminant. □

The idea here is to use explicit candidate values of t_0 right from the start. In practice, the above complexity becomes $\widetilde{\mathcal{O}}_B(d^5\sigma)$, because a constant number of tries or a random value will typically suffice. For an alternative approach see [GVN02], also [BPM06]. It is straightforward to compute the multiplicities of the sheared system. Then, we need to match the latter with the roots of the original system, which is nontrivial in practice.

Theorem 3.3. *Consider the setting of th. 3.1. Having isolated all real roots of* $f = g = 0$*, it is possible to determine their multiplicities in* $\widetilde{\mathcal{O}}_B(d^{12}+d^{11}\sigma+d^{10}\sigma^2)$.

Proof. By the previous lemma, $t \in \mathbb{Z}$ is determined, with $\mathcal{L}(t) = \mathcal{O}(\log d)$, in $\widetilde{\mathcal{O}}_B(d^{10} + d^9\sigma)$. Using this value, we isolate all the real roots of $R_t(x)$, defined in (3.1), and determine their multiplicities in $\widetilde{\mathcal{O}}_B(d^{12} + d^{10}\sigma^2)$. Let $\rho_j \simeq (R_t(x), [r_j, r'_j])$ be the real roots, for $j = 0, \ldots, r-1$.

By assumption, we have already isolated the roots of the system, denoted by $(\alpha_i, \beta_i) \in [a_i, a'_i] \times [b_i, b'_i]$, where $a_i, a'_i, b_i, b'_i \in \mathbb{Q}$ for $i = 0, \ldots, r-1$. It remains to match each pair (α_i, β_i) to a unique ρ_j by determining function $\phi : \{0, \ldots, r-1\} \to \{0, \ldots, r-1\}$, such that $\phi(i) = j$ iff $(\rho_j, \beta_i) \in \mathbb{R}^2_{alg}$ is a root of the sheared system and $\alpha_i = \rho_j + t\beta_i$.

Let $[c_i, c'_i] = [a_i, a'_i] - t[b_i, b'_i] \in \mathbb{Q}^2$. These intervals may be overlapping. Since the endpoints have bitsize $\mathcal{O}(d^4 + d^3\sigma)$, the intervals $[c_i, c'_i]$ are sorted in $\widetilde{\mathcal{O}}_B(d^6 + d^5\sigma)$. The same complexity bounds the operation of merging this interval list with the list of intervals $[r_j, r'_j]$. If there exist more than one $[c_i, c'_i]$ overlapping with some $[r_j, r'_j]$, some subdivision steps are required so that the intervals reach the bitsize of s_j, where 2^{s_j} bounds the separation distance associated to the j-th root. By proposition 2.29, $\sum_i s_i = \mathcal{O}(d^4 + d^3\sigma)$.

Our analysis resembles that of [EMT07] for proving proposition 2.26. The total number of steps is $\mathcal{O}(\sum_i s_i) = \mathcal{O}(d^4 + d^3\sigma)$, each requiring an evaluation of $R(x)$ over a endpoint of size $\leqslant s_i$. This evaluation costs $\widetilde{\mathcal{O}}_B(d^4 s_i)$, leading to an overall cost of $\widetilde{\mathcal{O}}_B(d^8 + d^7\sigma)$ per level of the tree of subdivisions. Hence the overall complexity is bounded by $\widetilde{\mathcal{O}}_B(d^{12} + d^{11}\sigma + d^{10}\sigma^2)$. □

3.2 The m_rur algorithm

M_RUR assumes that the polynomials are in Generic Position: different roots project to different x-coordinates and leading coefficients w.r.t. y have no common real roots.

Proposition 3.4. *[GVEK96, BPM06] Let* f, g *be co-prime polynomials, in generic position. If* $SR_j(x, y) = sr_j(x)y^j + sr_{j,j-1}(x)y^{j-1} + \cdots + sr_{j,0}(x)$*, and* (α, β) *is a real solution of the system* $f = g = 0$*, then there exists* k*, such that* $sr_0(\alpha) = \cdots = sr_{k-1}(\alpha) = 0$*,* $sr_k(\alpha) \neq 0$ *and* $\beta = -\frac{1}{k}\frac{sr_{k,k-1}(\alpha)}{sr_k(\alpha)}$.

This expresses the ordinate of a solution in a Rational Univariate Representation (RUR) of the abscissa. The RUR applies to multivariate algebraic systems [Ren89, Can88, Rou99, BPM06]; it generalizes the primitive element method by Kronecker. Here we adapt it to small-dimensional systems.

M_RUR is similar to [GVN02, GVEK96]. However, their algorithm computes only a RUR using proposition 3.4, so the representation of the ordinates remains implicit. Often, this representation is not sufficient (we can always compute the minimal polynomial of the roots, but this is highly inefficient). We modified the algorithm [ET05], so that the output includes isolating rectangles, hence the name modified-RUR (M_RUR). The most important difference with [GVEK96]

is that they represent algebraic numbers by Thom's encoding while the approach of this thesis is isolating intervals, which were thought of having high theoretical complexity. It has been proved that this is not the case [DET07a, DET07b].

The pseudo-code of M_RUR is in algorithm 5. Initially, projection is

Algorithm 5: STURM::M_RUR.

Input: $f \in \mathbb{Z}[x,y], g \in \mathbb{Z}[x,y]$.
Output: A list S of solutions $(x,y) \in \mathbb{R}_{alg} \times \mathbb{R}_{alg}$.

```
1 X ⟵ Solve(Resultant(f, g, y));
2 Y ⟵ Solve(Resultant(f, g, x));
3 S ⟵ ∅;
4 PRS ⟵ StHa (f, g, y);
5 Q ⟵ IntermediatePoints(Y);
6 K ⟵ Compute_K(PRS, X);
7 for i ⟵ 1 to |X| do
8 |   S ⟵ S ∪ (X_i, Find(X_i, K_i, PRS, Y, Q))
```

performed on the x and the y-axis; for each real solution on the x-axis its ordinate is computed using proposition 3.4. Using corollary 2.34 the sequence $SR(f,g)$ w.r.t. y is computed in $\widetilde{\mathcal{O}}_B(d^5\,\sigma)$ time.

3.2.1 Projection.

This is similar to GRID. By corollary 2.35 the computation of R_x has complexity $\widetilde{\mathcal{O}}_B(d^4\,\sigma)$. An alternative approach would be to compute R_x as the first non-vanishing polynomial, counting from the end, of the sequence $SR(f,g)$, since this step is not the bottleneck of the algorithm. Now $R_x \in \mathbb{Z}[X]$, $\deg(R_x) = \mathcal{O}(d^2)$, and $\mathcal{L}(R_x) = \mathcal{O}(d\,\sigma)$. By proposition 2.26 its roots are isolated in $\widetilde{\mathcal{O}}_B(d^{12} + d^{10}\,\sigma^2)$. The representation contains the square-free part of R_x, with bitsize $\mathcal{O}(d^2 + d\,\sigma)$, whereas the intervals' endpoints are rationals with aggregate bitsize $\mathcal{O}(d^3\,\sigma)$. Let the roots be

$$\alpha_1 < \alpha_2 < \cdots < \alpha_{m-1} < \alpha_m \tag{3.2}$$

where $m \leqslant 2d^2$ is the number of real roots of R_x. The multiplicity of α_i is the multiplicity of (α_i, β_j) as a solution of the system, $\beta_j \in \mathbb{R}_{alg}$.

For projection on the y-axis a similar procedure is performed. The real roots of R_y are in list L_y and their multiplicities in M_y. We compute rational numbers q_j between the real roots in $\widetilde{\mathcal{O}}_B(d^5\sigma)$; the q_j have aggregate bitsize $\mathcal{O}(d^3\,\sigma)$:

$$q_0 < \beta_1 < q_1 < \beta_2 < \cdots < \beta_{\ell-1} < q_{\ell-1} < \beta_\ell < q_\ell, \tag{3.3}$$

where $\ell \leqslant 2\,d^2$. Every β_j corresponds to a unique α_i. The multiplicity of α_i as a root of R_x is the multiplicity of a real solution of the system, that has it as abscissa.

3.2.2 The sub-algorithm compute_k

In order to apply proposition 3.4, for every α_i one must compute $k \in \mathbb{N}^*$ such the assumptions of the theorem are fulfilled; this is possible by genericity. Let

$$\Phi_0(x) = \frac{\mathrm{sr}_0(x)}{\gcd(\mathrm{sr}_0(x), \mathrm{sr}_0'(x))}.$$

Following [MPS+06, GVEK96] one can define recursively the polynomials $\Gamma_j(x)$:

$$\begin{array}{rclcrcl} \Phi_1(X) & = & \gcd(\Phi_0(X), \mathrm{sr}_1(X)) & & \Gamma_1 & = & \frac{\Phi_0(X)}{\Phi_1(X)} \\ \Phi_2(X) & = & \gcd(\Phi_1(X), \mathrm{sr}_2(X)) & & \Gamma_2 & = & \frac{\Phi_1(X)}{\Phi_2(X)} \\ & \vdots & & & & \vdots & \\ \Phi_{n-1}(X) & = & \gcd(\Phi_{n-2}(X), \mathrm{sr}_{n-1}(X)) & & \Gamma_{n-1} & = & \frac{\Phi_{n-2}(X)}{\Phi_{n-1}(X)} \end{array}$$

Now $\mathrm{sr}_i(x) \in \mathbb{Z}[x]$ is the principal subresultant coefficient of $\mathrm{SR}_i \in (\mathbb{Z}[x])[y]$, and $\Phi_0(x)$ is the square-free part of $R_x = \mathrm{sr}_0(x)$. By construction, $\Phi_0(x) = \prod_j \Gamma_j(x)$ and $\gcd(\Gamma_j, \Gamma_i) = 1$, if $j \neq i$. Hence every α_i is a root of a unique Γ_j and the latter switches sign at the interval's endpoints. Then, $\mathrm{sr}_0(\alpha) = \mathrm{sr}_1(\alpha) = 0, \ldots, \mathrm{sr}_j(\alpha) = 0$, $\mathrm{sr}_{j+1}(\alpha) \neq 0$; thus $k = j + 1$.

It holds that $\deg(\Phi_0) = \mathcal{O}(d^2)$ and $\mathcal{L}(\Phi_0) = \mathcal{O}(d^2 + d\,\sigma)$. Moreover, $\sum_j \deg(\Gamma_j) = \sum_j \delta_j = \mathcal{O}(d^2)$ and, by Mignotte's bound [MS99], $\mathcal{L}(\Gamma_j) = \mathcal{O}(d^2 + d\sigma)$. To compute the factorization $\Phi_0(x) = \prod_j \Gamma_j(x)$ as a product of the $\mathrm{sr}_j(x)$, we perform $\mathcal{O}(d)$ gcd computations of polynomials of degree $\mathcal{O}(d^2)$ and bitsize $\widetilde{\mathcal{O}}(d^2 + d\sigma)$. By proposition 2.18 each gcd computation costs $\widetilde{\mathcal{O}}_B(d^6 + d^5\,\sigma)$ and thus the overall cost is $\widetilde{\mathcal{O}}_B(d^7 + d^6\,\sigma)$.

By lemma 2.29 the sign of the Γ_j over all the $O(d^2)$ isolating endpoints of the α_i, which have aggregate bitsize $\mathcal{O}(d^4 + d^3\,\sigma)$ can be computed in $\widetilde{\mathcal{O}}_B(\delta_j d^4 + \delta_j d^3\sigma + \delta_j^2(d^4 + d^3\sigma))$, using Horner's rule. Summing over all δ_j, the complexity is $\widetilde{\mathcal{O}}_B(d^8 + d^7\sigma)$. Thus the overall complexity is $\widetilde{\mathcal{O}}_B(d^9 + d^8\,\sigma)$.

3.2.3 Matching solutions and algorithm find

The process takes a real root of R_x and computes ordinate β of the corresponding root of the system. For some real root α of R_x one represents the ordinate

$$A(\alpha) = -\frac{1}{k}\frac{\mathrm{sr}_{k,k-1}(\alpha)}{\mathrm{sr}_k(\alpha)} = \frac{A_1(\alpha)}{A_2(\alpha)}.$$

The generic position assumption guarantees that there is a unique β_j, in P_y, such that $\beta_j = A(\alpha)$, where $1 \leqslant j \leqslant \ell$. By 3.3 one can compute j such that:

$$q_j < A(\alpha) = \frac{A_1(\alpha)}{A_2(\alpha)} = \beta_j < q_{j+1}.$$

Thus j can be computed by binary search in $\mathcal{O}(\lg \ell) = \mathcal{O}(\lg d)$ comparisons of $A(\alpha)$ with the q_j. This is equivalent to computing the sign of $B_j(X) = A_1(X) - q_j\,A_2(X)$ over α by executing $\mathcal{O}(\lg d)$ times, SIGN_AT(B_j, α).

Now, $\mathcal{L}(q_j) = \mathcal{O}(d^4 + d^3\sigma)$ and $\deg(A_1) = \deg(sr_{k,k-1}) = \mathcal{O}(d^2)$, $\deg(A_2) = \deg(sr_k) = \mathcal{O}(d^2)$, $\mathcal{L}(A_1) = \mathcal{O}(d\,\sigma)$, $\mathcal{L}(A_2) = \mathcal{O}(d\,\sigma)$. Thus $\deg(B_j) = \mathcal{O}(d^2)$ and $\mathcal{L}(B_j) = \mathcal{O}(d^4 + d^3\,\sigma)$. Therefore, by corollary 2.27, SIGN_AT(B_j, α) and FIND have complexity $\widetilde{\mathcal{O}}_B(d^8 + d^7\sigma)$. As for the overall complexity of the loop (lines 7-8) the complexity is $\widetilde{\mathcal{O}}_B(d^{10} + d^9\sigma)$, since it is executed $\mathcal{O}(d^2)$ times.

Theorem 3.5. *Let $f, g \in \mathbb{Z}[x, y]$ such that they are in generic position, their total degrees are bounded by d, and their bitsize by σ. If the polynomials are not relatively prime, the algorithm reports this and stops. Otherwise, it isolates all real roots of the system $f = g = 0$ with complexity $\widetilde{\mathcal{O}}_B(d^{12} + d^{10}\sigma^2)$.*

The generic position assumption is without loss of generality since we can always put the system in such position by applying a shear transform; $(X, Y) \mapsto (X + tY, Y)$, where t is either a random number or computed deterministically, see [GVEK96, SF90] or section 3.1.1. The bitsize of the polynomials of the (sheared) system becomes $\widetilde{\mathcal{O}}(d + \sigma)$ [GVEK96] and does not change the bound of theorem 3.5. However, now is raised the problem of expressing the real roots in the original coordinate system (see also the proof of theorem 3.3).

3.3 The g_rur algorithm

The last algorithm for bivariate solving that is presented uses some ideas from RUR but relies on GCD computations of polynomials with coefficients in an extension field to achieve efficiency (hence the name G_RUR). For the GCD computations the algorithm (and the implementation) of [vHM02] is used. The algorithm is presented in pseudocode in algorithm 6.

Algorithm 6: STURM::G_RUR.

Input: $f \in \mathbb{Z}[x, y], g \in \mathbb{Z}[x, y]$.
Output: A list S of solutions $(x, y) \in \mathbb{R}_{alg} \times \mathbb{R}_{alg}$.

```
1  X ⟵ Solve(Resultant(f, g, y));
2  Y ⟵ Solve(Resultant(f, g, x));
3  Inter ⟵ IntermediatePoints(Y);
4  S ⟵ ∅;
5  for i ← 1 to |X| do
6      f_{X_i}(y) ⟵ Square-Free-Part(f(X_i, y)) ∈ Z[X_i][y];
7      g_{X_i}(y) ⟵ Square-Free-Part(g(X_i, y)) ∈ Z[X_i][y];
8      H ⟵ gcd(f_{X_i}(y), g_{X_i}(y));
9      for j ← 1 to |Inter| − 1 do
10         if H(Inter_j) · H(Inter_{j+1}) < 0 then
11             S ⟵ S ∪ (X_i, Y_j)
```

The first steps are similar to the previous algorithms: Projecting on both axes, real solving the respective resultants and computing the intermediate

points on the y-axis. It has already been shown that the complexity of these steps is $\widetilde{\mathcal{O}}_B(d^{12} + d^{10}\sigma^2)$.

For each x-coordinate, say α, we compute the square-free part of $f(\alpha, y)$ and $g(\alpha, y)$, say $\bar{f}$ and $\bar{g}$. The complexity is that of computing the gcd with the derivative. In [vHM02] the cost is $\widetilde{\mathcal{O}}_B(mMND + mN^2D^2 + m^2kD)$, where M is the bitsize of the largest coefficient, N is the degree of the largest polynomial, D is the degree of the extension, k is the degree of the gcd, and m is the number of primes needed. The complexity does not assume fast multiplication algorithms, thus, under this assumption, it becomes $\widetilde{\mathcal{O}}_B(mMND + mND + mkD)$.

In this case $M = \mathcal{O}(\sigma)$, $N = \mathcal{O}(d)$, $D = \mathcal{O}(d^2)$, $k = \mathcal{O}(d)$, and $m = \mathcal{O}(d\sigma)$. The cost is $\widetilde{\mathcal{O}}_B(d^4\sigma^2)$ and since we have to do it $\mathcal{O}(d^2)$ times, the overall cost is $\widetilde{\mathcal{O}}_B(d^6\sigma^2)$. Notice the bitsize of the result is $\widetilde{\mathcal{O}}_B(d + \sigma)$ [BPM06].

Now for each α, we compute $H = \gcd(\bar{f}, \bar{g})$. We have $M = \mathcal{O}(d + \sigma)$, $N = \mathcal{O}(d)$, $D = \mathcal{O}(d^2)$, $k = \mathcal{O}(d)$, and $m = \mathcal{O}(d^2 + d\sigma)$ and so the cost of each operation is $\widetilde{\mathcal{O}}_B(d^6 + d^4\sigma^2)$ and overall $\widetilde{\mathcal{O}}_B(d^8 + d^6\sigma^2)$. The size of m comes from Mignotte's bound [MS99]. Notice that H is a square-free polynomial in $(\mathbb{Z}[\alpha])[y]$, of degree $\mathcal{O}(d)$ and bitsize $\mathcal{O}(d^2 + d\sigma)$, the real roots of which correspond to the real solutions of the system with abscissa α. It should change sign only over the intervals that contain its real roots. To check these signs, we have to substitute y in H by the intermediate points, thus obtaining a polynomial in $\mathbb{Z}[\alpha]$, of degree $\mathcal{O}(d)$ and bitsize $\mathcal{O}(d^2 + d\sigma + ds_j)$, where s_j is the bitsize of the j-th intermediate point.

Now, we consider this polynomial in $\mathbb{Z}[x]$ and evaluate it over α. Using corollary 2.27 with $p = d^2$, $\tau_f = d^2 + d\sigma$, $q = d$, and $\tau_g = d^2 + d\sigma + ds_j$, this costs $\widetilde{\mathcal{O}}_B(d^6 + d^5\sigma + d^4 s_j)$. Summing over $\mathcal{O}(d^2)$ points and using lemma 2.29, we obtain $\widetilde{\mathcal{O}}_B(d^8 + d^7\sigma)$. Thus, the overall complexity is $\widetilde{\mathcal{O}}_B(d^{10} + d^9\sigma)$.

Theorem 3.6. *Isolating all real roots of the system* $f = g = 0$*, using* `G_RUR` *in* $\widetilde{\mathcal{O}}_B(d^{12} + d^{10}\sigma)$.

3.4 Applications

This section deals with applications of the algorithms and the complexity results that were presented earlier in this chapter in closely related problems.

3.4.1 Real root counting.

Let $F \in \mathbb{Z}[x, y]$, such that $\deg_x(F) = \deg_y(F) = n_1$ and $\mathcal{L}(F) = \sigma$. Let $\alpha, \beta \in \mathbb{R}_{alg}$, such that $\alpha = (A, [a_1, a_2])$ and $\beta = (B, [b_1, b_2])$, where $\deg(A), \deg(B) = n_2, \mathcal{L}(A), \mathcal{L}(B) \leqslant \tau$ and $c \in \mathbb{Q}$, such that $\mathcal{L}(c) = \lambda$. Moreover, assume that $n_1^2 = \mathcal{O}(n_2)$. We want to count the number of real roots of $\bar{F} = F(\alpha, y) \in (\mathbb{Z}(\alpha))[y]$ in $(-\infty, +\infty)$, in $(c, +\infty)$ and in $(\beta, +\infty)$.

We may assume that the leading coefficient of $\bar{F}$ is nonzero. This is without loss of generality since we can easily check it, and/or we can use the good specialization properties of the subresultants [LR01, GVLRR89, GVEK96].

Using Sturm's theorem, e.g. [BPM06, Yap00], the number of real roots of $\bar{F}$ is $\mathtt{VAR}(SR(\bar{F}, \bar{F}_y; -\infty)) - \mathtt{VAR}(SR(\bar{F}, \bar{F}_y; +\infty))$. Hence, we have to compute the sequence $SR(\bar{F}, \bar{F}_y)$ with respect to variable y, and evaluate it on $\pm\infty$, or equivalently to compute the signs of the principal subresultant coefficients, which lie in $\mathbb{Z}(\alpha)$.

The above procedure is equivalent, due to the good specialization properties of subresultants [BPM06, GVLRR89], to that of computing the principal subresultant coefficients of $SR(F, F_y)$, which are polynomials in $\mathbb{Z}[x]$, and to evaluate them over α. In other words the good specialization properties assure us that we can compute a nominal sequence by considering the bivariate polynomials, and then perform the substitution $x = \alpha$.

The sequence, sr, of the principal subresultant coefficients can be computed in $\widetilde{\mathcal{O}}_B(n_1^4\sigma)$, using corollary 2.35 with $p = q = d = n_1$, and $\tau = \sigma$. The sequence sr, contains $\mathcal{O}(n_1)$ polynomials in $\mathbb{Z}[x]$, each of degree $\mathcal{O}(n_1^2)$ and bitsize $\mathcal{O}(n_1\sigma)$. We compute the sign of each one evaluated over α in

$$\widetilde{\mathcal{O}}_B(n_1^2 n_2 \max\{\tau, n_1\sigma\} + n_2 \min\{n_1^2, n_2\}^2\tau)$$

using corollary 2.27 with $p = n_2$, $q = n_1^2$, $\tau_f = \tau$, and $\tau_g = n_1\sigma$. This proves the following:

Lemma 3.7. *We count the number of real roots of* $\bar{F}$ *in* $\widetilde{\mathcal{O}}_B(n_1^4 n_2\sigma + n_1^5 n_2\tau)$.

In order to compute the number of real roots of $\bar{F}$ in $(\beta, +\infty)$, we use again Sturm's theorem. The complexity of the computation is dominated by the cost of computing $\mathtt{VAR}(SR(\bar{F}, \bar{F}_y; \beta))$, which is equivalent to computing $SR(F, F_y)$ with respect to variable y, which contains bivariate polynomials, and to compute their signs over (α, β). The cost of computing $SR(F, F_y)$ is $\widetilde{\mathcal{O}}_B(n_1^5\sigma)$ using corollary 2.34 with $p = q = d = n_1$, and $\tau = \sigma$. The sequence contains $\mathcal{O}(n_1)$ polynomials in $\mathbb{Z}[x, y]$ of degrees $\mathcal{O}(n_1)$ and $\mathcal{O}(n_1^2)$, with respect to variables x and y respectively, and bitsize $\mathcal{O}(n_1\sigma)$. We can compute the sign of each of them evaluated it over (α, β) in $\widetilde{\mathcal{O}}_B(n_1^4 n_2^3 \max\{n_1\sigma, \tau\})$ (theorem 2.37). This proves the following:

Lemma 3.8. *We can count the number of real roots of* $\bar{F}$ *in* $(\beta, +\infty)$ *in* $\widetilde{\mathcal{O}}_B(n_1^5 n_2^3 \max\{n_1\sigma, \tau\})$.

By a more involved analysis, taking into account the difference in the degrees of the bivariate polynomials, we can gain a factor. This is omitted for simplicity reasons.

Finally, in order to count the real roots of $\bar{F}$ in $(c, +\infty)$, it suffices to evaluate the sequence $SR(F, F_y)$ with respect to variable y on c, thus obtaining polynomials in $\mathbb{Z}[x]$ and compute the signs of these polynomials evaluated over α.

The cost of the evaluation $SR(F, F_y; c)$ is $\widetilde{\mathcal{O}}_B(n_1^4 \max\{\sigma, \lambda\})$, using corollary 2.36 with $p = q = d = n_1$, $\tau = \sigma$ and $\sigma = \lambda$. The evaluated sequence contains $\mathcal{O}(n_1)$ polynomials in $\mathbb{Z}[x]$, of degree $\mathcal{O}(n_1^2)$ and bitsize $\mathcal{O}(n_1 \max\{\sigma, \lambda\})$. The sign of each one evaluated over α can be compute in

$$\widetilde{\mathcal{O}}_B(n_1^2 n_2 \max\{\tau, n_1\sigma, n_1\lambda\} + n_1^4 n_2\tau),$$

using corollary 2.27 with $p = n_2$, $q = n_1^2$, $\tau_f = \tau$ and $\tau_g = n_1 \max\{\sigma, \lambda\}$. This leads to the following:

Lemma 3.9. *We can count the number of real roots of* $\bar{F}$ *in* $(c, +\infty)$ *in* $\widetilde{\mathcal{O}}_B(n_1^4 n_2 \max\{n_1\tau, \sigma, \lambda\})$.

3.4.2 Simultaneous inequalities in two variables.

Let $P, Q, A_1, \ldots, A_{\ell_1}, B_1, \ldots, B_{\ell_2}, C_1, \ldots, C_{\ell_3} \in \mathbb{Z}[X, Y]$, such that their total degrees are bounded by n and their bitsize by σ. We wish to compute $(\alpha, \beta) \in \mathbb{R}^2_{alg}$ such that $P(\alpha, \beta) = Q(\alpha, \beta) = 0$ and also $A_i(\alpha, \beta) > 0$, $B_j(\alpha, \beta) < 0$ and $C_k(\alpha, \beta) = 0$, where $1 \leqslant i \leqslant \ell_1, 1 \leqslant j \leqslant \ell_2, 1 \leqslant k \leqslant \ell_3$. Let $\ell = \ell_1 + \ell_2 + \ell_3$.

Corollary 3.10. *There is an algorithm that solves the problem of* ℓ *simultaneous inequalities of degree* $\leqslant n$ *and bitsize* $\leqslant \sigma$*, in* $\widetilde{\mathcal{O}}_B(\ell n^{12} + \ell n^{11}\sigma + n^{10}\sigma^2)$.

Proof. Initially we compute the isolating interval representation of the real roots of $P = Q = 0$ in $\widetilde{\mathcal{O}}_B(n^{12} + n^{10}\sigma^2)$, using GRUR. There are $\mathcal{O}(n^2)$ real solutions, which are represented in isolating interval representation, with polynomials of degrees $\mathcal{O}(n^2)$ and bitsize $\mathcal{O}(n^2 + n\sigma)$.

For each real solution, say (α, β), for each polynomial A_i, B_j, C_k we compute the signs of $\operatorname{sign}(A_i(\alpha, \beta))$, $\operatorname{sign}(B_i(\alpha, \beta))$ and $\operatorname{sign}(C_i(\alpha, \beta))$. Each sign evaluation costs $\widetilde{\mathcal{O}}_B(n^{10} + n^9\sigma)$, using theorem 2.37 with $n_1 = n$, $n_2 = n^2$ and $\sigma = n^2 + n\sigma$. In the worst case we need n^2 of them, hence, the cost for all sign evaluations is $\widetilde{\mathcal{O}}_B(\ell n^{12} + \ell\, n^{11}\, \sigma)$. □

3.4.3 The complexity of topology.

The complexity of computing the topology of a real plane algebraic curve is also improved. See [BPM06, GVEK96, MPS+06] for the algorithm.

In studying Algebraic curves we use the following:

Lemma 3.11. *Given* $f \in \mathbb{Z}[x, y]$*, the shear transformation commutes with differentiation with respect to variable* x*, while it does not commute with differentiation with respect to variable* y*. In other words the following hold:*

$$\left.\frac{d}{dx} f(x, y)\right|_{x=x+ty} = \frac{d}{dx} f(x + ty, y), \quad t \in \mathbb{Z}$$

$$\left.\frac{d}{dy} f(x, y)\right|_{x=x+ty} \neq \frac{d}{dy} f(x + ty, y), \quad t \in \mathbb{Z}$$

Proof. Regarding the first part that deals with variable x it holds that:

$$\frac{d}{dx} f(x, y) = \frac{d}{dx} \sum_{i=0}^{N} a_i x^i h_i(y) = \sum_{i=1}^{N} a_i h_i(y) i x^{i-1}$$

and

$$\begin{aligned}\frac{d}{dx}f(x+ty,y) &= \frac{d}{dx}\sum_{i=1}^{N} a_i(x+ty)^i h_i(y) + \frac{d}{dx}h_0(y)\\ &= \sum_{i=1}^{N} a_i h_i(y) i(x+ty)^{i-1}.\end{aligned}$$

However, assuming that f depends on x the shear transform does not commute with differentiation with respect to variable y. For a counter-example, take $f(x,y)=x$. □

We consider the curve, in generic position, defined by $F \in \mathbb{Z}[x,y]$, such that $\deg(F)=n$ and $\mathcal{L}(F)=\sigma$. We compute the critical points of the curve, i.e. solve $F=F_y=0$ in $\widetilde{\mathcal{O}}_B(n^{12}+n^{10}\sigma^2)$. Next, we compute the intermediate points on the x axis, in $\widetilde{\mathcal{O}}_B(n^4+n^3\sigma)$ (lemma 2.29). For each intermediate point, say q_j, we need to compute the number of branches of the curve that cross the vertical line $x=q_j$. This is equivalent to computing the number of real solutions of the polynomial $F(q_j,y)\in\mathbb{Z}[y]$, which has degree d and bitsize $\mathcal{O}(n\mathcal{L}(q_j))$. For this we use Sturm's theorem and theorem 2.19 and the cost is $\widetilde{\mathcal{O}}_B(n^3\mathcal{L}(q_j))$. For all q_j's the cost is $\widetilde{\mathcal{O}}_B(n^7+n^6\sigma)$.

For each critical point, say (α,β) we need to compute the number of branches of the curve that cross the vertical line $x=\alpha$, and the number of them that are above $y=\beta$. The first task corresponds to computing the number of real roots of $F(\alpha,y)$, by application of lemma 3.7, in $\widetilde{\mathcal{O}}_B(n^9+n^8\sigma)$, where $n_1=n$, $n_2=n^2$, and $\tau=n^2+n\sigma$. Since there are $\mathcal{O}(n^2)$ critical values, the overall cost of the step is $\widetilde{\mathcal{O}}_B(n^{11}+n^{10}\sigma)$.

Finally, we compute the number of branches that cross the line $x=\alpha$ and are above $y=\beta$. We do this by lemma 3.8, in $\widetilde{\mathcal{O}}_B(n^{13}+n^{12}\sigma)$. Since there are $\mathcal{O}(n^2)$ critical points, the complexity is $\widetilde{\mathcal{O}}_B(n^{15}+n^{14}\sigma)$. It remains to connect the critical points according to the information that we have for the branches. The complexity of this step is dominated. It now follows that the complexity of the algorithm is $\widetilde{\mathcal{O}}_B(n^{15}+n^{14}\sigma+n^{10}\sigma^2)$, or $\widetilde{\mathcal{O}}_B(N^{15})$, which is worse by a factor than [BPM06].

We improve the complexity of the last step since M_RUR computes the RUR representation of the ordinates. Thus, instead of performing bivariate sign evaluations in order to compute the number of branches above $y=\beta$, we can substitute the RUR representation of β and perform univariate sign evaluations. This corresponds to computing the sign of $\mathcal{O}(n^2)$ polynomials of degree $\mathcal{O}(n^2)$ and bitsize $\mathcal{O}(n^4+n^3\sigma)$, over all the α's [GVEK96]. Using lemma 2.29 for each polynomial the cost is $\widetilde{\mathcal{O}}_B(n^{10}+n^9\sigma)$, and since there are $\widetilde{\mathcal{O}}_B(n^2)$ of them, the total cost is $\widetilde{\mathcal{O}}_B(n^{12}+n^{11}\sigma)$.

Theorem 3.12. *The topology of a real plane algebraic curve, defined by a polynomial of degree n and bitsize σ, can be computed in $\widetilde{\mathcal{O}}_B(n^{12}+n^{11}\sigma+n^{10}\sigma)$.*

Thus the overall complexity of the algorithm improves the previously known bound by a factor of N^2. We assumed generic position, since we can apply a shear to achieve this; refer to section 3.1.1.

Chapter 4

Implementation and Experiments

This chapter describes the open source MAPLE implementation[1] that was created as part of this thesis and illustrates its capabilities through comparative experiments. The design is based on object oriented programming and the generic programming paradigm in view of transferring the implementation to C++ in the future.

The class of real algebraic numbers represents them in isolating interval representation. We provide algorithms for computing *signed* polynomial remainder sequences; more particularly euclidean, primitive-part, subsresultant and Sturm-Habicht sequences. In addition to that, we perform real solving of univariate polynomials using Sturm's algorithm, and allow computations with one and two real algebraic numbers, such as sign evaluation and comparison. Finally, the current implementation exhibits the algorithms for real solving of bivariate systems that were mentioned in chapter 3.

However, in order to speedup the various computations and create a more real-world library, filtering techniques have been used. For this purpose, two instances of the rational endpoints that define the isolating intervals of the various real algebraic numbers are stored; one pair of endpoints (usually with larger bitsize) is used in filtering techniques, while the other one is used for exact computations via Sturm sequences.

4.1 Augmenting performance

This section is devoted to the filtering techniques that are currently used in the library.

A. **Pre-computation filtering in** M_RUR
Recall that M_RUR binary-searches for solutions along the y-axis. For

[1] www.di.uoa.gr/~erga/soft/SLV_index.html

this reason the intervals of candidate solutions along the x-axis are refined [Abb06] in order to help the interval arithmetic filters (refer to the following paragraph) that will be used inside the FIND procedure.

B. **Interval Arithmetic**
In cases where one wants to compute the sign of a polynomial evaluated at a real algebraic number, the first attempt is to yield the result via interval arithmetic techniques. The reader may refer to [Neu90] for details in the evaluations that arise. This filter is applied heuristically several times, based on the total degree of the input polynomials, with a combination of quadratic refinement of the defining intervals [Abb06] between executions in each loop.

C. **GCD**
In cases where the above filter fails to yield a result and one either wants to compare two real algebraic numbers or perform univariate SIGN_AT the gcd of the two polynomials that are involved is computed. By definition, the gcd of the two polynomials has a root in (the intersection of) the intervals if and only if both polynomials have a same root, in which case the two numbers are equal, or equivalently the required sign is zero.

Concluding, if both of the above filtering techniques fail, the library switches to exact and costly computations via Sturm sequences. Note however, that in these computations the rational endpoints with higher bitsize that have arisen through the above filtering techniques are not used; instead the initial endpoints with smaller bitsize are used.

4.2 Bivariate solving and slv library

In order to evaluate the implementation we have performed tests with the polynomial systems that are presented in section A.1. The performance of the implemented algorithms for bivariate solving is averaged over 10 iterations in MAPLE 9.5 console and is shown in table 4.1. Polynomial systems R_i, M_i, and D_i are presented in [ET05], systems C_i in [GVN02], and W_i are the C_i after swapping the x and y variables. Note that systems C_i and W_i are of the form $f = \frac{\partial f}{\partial y} = 0$ that arise in the topology of real plane algebraic curves. Finally, the polynomial system W_5 is not generated since the initial curve is a symmetric polynomial.

Recall that computations are performed first using intervals with floating point arithmetic (as it was described in section 4.1) and, if they fail, then an exact algorithm using rational arithmetic is called. For GCD computations in an extension field the MAPLE package of [vHM02] is used. Finally, also note that the optimal algorithms for computing and evaluating polynomial remainder sequences have not yet been implemented. Hence, it is reasonable to expect more efficient computations on a future release of the library.

It seems that G_RUR is the solver of choice since it is faster than GRID and M_RUR in 17 out of the 18 instances. However, this may not hold when the

system	deg		$\mathbb{R}_{alg}$	Average Time (msecs)		
	f	g	solutions	GRID	M_RUR	G_RUR
R_1	3	4	2	5	9	5
R_2	3	1	1	66	21	36
R_3	3	1	1	1	2	1
M_1	3	3	4	87	72	10
M_2	4	2	3	4	5	4
M_3	6	3	5	803	782	110
M_4	9	10	2	218	389	210
D_1	4	5	1	6	12	6
D_2	2	2	4	667	147	128
C_1	7	6	6	1,896	954	222
C_2	4	3	6	177	234	18
C_3	8	7	13	580	1,815	75
C_4	8	7	17	5,903	80,650	370
C_5	16	15	17	$> 20'$	60,832	3,877
W_1	7	6	9	2,293	2,115	247
W_2	4	3	5	367	283	114
W_3	8	7	13	518	2,333	24
W_4	8	7	17	5,410	77,207	280

Table 4.1: Performance averages over 10 runs in MAPLE 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

extension field is of high degree. G_RUR yields solutions in less than a second, apart from system C_5. Overall, for total degrees $\leqslant 8$, G_RUR requires less than 0.4 secs to respond. On average, G_RUR is 7-11 times faster than GRID, and about 38 times than M_RUR. The inefficiency of M_RUR can be justified by the fact that M_RUR solves sheared systems which are dense and of increased bitsize w.r.t. the original systems. Finally, it should be noted that GRID reaches a stack limit with the default MAPLE stack size (8,192) when trying to solve system C_5. However, even when we increased the stack ten times, GRID could not yield all solutions within 20 minutes. Setting the stack size to the required limit can be done with the following MAPLE command:

```
kernelopts(stacklimit=81920);
```

4.2.1 Comparing slv solvers

The following two paragraphs will briefly compare G_RUR with GRID and M_RUR in bivariate solving.

g_rur vs. grid

Table 4.2 presents running times for bivariate solving between GRID and G_RUR. The final column in this table indicates the speedup that is achieved when preferring G_RUR for bivariate solving. In other words, speedup $= \frac{\text{TIME}_{\text{GRID}}}{\text{TIME}_{\text{G\_RUR}}}$. As it is shown from the table G_RUR can be up to 21.58 times faster than GRID

system	Average Time		speedup
	GRID	G_RUR	
R_1	5	5	1.00
R_2	66	36	1.83
R_3	1	1	1.00
M_1	87	10	8.70
M_2	4	4	1.00
M_3	803	110	7.30
M_4	218	210	1.04
D_1	6	6	1.00
D_2	667	128	5.21
C_1	1,896	222	8.54
C_2	177	18	9.83
C_3	580	75	7.73
C_4	5,903	370	15.95
C_5	$> 20'$	3,877	–
W_1	2,293	247	9.28
W_2	367	114	3.22
W_3	518	24	21.58
W_4	5,410	280	19.32

Table 4.2: The performance of GRID and G_RUR implementations on bivariate solving and the speedup that is achieved when choosing G_RUR.

with an average speedup of around 7.27 among the input systems and excluding system C_5 where GRID failed to reply within 20 minutes. Moreover, in terms of total computing times for the entire test-set (again excluding system C_5) we can observe that:

- Total time for GRID = 19,001 msecs.
- Total time for G_RUR = 1,860 msecs.

In other words, the speedup in terms of total computing time is about 10.22.

g_rur vs. m_rur

Table 4.3 presents running times for bivariate solving between M_RUR and G_RUR. Similarly with the previous table, the final column indicates the speedup that is achieved when preferring G_RUR for bivariate solving. As it

system	Average Time		speedup
	M_RUR	G_RUR	
R_1	9	5	1.80
R_2	21	36	0.58
R_3	2	1	2.00
M_1	72	10	7.20
M_2	5	4	1.25
M_3	782	110	7.11
M_4	389	210	1.85
D_1	12	6	2.00
D_2	147	128	1.15
C_1	954	222	4.30
C_2	234	18	13.00
C_3	1,815	75	24.20
C_4	80,650	370	217.97
C_5	60,832	3,877	15.69
W_1	2,115	247	8.56
W_2	283	114	2.48
W_3	2,333	24	97.21
W_4	77,207	280	275.74

Table 4.3: The performance of M_RUR and G_RUR implementations on bivariate solving and the speedup that is achieved when choosing G_RUR.

is shown from the table G_RUR can be up to 275.74 times faster than M_RUR with an average speedup of around 38.01 among the input polynomial systems. Moreover, in terms of total computing times for the entire test-set we can observe that:

- Total time for M_RUR = 227,862 msecs.
- Total time for G_RUR = 5,737 msecs.

In other words, the speedup in terms of total computing time is about 39.72.

Again, it should be noted that M_RUR solves sheared systems which are dense and of increased bitsize. In addition to that, since the polynomial systems are sheared (whenever necessary) in M_RUR's case, M_RUR also computes the multiplicities on the intersections. A more accurate comparison will follow when all solvers will compute solutions on the same sheared systems and hence all of them will be able to decide the multiplicities on the intersections.

4.2.2 Decomposing running times

The following paragraphs demonstrate the decomposition of computing-time required by each algorithm in its respective major function calls as these timings were measured in the test-bed polynomial systems. Table 4.5 presents detailed

statistics of every algorithm on every polynomial system from the test-set, while table 4.4 tries to capture the basic statistical properties of the previous table.

The major function calls and thereby the decomposition of running times and the respective entries on the above tables can be summarized as follows. `Projections` shows the time for the computation of the resultants, `Univ. Solving` for real solving the resultants, and `Sorting` for sorting solutions. In GRID's and M_RUR's case, `biv. solving` corresponds to matching. In G_RUR's case timings for matching are divided between `rational biv.` and $\mathbb{R}_{alg}$ `biv.`; the first refers to when at least one of the co-ordinates is a rational number, while the latter indicates timings when both co-ordinates are not rational. `Inter. points` refers to computation of the intermediate points between resultant roots along the y-axis. `StHa seq.` refers to the computation of the StHa sequence. `Filter x-cand` shows the time for additional filtering. `Compute K` reflects the time for sub-algorithm COMPUTE-K.

	phase of the algorithm	interval		median	mean	std dev
		min	max			
GRID	projections	00.00	00.53	00.04	00.08	00.13
	univ. solving	02.05	99.75	07.08	26.77	35.88
	biv. solving	00.19	97.93	96.18	73.03	36.04
	sorting	00.00	01.13	00.06	00.12	00.26
MRUR	projection	00.00	00.75	00.06	00.14	00.23
	univ. solving	00.18	91.37	15.55	17.47	20.79
	StHa seq.	00.08	38.23	01.17	05.80	09.91
	inter. points	00.00	03.23	00.09	00.32	00.75
	filter x-cand	00.68	72.84	26.68	23.81	21.93
	compute K	00.09	34.37	02.04	07.06	10.21
	biv. solving	01.77	98.32	51.17	45.41	28.71
GRUR	projections	00.02	03.89	00.23	00.48	00.88
	univ. solving	07.99	99.37	39.83	41.68	25.52
	inter. points	00.02	03.81	00.54	01.11	01.28
	rational biv.	00.07	57.07	14.83	15.89	19.81
	$\mathbb{R}_{alg}$ biv.	00.00	91.72	65.30	40.53	36.89
	sorting	00.00	01.50	00.22	00.32	00.43

Table 4.4: Statistics on the performance of SLV's algorithms in bivariate solving.

In a nutshell, GRID spends more than 73% of its time in matching. Recall that this percent includes the application of filters and does not take into account the polynomial system C_5 where GRID failed to reply within 20 minutes. M_RUR spends about 45-50% of its time in matching and about 24-27% in the pre-computation filtering technique. G_RUR spends 55-80% of its time in matching, including gcd computations in an extension field.

Note also the significance of table 4.5 in order to draw further conclusions regarding the current implementation. Table 4.4 provides a mean of around

6% for the computation of the StHa sequence of f and g required by M_RUR. However, we can observe that this step might very well take up to 38.23% of the total computing time. Indeed, a closer look on table 4.5 reveals that this is the case for the difficult system C_5. Moreover, by table 4.1 we can observe that M_RUR requires about 61 seconds to solve system C_5. Hence, we can obtain a practical lower bound of about 23 seconds for M_RUR in this case, which is already bad compared to the performance of G_RUR for the entire problem (solving the system). This is a consequence and also a reminder for future work on the implementation of optimal algorithms on subresultant and Sturm-Habicht sequences. As a very important sidenote it should be stressed that implementing these optimal algorithms in sequences computations, the overall performance results for *all* solvers will be improved since the entire library is based on Sturm sequences to perform computations, such as pure univariate solving (root isolation), comparison of real algebraic numbers, and univariate and bivariate sign determination of functions evaluated respectively over one or a pair of real algebraic numbers.

System	GRID			M_RUR							G_RUR				
	Projections	**Univariate**	**Bivariate**	**Projection on x-axis**	**Univariate**	**StHa Sequence**	**Interm. Points**	**Filtering on x-axis**	**Compute K**	**FIND (Biv. Sol.)**	**Projections**	**Univariate**	**Interm. Points**	**Rational Bivariate**	**$\mathbb{R}_{alg}$ Bivariate**
R_1	0.19	73.71	25.78	0.06	28.30	17.91	0.64	1.21	19.79	32.09	0.22	53.75	2.08	43.71	0.02
R_2	0.01	4.47	95.52	0.00	16.30	0.61	0.09	72.84	3.50	6.66	0.07	7.99	0.12	0.10	91.72
R_3	0.53	78.46	20.84	0.17	33.04	20.01	0.97	2.79	27.45	15.57	0.67	40.29	1.85	57.07	0.04
M_1	0.04	10.13	89.75	0.05	21.06	1.46	0.14	35.63	2.97	38.69	0.14	79.62	2.83	16.13	0.02
M_2	0.13	56.29	42.45	0.12	32.57	9.49	3.23	0.68	34.37	19.54	0.48	39.83	3.81	55.07	0.00
M_3	0.00	4.98	95.02	0.02	7.39	0.16	0.02	60.60	1.18	30.62	0.03	28.60	0.67	0.50	70.14
M_4	0.06	99.75	0.19	0.74	91.37	0.44	0.00	1.25	4.43	1.77	0.07	99.37	0.03	0.54	0.00
D_1	0.11	95.25	4.61	0.06	33.81	9.47	0.20	21.14	19.57	15.75	1.20	81.26	0.54	16.93	0.00
D_2	0.01	3.80	96.18	0.00	15.55	0.31	0.11	57.51	1.99	24.53	0.02	17.94	0.22	0.07	81.69
C_1	0.04	2.69	97.27	0.27	5.02	2.37	0.04	28.19	2.02	62.09	0.23	21.00	0.16	2.32	76.25
C_2	0.02	6.60	93.32	0.01	9.40	0.44	0.08	20.57	2.04	67.46	0.22	75.83	2.47	21.08	0.01
C_3	0.01	2.88	97.03	0.04	2.05	1.17	0.00	28.66	1.62	66.46	0.33	16.47	0.16	14.83	67.69
C_4	0.18	2.07	97.74	0.02	0.18	0.08	0.00	1.30	0.09	98.32	0.55	33.57	0.32	3.23	62.00
C_5	—	—	—	0.75	1.92	38.23	0.00	6.43	1.49	51.17	3.89	30.43	0.02	0.35	65.30
W_1	0.04	2.67	97.27	0.07	3.60	1.03	0.02	26.68	1.47	67.13	0.04	20.56	0.16	1.66	77.55
W_2	0.00	7.08	92.89	0.00	11.02	0.22	0.18	39.44	1.72	47.42	0.03	21.78	0.27	0.95	76.89
W_3	0.02	2.18	97.73	0.05	1.63	0.94	0.00	22.26	1.27	73.84	0.41	48.02	3.69	46.37	0.00
W_4	0.01	2.05	97.93	0.00	0.23	0.12	0.00	1.36	0.10	98.19	0.02	33.85	0.51	5.17	60.18

Table 4.5: Analyzing the percent of time required for various procedures in each algorithm. Values in M_RUR refer to sheared systems (whenever it was necessary). A column about Sorting in the case of GRID and G_RUR is not shown.

4.2.3 The effect of filtering

In the following paragraphs we measure the effect of interval arithmetic filters.

grid

Table 4.6 presents running times for GRID solver in cases where no filtering is performed in computations, i.e. all computations rely on Sturm sequences, or all filters have been applied as these were described in section 4.1. The final column speedup indicates the speedup achieved by filters in every case. Based

system	deg		sols	Average Time (msecs)		Speedup
				SLV-GRID		
	f	g		NO FILTERS	FILTERED	
R_1	3	4	2	5	5	1.00
R_2	3	1	1	41	66	0.62
R_3	3	1	1	1	1	1.00
M_1	3	3	4	22	87	0.25
M_2	4	2	3	4	4	1.00
M_3	6	3	5	1,231	803	1.53
M_4	9	10	2	262	218	1.20
D_1	4	5	1	6	6	1.00
D_2	2	2	4	583	667	0.87
C_1	7	6	6	2,601	1,896	1.37
C_2	4	3	6	65	177	0.37
C_3	8	7	13	106	580	0.18
C_4	8	7	17	35,168	5,903	5.98
C_5	16	15	17	$> 20'$	$> 20'$	–
W_1	7	6	9	2,895	2,293	1.26
W_2	4	3	5	514	367	1.40
W_3	8	7	13	104	518	0.20
W_4	8	7	17	35,054	5,410	6.48

Table 4.6: Performance averages over 10 runs in MAPLE 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

on the numbers of the above table, the average speedup achieved by filtering techniques is about 1.51. However, in terms of total computing time for the entire test-set we can observe that:

- Total time without filtering = 78,662 msecs.
- Total time with filtering = 19,001 msecs.

Hence, the speedup achieved for the entire test-set is about 4.14. Note that in both of the above computations system C_5 has been excluded since neither variation of GRID was able to solve the system within 20 minutes. However,

there are indications that filtering techniques help more in other cases, see for example section 4.4.3.

m_rur

The effect of filtering techniques in the case of M_RUR will be discussed in section 4.4.3 where all solvers deal with bivariate systems in generic position.

g_rur

A similar table with that in the case of GRID is table 4.7. This time the average

system	deg		sols	Average Time (msecs)		Speedup
				SLV-G_RUR		
	f	g		NO FILTERS	FILTERED	
R_1	3	4	2	6	5	1.20
R_2	3	1	1	36	36	1.00
R_3	3	1	1	1	1	1.00
M_1	3	3	4	10	10	1.00
M_2	4	2	3	4	4	1.00
M_3	6	3	5	141	110	1.28
M_4	9	10	2	201	210	0.96
D_1	4	5	1	6	6	1.00
D_2	2	2	4	171	128	1.34
C_1	7	6	6	236	222	1.06
C_2	4	3	6	18	18	1.00
C_3	8	7	13	75	75	1.00
C_4	8	7	21*	382	370	1.03
C_5	16	15	17	3,861	3,877	1.00
W_1	7	6	9	277	247	1.12
W_2	4	3	5	141	114	1.23
W_3	8	7	13	24	24	1.00
W_4	8	7	17	318	280	1.13

Table 4.7: Performance averages over 10 runs in MAPLE 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

speedup achieved by filtering is about 1.08. In terms of total computing time for the entire test-set we can observe that:

- Total time without filtering $= 5,908$ msecs.
- Total time with filtering $= 5,737$ msecs.

In other words, the speedup that is achieved by filtering for the entire test-set is about 1.03. Thus G_RUR seems not to be affected at a significant level by filtering. However, this is more or less expected since G_RUR relies heavily on

gcd computations in extension fields and MAPLE's built-in function for factoring. Even when computing the multiplicities of the given system, G_RUR seems not to be affected much from filtering. For a more concrete comparison, please refer to section 4.4.3 that discusses the problem of computing the multiplicities of the given system.

4.3 Bivariate solving and other packages

For the sake of completeness on the evaluation of the initial release of the SLV library tests have been made with other solvers on the same polynomial systems. First of all, FGB/RS [2] [Rou99], which performs exact real solving using Gröbner bases and RUR, through its MAPLE interface has been tested. It should be underlined though that communication with MAPLE increases the runtimes and additional tuning might offer 20-30% efficiency increase. Moreover, 3 SYNAPS [3] solvers have been tested: STURM is a naive implementation of GRID [ET05]; SUBDIV implements [MP05], and is based on Bernstein basis and `double` arithmetic. It needs an initial box for computing the real solutions of the system and in all the cases the box $[-10, 10] \times [-10, 10]$ was used. NEWMAC [MT00], is a general purpose solver based on computations of generalized eigenvectors using LAPACK, which computes all complex solutions.

Other MAPLE implementations have also been tested: INSULATE is a package that implements [WS05] for computing the topology of real algebraic curves, and TOP implements [GVN02]. Both packages were kindly provided by their authors. We tried to modify the packages so as to stop them as soon as they compute the real solutions of the corresponding bivariate system and hence achieve an accurate timing in every case. Finally, it should be noted that TOP has an additional parameter that sets the initial precision (decimal digits). A very low initial precision or a very high one results in inaccuracy or performance loss; but there is no easy way for choosing a good value. Hence, we followed [EKW07] and recorded its performance on initial values of 60 and 500 digits.

It should be underlined that experiments are not considered as competition, but as a crucial step for improving existing software. Moreover, it is very difficult to compare different packages, since in most cases they are made for different needs. In addition, accurate timing in MAPLE is hard, since it is a general purpose package and a lot of overhead is added to its function calls. For example this is the case for FGB/RS.

Overall performance results are shown on tab. 4.8, averaged over 10 iterations. Although the current solver of choice for SLV library is G_RUR, the other solvers are presented as well for completeness. Note that for the first data set, there are no timings for INSULATE and TOP since it was not easy to modify their code so as to deal with general polynomial systems. The rest (systems C_i and W_i) correspond to algebraic curves, i.e. polynomial systems of the form $f = \frac{\partial f}{\partial y} = 0$, that all packages can deal with.

[2] `http://www-spaces.lip6.fr/index.html`
[3] `http://www-sop.inria.fr/galaad/logiciels/synaps/`

In cases where the solvers failed to find the correct number of real solutions we indicate so with an asterisk (*). In the case of NEWMAC where all complex solutions are computed, the (*) is placed in one more case: since NEWMAC computes all complex solutions, a further computing step is required so as to distinguish the ones that reflect the real solutions.

system	deg		solutions	Average Time (msecs)									
				BIVARIATE SOLVING							TOPOLOGY		
				SLV			FGB/RS	SYNAPS			INSULATE	TOP	
	f	g		GRID	M_RUR	G_RUR		STURM	SUBDIV	NEWMAC		60	500
R_1	3	4	2	5	9	5	26	2	2	5*	–	–	–
R_2	3	1	1	66	21	36	24	1	1	1*	–	–	–
R_3	3	1	1	1	2	1	22	1	2	1*	–	–	–
M_1	3	3	4	87	72	10	25	2	1	2*	–	–	–
M_2	4	2	3	4	5	4	24	1	289*	2*	–	–	–
M_3	6	3	5	803	782	110	30	230	5,058*	7*	–	–	–
M_4	9	10	2	218	389	210	158	90	3*	447*	–	–	–
D_1	4	5	1	6	12	6	28	2	5	8*	–	–	–
D_2	2	2	4	667	147	128	26	21	1*	2	–	–	–
C_1	7	6	6	1,896	954	222	93	479	170,265*	39*	524	409	1,367
C_2	4	3	6	177	234	18	27	12	23*	4*	28	36	115
C_3	8	7	13	580	1,815	75	54	23	214*	25*	327	693	2,829
C_4	8	7	17	5,903	80,650	370	138	3,495	217*	190*	1,589	1,624	6,435
C_5	16	15	17	$> 20'$	60,832	3,877	4,044	$> 20'$	6,345*	346*	179,182	91,993	180,917
W_1	7	6	9	2,293	2,115	247	92	954	55,040*	39*	517	419	1,350
W_2	4	3	5	367	283	114	29	20	224*	3*	27	20	60
W_3	8	7	13	518	2,333	24	56	32	285*	25*	309	525	1,588
W_4	8	7	17	5,410	77,207	280	148	4,086	280*	207*	1,579	1,458	4,830

Table 4.8: Performance averages over 10 runs in MAPLE 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

4.3.1 g_rur and other solvers

In the following paragraphs we will try to compare the performance of G_RUR with the rest of the solvers. For this purpose, we conduct speedup-tables like the ones that were drawn in section 4.2.1.

g_rur vs. fgb/rs

Table 4.9 presents running times for FGB/RS and G_RUR as well as the speedup that one gains when choosing G_RUR instead of FGB/RS for bivariate solving. As it is shown from the table G_RUR is faster than FGB/RS in 8 out of the 18

system	Average Time		speedup
	FGB/RS	G_RUR	
R_1	26	5	5.20
R_2	24	36	0.67
R_3	22	1	22.00
M_1	25	10	2.50
M_2	24	4	6.00
M_3	30	110	0.27
M_4	158	210	0.75
D_1	28	6	4.67
D_2	26	128	0.20
C_1	93	222	0.42
C_2	27	18	1.50
C_3	54	75	0.72
C_4	138	370	0.37
C_5	4,044	3,877	1.04
W_1	92	247	0.37
W_2	29	114	0.25
W_3	56	24	2.33
W_4	148	280	0.53

Table 4.9: The performance of FGB/RS and G_RUR on bivariate solving and the speedup that is achieved when choosing G_RUR.

instances, including the difficult system C_5. The speedup factor ranges from 0.2 to 22 with an average of 2.62. However, in terms of total computing times for the entire test-set we can observe that:

- Total time for FGB/RS $= 5,044$ msecs.
- Total time for G_RUR $= 5,737$ msecs.

Hence, the speedup in terms of total computing time is about 0.88. This is an indication that although the computation of the ideal of the given system is a more expensive operation on average, it may be faster when someone faces a set of different polynomial systems.

g_rur vs. synaps/sturm

Let's move on with a comparison between G_RUR and SYNAPS's STURM implementation. Table 4.10 presents running times for SYNAPS/STURM and G_RUR and the speedup gained when preferring G_RUR. G_RUR is faster than STURM

system	Average Time		speedup
	STURM	G_RUR	
R_1	2	5	0.40
R_2	1	36	0.03
R_3	1	1	1.00
M_1	2	10	0.20
M_2	1	4	0.25
M_3	230	110	2.09
M_4	90	210	0.43
D_1	2	6	0.33
D_2	21	128	0.16
C_1	479	222	2.16
C_2	12	18	0.67
C_3	23	75	0.31
C_4	3,495	370	9.45
C_5	$> 20'$	3,877	–
W_1	954	247	3.86
W_2	20	114	0.18
W_3	32	24	1.33
W_4	4,086	280	14.59

Table 4.10: The performance of SYNAPS/STURM and G_RUR on bivariate solving and the speedup that is achieved when choosing G_RUR.

in 6 out of the 18 instances. On the other hand, G_RUR behaves worse usually in polynomial systems that are solved by both implementations in less than 100 msecs, something that is expected since STURM is implemented in C++. However, as the dimension of the polynomial systems increases, G_RUR outperforms STURM and the latter's lack of modular algorithms for computing resultants is more and more evident. Overall, an average speedup of about 2.2 is achieved when someone prefers G_RUR. In terms of total computing times for the entire test-set (excluding system C_5 where STURM failed to reply within 20 minutes) we can observe that:

- Total time for SYNAPS/STURM $= 9,451$ msecs.
- Total time for G_RUR $= 1,860$ msecs.

Hence, if someone considers the speedup that is achieved in terms of total computing time for the entire test set, it can be observed that G_RUR is about 5.08 times faster than STURM highlighting the previous remark regarding resultants in SYNAPS.

g_rur vs. synaps/subdiv

We now switch to a comparison between G_RUR and SYNAPS's SUBDIV implementation. Table 4.11 presents running times for SYNAPS/SUBDIV and G_RUR, and as in the earlier tables, the last column shows the speedup gained when preferring G_RUR. It should be mentioned however, that SUBDIV requires an initial box where all the real solutions of the system reside. In the experiments, the box $[-10, 10] \times [-10, 10]$ was used in every case. The solver was called with the following command:

```
sols = solve( pols, SBDSLV< NT, SBDSLV_RDL >(1e-10), box);
```

where pols are of type Seq< MPOL > and sols are of type Seq< VectDse< NT> >. Finally, in cases where SUBDIV failed to compute the correct number of real solutions, an asterisk (*) is placed to indicate so. G_RUR is faster

system	Average Time		speedup
	SUBDIV	G_RUR	
R_1	2	5	0.40
R_2	1	36	0.03
R_3	2	1	2.00
M_1	1	10	0.10
M_2	289*	4	72.25
M_3	5,058*	110	45.98
M_4	3*	210	0.01
D_1	5	6	0.83
D_2	1*	128	0.01
C_1	170,265*	222	766.96
C_2	23*	18	1.28
C_3	214*	75	2.85
C_4	217*	370	0.59
C_5	6,345*	3,877	1.64
W_1	55,040*	247	222.83
W_2	224*	114	1.96
W_3	285*	24	11.88
W_4	280*	280	1.00

Table 4.11: The performance of SYNAPS/SUBDIV and G_RUR on bivariate solving and the speedup that is achieved when choosing G_RUR.

than SUBDIV in half of the instances. However, the case is similar to STURM's. G_RUR may require more computing time on polynomial systems that are solved in less than 400 msecs by both solvers, while on system C_5 G_RUR is faster than SUBDIV by about 2.47 seconds. A striking experimental result though is SUBDIV's inefficiency on polynomial systems C_1 and W_1. Note that the initial box $[-10, 10] \times [-10, 10]$ is not large enough to justify easily such deficiency. For example, all real solutions of the system C_1 can be found inside the rectangle

$[-2,2] \times [-1,2]$ while the x-coordinates can take both of the extreme values; i.e. -2 and 2. On average, G_RUR achieves a speedup of 62.92 which is the result of the problematic behavior of SUBDIV in systems C_1 and W_1. If these systems are omitted from the computation, then G_RUR achieves a speedup of 8.93. In terms of total computing times for the entire test-set we can observe that:

- Total time for SYNAPS/SUBDIV $= 238,255$ msecs.
- Total time for G_RUR $= 5,737$ msecs.

Hence, the speedup under these terms is about 41.53 favoring G_RUR. However, this value is again greatly increased due to systems C_1 and W_1. Omitting these systems, we can observe that total computing times are as follows:

- Total time for SYNAPS/SUBDIV $= 12,950$ msecs.
- Total time for G_RUR $= 5,268$ msecs.

This time the speedup in terms of total computing time is about 2.46. As a final comment, one can not forget that SUBDIV is based on finite precision arithmetic and consequently numerical errors occur in the computations of the solutions as this is signified by an asterisk (*) in tables 4.8 and 4.11.

g_rur vs. synaps/newmac

A comparison between G_RUR and SYNAPS's NEWMAC implementation can be made with the help of table 4.12 which has similar structure with the previous tables. The solver was called with the following command:

```
sols = solve( pols, Newmac<coeff_t,sol_t>());
```

where `pols` are of type `std::list<MPOL>` and `sols` are of type `sol_t`. Note that an asterisk (*) indicates incorrect number of real solutions. The problem is that NEWMAC computes all *complex* solutions of the input polynomial system and some considerations are needed in these cases. But these will be addressed in the following paragraph. G_RUR is faster than NEWMAC in systems M_4, D_1 and W_3 and exhibits similar performance in systems R_1 and R_3. This time the average speedup is about 0.53 if someone prefers G_RUR, and in terms of total computing times for the entire test-set we have:

- Total time for SYNAPS/NEWMAC $= 1,353$ msecs.
- Total time for G_RUR $= 5,737$ msecs.

In other words, G_RUR is slower than newmac about 4.24 times for the entire test-set.

However, these numbers do not necessarily reflect the truth for various reasons. First of all, NEWMAC is based on computations of generalized eigenvectors using LAPACK, which computes all complex solutions. This can be really fast in

system	Average Time		speedup
	NEWMAC	G_RUR	
R_1	5*	5	1.00
R_2	1*	36	0.03
R_3	1*	1	1.00
M_1	2*	10	0.20
M_2	2*	4	0.50
M_3	7*	110	0.06
M_4	447*	210	2.13
D_1	8*	6	1.33
D_2	2	128	0.02
C_1	39*	222	0.18
C_2	4*	18	0.22
C_3	25*	75	0.33
C_4	190*	370	0.51
C_5	346*	3,877	0.09
W_1	39*	247	0.16
W_2	3*	114	0.03
W_3	25*	24	1.04
W_4	207*	280	0.74

Table 4.12: The performance of SYNAPS/NEWMAC and G_RUR on bivariate solving and the speedup that is achieved when choosing G_RUR.

practice, but an additional problem arises; that of classifying the real solutions among all complex solutions computed by NEWMAC. This is not as trivial as it may sound, since finite precision arithmetic is used, resulting in numerical errors while computing all complex solutions. So, there is one problem on retracting only the real solutions among all complex solutions computed (with the possible numerical errors that these may contain). In addition to that, finite precision has further impacts on the solution set that is computed. There are cases where NEWMAC may not compute some of the real solutions. A representative example in this class of problems is system C_4 which has 17 real solutions and NEWMAC claims that the total number of real and complex solutions is exactly 0.

Hence, NEWMAC requires a better and more accurate implementation than LAPACK when computing the various eigenvectors and eigenvalues that are needed in order to solve the input systems. However, this might still not eliminate all numerical errors that are introduced in the entire complex solution set. Even with this enhancement, some additional time will be possibly required in order to filter the few (in general) real solutions among the entire complex solution set. For instance, NEWMAC computes 90 complex solutions in system M_4 while the number of real solutions for the system is only 2. Concluding, having all these observations in mind, G_RUR seems to be a competitive alternative to NEWMAC since it is not affected by these problems.

g_rur vs. insulate

Let's turn our attention on a comparison between G_RUR and INSULATE which computes the topology of real plane algebraic curves. In this case INSULATE has been modified so as to stop as soon as it computes all real solutions. A comparative performance table similar to the ones in the previous paragraphs is presented in table 4.13. Note that the comparison takes place on the second set of the test-set where the polynomial systems are of the form $f = \frac{\partial f}{\partial y} = 0$ that both packages can manage. G_RUR is faster in all but W_2 system yielding

system	Average Time		speedup
	INSULATE	G_RUR	
C_1	524	222	2.36
C_2	28	18	1.55
C_3	327	75	4.36
C_4	1,589	370	4.29
C_5	179,182	3,877	46.22
W_1	517	247	2.09
W_2	27	114	0.24
W_3	309	24	12.88
W_4	1,579	280	5.64

Table 4.13: The performance of INSULATE and G_RUR on bivariate solving and the speedup that is achieved when choosing G_RUR.

an average speedup this time of 8.85. However, as the dimension of the input polynomial systems increases, G_RUR seems to be more efficient. In terms of total computing time for the entire test set we can observe:

- Total time for INSULATE = 184,082 msecs.
- Total time for G_RUR = 5,227 msecs.

Hence the speedup under this point of view is about 35.22. In any case though, the amount of experiments is relatively small in order to draw safe conclusions on the relative performance of the two implementations in real solving of bivariate polynomial systems.

g_rur vs. top

Finally, a comparison between G_RUR and TOP which computes the topology of real plane algebraic curves is performed. Recall that TOP requires an extra parameter which sets the initial precision in computations (decimal digits). As it has already been stated, this is a problem since there is no easy way on computing a good value and furthermore, a very low initial precision might result in loss in the number of real solutions, while a very high initial precision might result in performance deficiency. For this purpose, the route of [EKW07] has been followed and the performance of TOP was recorded for initial precisions

of 60 and 500 digits. Similarly with INSULATE case, the comparison takes place on the systems C_i and W_i that are of the form $f = \frac{\partial f}{\partial y} = 0$ that both packages can manage.

60 **digits precision:** The comparison in this case is shown in table 4.14. G_RUR is faster in all but W_2 system yielding an average speedup this time of 7.79. Similarly with INSULATE's case, as the dimension of the input polynomial systems increases, G_RUR seems to be more efficient. In terms of total computing time for the entire test set we can observe:

system	Average Time		speedup
	TOP_{60}	G_RUR	
C_1	409	222	1.84
C_2	36	18	3.00
C_3	693	75	9.24
C_4	1,624	370	4.39
C_5	91,993	3,877	23.73
W_1	419	247	1.70
W_2	20	114	0.18
W_3	525	24	21.88
W_4	1,458	280	5.21

Table 4.14: The performance of TOP with precision set to 60 digits and G_RUR on bivariate solving and the speedup that is achieved when choosing G_RUR.

- Total time for $TOP_{60} = 97,177$ msecs.
- Total time for G_RUR $= 5,227$ msecs.

Hence the speedup under this point of view is about 18.59.

500 **digits precision:** The comparison in this case is shown in table 4.15. An interesting result is that although TOP computations have been slowed down with this precision, TOP is still faster in solving system W_2. This time the average speedup that is achieved by G_RUR is 22.64. In terms of total computing time for the entire test set this time we have:

- Total time for $TOP_{500} = 199,491$ msecs.
- Total time for G_RUR $= 5,227$ msecs.

Hence the speedup under this point of view is about 38.17.

4.4 Computing multiplicities

This section presents the performance of SLV library when someone wants to compute the multiplicities at the various intersecting points. Moreover, the effect of filtering will be discussed once more.

system	Average Time		speedup
	TOP_{500}	G_RUR	
C_1	1,367	222	6.16
C_2	115	18	6.39
C_3	2,829	75	37.72
C_4	6,435	370	17.39
C_5	180,917	3,877	46.66
W_1	1,350	247	5.47
W_2	60	114	0.53
W_3	1,588	24	66.17
W_4	4,830	280	17.25

Table 4.15: The performance of TOP with precision set to 500 digits and G_RUR on bivariate solving and the speedup that is achieved when choosing G_RUR.

Overall performance results for the three projection based algorithms are shown on table 4.16. In order to compute the multiplicities the initial systems were sheared whenever it was necessary based on the algorithm that was presented in section 3.1.1. Since the polynomial systems were in generic position, the algorithms stopped searching for solution along the various vertical lines as soon as a solution was computed. Note that running times in M_RUR's case have not changed from table 4.1 since M_RUR by default requires a system in generic position.

Once again G_RUR presents the best performance. It is faster in 17 out of the 18 instances and apart from system C_5 provides solutions in less than a second. Moreover, now that the sheared systems have little or no linear factors and slightly increased bitsize GRID's high complexity starts to become more apparent: M_RUR is faster in 10 out of the 18 instances. In addition to that, it should be stressed once again that M_RUR's inefficiency is basically due to the lack of optimal algorithms for computing the various Sturm sequences. For example, when solving system C_5 M_RUR requires more than 23 seconds simply to generate the StHa sequence of the input polynomials f and g.

4.4.1 Comparing slv solvers

The following paragraphs will briefly compare G_RUR with GRID and M_RUR when computing multiplicities. Moreover, this time a comparison between M_RUR and GRID will be performed.

g_rur vs. grid

Table 4.17 presents running times for GRID and G_RUR when computing multiplicities. Again, the final column indicates the speedup that is achieved when someone prefers G_RUR. As it is shown from the table 4.17 G_RUR can be up to 15.81 times faster than GRID with an average speedup of around 5.26 among

system	deg		$\mathbb{R}_{alg}$	Average Time (msecs)		
	f	g	solutions	GRID	M_RUR	G_RUR
R_1	3	4	2	6	9	6
R_2	3	1	1	66	21	36
R_3	3	1	1	1	2	1
M_1	3	3	4	183	72	45
M_2	4	2	3	4	5	4
M_3	6	3	5	4,871	782	393
M_4	9	10	2	339	389	199
D_1	4	5	1	6	12	6
D_2	2	2	4	567	147	126
C_1	7	6	6	1,702	954	247
C_2	4	3	6	400	234	99
C_3	8	7	13	669	1,815	152
C_4	8	7	17	7,492	80,650	474
C_5	16	15	17	$> 20'$	60,832	6,367
W_1	7	6	9	3,406	2,115	393
W_2	4	3	5	1,008	283	193
W_3	8	7	13	1,769	2,333	230
W_4	8	7	17	5,783	77,207	709

Table 4.16: Performance averages over 10 runs in MAPLE 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

the input systems and excluding system C_5 where GRID failed to reply within 20 minutes. Moreover, in terms of total computing times for the entire test-set (again excluding system C_5) we can observe that:

- Total time for GRID $= 28,272$ msecs.
- Total time for G_RUR $= 3,313$ msecs.

In other words, the speedup in terms of total computing time is about 8.53.

g_rur vs. m_rur

Table 4.18 presents running times for M_RUR and G_RUR when computing multiplicities. Similarly with the previous table, the final column indicates the speedup that is achieved when preferring G_RUR. This time G_RUR can be up to 170.15 times faster than M_RUR with an average speedup of around 18.77 among the input polynomial systems. Moreover, in terms of total computing times for the entire test-set we can observe that:

- Total time for M_RUR $= 227,862$ msecs.
- Total time for G_RUR $= 9,680$ msecs.

In other words, the speedup in terms of total computing time is about 23.54.

system	Average Time		speedup
	GRID	G_RUR	
R_1	6	6	1.00
R_2	66	36	1.83
R_3	1	1	1.00
M_1	183	45	4.07
M_2	4	4	1.00
M_3	4,871	393	12.39
M_4	339	199	1.70
D_1	6	6	1.00
D_2	567	126	4.50
C_1	1,702	247	6.89
C_2	400	99	4.04
C_3	669	152	4.40
C_4	7,492	474	15.81
C_5	$> 20'$	6,367	–
W_1	3,406	393	8.67
W_2	1,008	193	5.22
W_3	1,769	230	7.69
W_4	5,783	709	8.16

Table 4.17: The performance of GRID and G_RUR implementations when computing multiplicities on the intersections and the speedup that is achieved when choosing G_RUR.

m_rur vs. grid

Table 4.19 presents running times for GRID and G_RUR when computing multiplicities. The final column in this table indicates the speedup that is achieved when preferring M_RUR for this operation. Excluding system C_5 where GRID failed to reply within 20 minutes, M_RUR can be up to 6.23 times faster, yielding an average speedup of around 1.71 among the input systems. Moreover, in terms of total computing times for the entire test-set (again excluding system C_5) we can observe that:

- Total time for GRID = 28,272 msecs.
- Total time for G_RUR = 167,030 msecs.

In other words, the speedup in terms of total computing time is about 0.17. However, it should be mentioned once again that system C_5 is not considered in these values.

4.4.2 Decomposing running times

This section is similar to 4.2.2. It presents statistics for the various solvers in the *sheared* case of the test-set polynomial systems. Hence, the interpretation

system	Average Time		speedup
	M_RUR	G_RUR	
R_1	9	6	1.50
R_2	21	36	0.58
R_3	2	1	2.00
M_1	72	45	1.60
M_2	5	4	1.25
M_3	782	393	1.99
M_4	389	199	1.95
D_1	12	6	2.00
D_2	147	126	1.17
C_1	954	247	3.86
C_2	234	99	2.36
C_3	1,815	152	11.94
C_4	80,650	474	170.15
C_5	60,832	6,367	9.55
W_1	2,115	393	5.38
W_2	283	193	1.47
W_3	2,333	230	10.14
W_4	77,207	709	108.90

Table 4.18: The performance of M_RUR and G_RUR implementations when computing multiplicities on the intersections and the speedup that is achieved when choosing G_RUR.

of the two tables is identical to the tables presented in section 4.2.2.

Things have not changed much from section 4.2.2 in GRID's and M_RUR's case. In a nutshell, GRID spends more than 72% of its time in matching. Similarly with table 4.4, this percent includes the application of filters and does not take into account the polynomial system C_5 where GRID failed to reply within 20 minutes. M_RUR spends about 45-50% of its time in matching and about 24-27% in the pre-computation filtering technique. Finally, G_RUR spends 68-80% of its time in matching, including gcd computations in an extension field. This time, in absence of excessive factoring G_RUR spends significantly more time in bivariate solving.

The equivalent table to table 4.5 is table 4.21. It presents the running-time breakdown for the various algorithms in the various cases. Again note, that values presented in M_RUR's case are identical in both tables due to the requirements of the algorithm, i.e. M_RUR has to solve sheared systems.

system	Average Time		speedup
	GRID	M_RUR	
R_1	6	9	0.67
R_2	66	21	3.14
R_3	1	2	0.50
M_1	183	72	2.54
M_2	4	5	0.80
M_3	4,871	782	6.23
M_4	339	389	0.87
D_1	6	12	0.50
D_2	567	147	3.86
C_1	1,702	954	1.78
C_2	400	234	1.71
C_3	669	1,815	0.37
C_4	7,492	80,650	0.09
C_5	$> 20'$	60,832	–
W_1	3,406	2,115	1.61
W_2	1,008	283	3.56
W_3	1,769	2,333	0.76
W_4	5,783	77,207	0.07

Table 4.19: The performance of GRID and M_RUR implementations when computing multiplicities on the intersections and the speedup that is achieved when choosing M_RUR.

	phase of the algorithm	interval		median	mean	std dev
		min	max			
GRID	projections	00.00	00.53	00.06	00.08	00.12
	univ. solving	01.65	99.63	05.42	27.39	37.65
	biv. solving	00.30	98.33	96.75	72.42	37.82
	sorting	00.00	01.15	00.02	00.11	00.27
MRUR	projection	00.00	00.75	00.06	00.14	00.23
	univ. solving	00.18	91.37	15.55	17.47	20.79
	StHa seq.	00.08	38.23	01.17	05.80	09.91
	inter. points	00.00	03.23	00.09	00.32	00.75
	filter x-cand	00.68	72.84	26.68	23.81	21.93
	compute K	00.09	34.37	02.04	07.06	10.21
	biv. solving	01.77	98.32	51.17	45.41	28.71
GRUR	projections	00.02	03.73	00.11	00.58	01.14
	univ. solving	06.60	99.16	22.35	30.27	23.48
	inter. points	00.01	03.93	00.20	00.59	01.05
	rational biv.	00.07	55.59	02.61	11.91	19.22
	$\mathbb{R}_{alg}$ biv.	00.00	93.04	77.51	56.50	35.53
	sorting	00.00	00.83	00.08	00.14	00.21

Table 4.20: Statistics on the performance of SLV's algorithms when computing multiplicities.

System	GRID			M_RUR							G_RUR				
	Projections	**Univariate**	**Bivariate**	**Projection on x-axis**	**Univariate**	**StHa Sequence**	**Interm. Points**	**Filtering on x-axis**	**Compute K**	**FIND (Biv. Sol.)**	**Projections**	**Univariate**	**Interm. Points**	**Rational Bivariate**	**$\mathbb{R}_{alg}$ Bivariate**
R_1	0.11	84.14	15.40	0.06	28.30	17.91	0.64	1.21	19.79	32.09	0.30	45.69	1.08	52.68	0.00
R_2	0.01	4.26	95.73	0.00	16.30	0.61	0.09	72.84	3.50	6.66	0.04	6.60	0.10	0.21	93.04
R_3	0.53	82.86	16.43	0.17	33.04	20.01	0.97	2.79	27.45	15.57	0.57	40.79	2.94	55.59	0.04
M_1	0.00	7.94	92.04	0.05	21.06	1.46	0.14	35.63	2.97	38.69	0.03	25.53	0.40	5.57	68.39
M_2	0.14	60.80	37.92	0.12	32.57	9.49	3.23	0.68	34.37	19.54	3.73	38.23	3.93	53.28	0.00
M_3	0.01	1.66	98.33	0.02	7.39	0.16	0.02	60.60	1.18	30.62	0.02	12.38	0.09	0.31	87.20
M_4	0.06	99.63	0.30	0.74	91.37	0.44	0.00	1.25	4.43	1.77	0.26	99.16	0.03	0.55	0.00
D_1	0.11	95.32	4.54	0.06	33.81	9.47	0.20	21.14	19.57	15.75	1.20	81.22	0.60	16.91	0.00
D_2	0.01	4.13	95.86	0.00	15.55	0.31	0.11	57.51	1.99	24.53	0.02	17.96	0.22	0.07	81.67
C_1	0.09	2.82	97.09	0.27	5.02	2.37	0.04	28.19	2.02	62.09	0.05	17.60	0.15	2.61	79.54
C_2	0.01	5.42	94.54	0.01	9.40	0.44	0.08	20.57	2.04	67.46	0.03	22.35	0.33	2.35	74.40
C_3	0.02	4.71	95.23	0.04	2.05	1.17	0.00	28.66	1.62	66.46	0.06	21.66	0.12	10.70	67.25
C_4	0.18	1.65	98.16	0.02	0.18	0.08	0.00	1.30	0.09	98.32	0.27	26.36	0.11	2.53	70.62
C_5	—	—	—	0.75	1.92	38.23	0.00	6.43	1.49	51.17	3.69	20.07	0.01	0.27	75.95
W_1	0.03	2.16	97.79	0.07	3.60	1.03	0.02	26.68	1.47	67.13	0.11	18.79	0.09	1.64	79.32
W_2	0.00	3.25	96.75	0.00	11.02	0.22	0.18	39.44	1.72	47.42	0.02	16.27	0.14	1.05	82.47
W_3	0.02	1.98	97.98	0.05	1.63	0.94	0.00	22.26	1.27	73.84	0.04	13.55	0.14	6.58	79.57
W_4	0.02	2.86	97.11	0.00	0.23	0.12	0.00	1.36	0.10	98.19	0.09	20.60	0.20	1.54	77.51

Table 4.21: Analyzing the percent of time required for various procedures in each algorithm. All values refer to the sheared systems (whenever it was necessary). A column about Sorting in the case of GRID and G_RUR is not shown.

4.4.3 The effect of filtering

Similarly with section 4.2.3 this section examines the effect of filtering techniques on the performance of all solvers.

grid

Table 4.22 presents running times for GRID solver in cases where no filtering is performed in computations, i.e. all computations rely on Sturm sequences, or all filters have been applied as these were described in section 4.1. The final column speedup indicates the speedup achieved by filters in every case. Based

system	deg		sols	Average Time (msecs)		Speedup
				SLV-GRID		
	f	g		NO FILTERS	FILTERED	
R_1	3	4	2	4	6	0.67
R_2	3	1	1	40	66	0.61
R_3	3	1	1	1	1	1.00
M_1	3	3	4	172	183	0.94
M_2	4	2	3	4	4	1.00
M_3	6	3	5	118,215	4,871	24.27
M_4	9	10	2	404	339	1.19
D_1	4	5	1	6	6	1.00
D_2	2	2	4	418	567	0.74
C_1	7	6	6	5,162	1,702	3.03
C_2	4	3	6	464	400	1.16
C_3	8	7	13	155	669	0.23
C_4	8	7	17	27,126	7,492	3.62
C_5	16	15	17	$> 20'$	$> 20'$	–
W_1	7	6	9	10,091	3,406	2.96
W_2	4	3	5	1,508	1,008	1.50
W_3	8	7	13	1,338	1,769	0.76
W_4	8	7	17	50,808	5,783	8.79

Table 4.22: Performance averages over 10 runs in MAPLE 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

on the numbers of the above table, the average speedup achieved by filtering techniques is about 3.14. However, in terms of total computing time for the entire test-set we can observe that:

- Total time without filtering = 215,916 msecs.
- Total time with filtering = 28,272 msecs.

Hence, the speedup achieved for the entire test-set is about 7.64. Note that in both of the above computations system C_5 has been excluded since neither variation of GRID was able to solve the system within 20 minutes.

m_rur

Table 4.23 presents the performance of the M_RUR solver with the application of all filters or not. Recall, that M_RUR uses one more heuristic technique (refer to section 4.1). This heuristic was present in the running times that are shown in filtered case in table 4.23. M_RUR was unable to solve system C_5 within 20

system	deg		sols	Average Time (msecs)		Speedup
				SLV-M_RUR		
	f	g		NO FILTERS	FILTERED	
R_1	3	4	2	9	9	1.00
R_2	3	1	1	8	21	0.38
R_3	3	1	1	2	2	1.00
M_1	3	3	4	49	72	0.68
M_2	4	2	3	4	5	0.80
M_3	6	3	5	2,054	782	2.63
M_4	9	10	2	323	389	0.83
D_1	4	5	1	10	12	0.83
D_2	2	2	4	88	147	0.60
C_1	7	6	6	22,006	954	23.07
C_2	4	3	6	138	234	0.59
C_3	8	7	13	38,307	1,815	21.11
C_4	8	7	17	784,613	80,650	9.73
C_5	16	15	17	$> 20'$	60,832	–
W_1	7	6	9	45,323	2,115	21.43
W_2	4	3	5	249	283	0.88
W_3	8	7	13	50,724	2,333	21.74
W_4	8	7	17	839,708	77,207	10.88

Table 4.23: Performance averages over 10 runs in MAPLE 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

minutes when filtering techniques were not present in the computations. In the rest of the cases, the average speedup achieved by filtering techniques is about 6.95. In terms of total computing time for the entire test-set (again excluding system C_5 from the computations) we can observe that:

- Total time without filtering $= 1,783,615$ msecs.
- Total time with filtering $= 167,030$ msecs.

Hence, the speedup achieved for the entire test-set is about 10.68.

The effect of preprocessing x-candidates However, it is interesting to investigate the effect of preprocessing x-candidates on M_RUR's performance. For this purpose, table 4.24 presents running times when this heuristic technique is applied or not (but interval arithmetic and gcd filtering are applied) and

system	deg		sols	Average Time (msecs)		Speedup
				SLV-M_RUR		
	f	g		−Preprocess	+Preprocess	
R_1	3	4	2	10	9	1.11
R_2	3	1	1	10	21	0.48
R_3	3	1	1	2	2	1.00
M_1	3	3	4	64	72	0.89
M_2	4	2	3	5	5	1.00
M_3	6	3	5	591	782	0.76
M_4	9	10	2	290	389	0.75
D_1	4	5	1	10	12	0.83
D_2	2	2	4	126	147	0.86
C_1	7	6	6	2,672	954	2.80
C_2	4	3	6	246	234	1.05
C_3	8	7	13	14,276	1,815	7.87
C_4	8	7	17	282,798	80,650	3.51
C_5	16	15	17	$> 20'$	60,832	–
W_1	7	6	9	9,239	2,115	4.37
W_2	4	3	5	354	283	1.25
W_3	8	7	13	13,235	2,333	5.67
W_4	8	7	17	242,199	77,207	3.14

Table 4.24: Performance averages over 10 runs in MAPLE 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

the speedup that is achieved with its application. Hence, the preprocessing heuristic provides M_RUR a speedup of about 2.20 on average. In terms of total computing time for the entire test-set we can observe that:

- Total time without preprocessing = 566,127 msecs.
- Total time with preprocessing = 167,030 msecs.

In other words, the speedup achieved for the entire test-set due to preprocessing is about 3.39. Note that in both of the above computations system C_5 has been excluded since neither variation of GRID was able to solve the system within 20 minutes.

g_rur

Table 4.25 presents the performance of the G_RUR solver with the application of filters or not. Based on the numbers of the above table, the average speedup achieved by filtering techniques is about 1.22. In terms of total computing time for the entire test-set we have:

- Total time without filtering = 12,727 msecs.

system	deg		sols	Average Time (msecs)		Speedup
				SLV-G_RUR		
	f	g		NO FILTERS	FILTERED	
R_1	3	4	2	6	6	1.00
R_2	3	1	1	36	36	1.00
R_3	3	1	1	1	1	1.00
M_1	3	3	4	54	45	1.20
M_2	4	2	3	5	4	1.25
M_3	6	3	5	619	393	1.58
M_4	9	10	2	273	199	1.37
D_1	4	5	1	6	6	1.00
D_2	2	2	4	171	126	1.36
C_1	7	6	6	278	247	1.13
C_2	4	3	6	137	99	1.38
C_3	8	7	13	146	152	0.96
C_4	8	7	17	494	474	1.04
C_5	16	15	17	8,448	6,367	1.33
W_1	7	6	9	482	393	1.23
W_2	4	3	5	297	193	1.54
W_3	8	7	13	296	230	1.29
W_4	8	7	17	978	709	1.38

Table 4.25: Performance averages over 10 runs in MAPLE 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

- Total time with filtering = 9,680 msecs.

Hence, the speedup achieved for the entire test-set is about 1.31. Once again we observe that filtering techniques do not help much G_RUR.

Chapter 5

Conclusion

Concluding, three projection-based algorithms have been proposed for the problem of real solving of bivariate polynomial systems. Two of them, M_RUR and G_RUR, achieve an $\widetilde{\mathcal{O}}_B(N^{12})$ bound which is also the theoretical bound for real solving the projections on x and y axes. Moreover, it is crucial to keep in mind that the input of the problem is of order $\mathcal{O}_B(N^3)$ and the output is $\mathcal{O}_B(N^4)$ since the projection implies univariate polynomials of degree $\mathcal{O}(N^2)$ with coefficients' size bounded by $\mathcal{O}(N^2)$. Hence these algorithms are $\mathcal{O}(I^4)$ and $\mathcal{O}(O^3)$ compared to the input and output respectively. Under this viewpoint, GRID solver's bound $\widetilde{\mathcal{O}}_B(N^{14})$ is closer to the bounds achieved by M_RUR and G_RUR than a plain comparison between 12 and 14 would indicate.

Our solver of choice, as it has already been stated, is G_RUR. Its performance is within a small constant factor with respect to the fastest C and C++ libraries (table 4.8). Of course for an accurate comparison between G_RUR and other solvers one must bear in mind all the comments that were made in section 4.3. The use of Sturm's algorithm and the isolating interval representation for real algebraic numbers guarantee exactness that some solvers can not demonstrate. Moreover, the library SLV as a whole, allows a generic platform in MAPLE where one can work with algorithms that manipulate real algebraic numbers in isolating interval representation. In addition to that, it is easily extensible in higher dimensions due to the intuitive (recursive) projection-based solvers for well-constrained multivariate polynomial systems.

Finally, theorem 2.37 for bivariate sign evaluation signifies the extension of computations on real solving for polynomials with coefficients in an extension field. This was shown in lemmas 3.7, 3.8.

5.1 Future Work

Extending theorem 2.37 to an arbitrary number of variables is of foremost concern. The bounds that will be achieved with this extension have a dual impact on applications. First of all, it will allow the extension of our projection-based solvers of polynomial systems in higher dimension. Secondly, we will be able to determine the sign of a uni- or multi-variate polynomial which has coefficients

in a multiple extension field of $\mathbb{Z}$, assuming that real algebraic numbers are given in isolating interval representation.

Extending algorithm 3 to a higher dimension is straight-forward. Hence SLV library can be easily augmented to cover projection-based solvers for polynomial systems of higher dimension. Moreover, extending theorem 2.37 to an arbitrary number of variables will also allow the computation of real solutions to polynomial systems composed by polynomials with coefficients composed by various algebraic numbers.

Another idea that needs to be reconsidered is that of multipoint evaluation based on Fan-In/Fan-Out techniques that were presented in section 2.1.7. We expect such an application to yield better performance in practice, although our preliminary analysis does not provide better bounds on our solvers than that of iterative Horner.

Finally, of extreme importance is to transfer the SLV library (and the extensions mentioned above) in a fast programming language such as C or C++. However, this project is not as easy as it may sound since optimal (modular) algorithms for core computations should be implemented as well. To highlight the problems that need to be tackled, consider that at the moment there is no open-source modular algorithm for computing the resultant in many variables; and this lies in the heart of our projection-based solvers of polynomial systems.

Appendix A

Test-Bed Polynomials

In what follows the polynomial systems that were used for testing the library are presented. Note that systems W_i differ from the respective C_i only on function g. In the case of the W_i the derivative of f is computed with respect to variable x, while on C_i the derivative is computed with respect to variable y.

A.1 Input Polynomials

System R_1:

$$f = 1 + 2x - 2x^2y - 5xy + x^2 + 3x^2y$$
$$g = 2 + 6x - 6x^2y - 11xy + 4x^2 + 5x^3y$$

System R_2:

$$f = x^3 + 3x^2 + 3x - y^2 + 2y - 2$$
$$g = 2x + y - 3$$

System R_3:

$$f = x^3 - 3x^2 - 3xy + 6x + y^3 - 3y^2 + 6y - 5$$
$$g = x + y - 2$$

System M_1:

$$f = y^2 - x^2 + x^3$$
$$g = y^2 - x^3 + 2x^2 - x$$

System M_2:

$$f = x^4 - 2x^2y + y^2 + y^4 - y^3$$
$$g = y - 2x^2$$

System M_3:

$$f = x^6 + 3x^4y^2 + 3x^2y^4 + y^6 - 4x^2y^2$$
$$g = y^2 - x^2 + x^3$$

System M_4:

$$f = x^9 - y^9 - 1$$
$$g = x^{10} + y^{10} - 1$$

System D_1:

$$f = x^4 - y^4 - 1$$
$$g = x^5 + y^5 - 1$$

System D_2:

$$f = -312960 - 2640x^2 - 4800xy - 2880y^2 + 58080x + 58560y$$
$$g = -584640 - 20880x^2 + 1740xy + 1740y + 274920x - 59160y$$

System C_1:

$$f = (x^3 + x - 1 - xy + 3y - 3y^2 + y^3)$$
$$(x^4 + 2x^2y^2 - 4x^2 - y^2 + y^4)$$
$$g = \mathrm{diff}(f, y)$$

System C_2:

$$f = y^4 - 6y^2x + x^2 - 4x^2y^2 + 24x^3$$
$$g = \mathrm{diff}(f, y)$$

System C_3:

$$f = ((x-1)^2 + y^2 - 2)((x+1)^2 + y^2 - 2)$$
$$((x-1)^2 + (y+2)^2 - 2)((x+1)^2 + (y+2)^2 - 2)$$
$$g = \mathrm{diff}(f, y)$$

System C_4:

$$f = (x^2 - 2x - 1 + y^2)(x^2 + 2x - 1 + y^2)$$
$$(x^2 - 2x + 3 + y^2 + 4y)$$
$$(100000x^2 + 200000x + 299999 + 100000y^2 + 400000y)$$
$$g = \mathrm{diff}(f, y)$$

System C_5:

$$f = (x^4 + 4x^3 + 6x^2 + 4x + y^4 + 4y^3 + 6y^2 + 4y)$$
$$(x^4 + 4x^3 + 6x^2 + 4x + y^4 - 4y^3 + 6y^2 - 4y)$$
$$(x^4 - 4x^3 + 6x^2 - 4x + y^4 + 4y^3 + 6y^2 + 4y)$$
$$(100000x^4 - 400000x^3 + 600000x^2 - 400000x$$
$$-1 + 100000y^4 - 400000y^3 + 600000y^2 - 400000y)$$
$$g = \mathrm{diff}(f, y)$$

System W_1:

$$f = (x^3 + x - 1 - xy + 3y - 3y^2 + y^3)$$
$$(x^4 + 2x^2y^2 - 4x^2 - y^2 + y^4)$$
$$g = \mathrm{diff}(f, x)$$

System W_2:

$$f = y^4 - 6y^2x + x^2 - 4x^2y^2 + 24x^3$$
$$g = \mathrm{diff}(f, x)$$

System W_3:

$$f = ((x-1)^2 + y^2 - 2)((x+1)^2 + y^2 - 2)$$
$$((x-1)^2 + (y+2)^2 - 2)((x+1)^2 + (y+2)^2 - 2)$$
$$g = \mathrm{diff}(f, x)$$

System W_4:

$$f = (x^2 - 2x - 1 + y^2)(x^2 + 2x - 1 + y^2)$$
$$(x^2 - 2x + 3 + y^2 + 4y)$$
$$(100000x^2 + 200000x + 299999 + 100000y^2 + 400000y)$$
$$g = \mathrm{diff}(f, x)$$

System W_5:

$$\begin{aligned} f = (&x^4 + 4x^3 + 6x^2 + 4x + y^4 + 4y^3 + 6y^2 + 4y) \\ &(x^4 + 4x^3 + 6x^2 + 4x + y^4 - 4y^3 + 6y^2 - 4y) \\ &(x^4 - 4x^3 + 6x^2 - 4x + y^4 + 4y^3 + 6y^2 + 4y) \\ &(100000x^4 - 400000x^3 + 600000x^2 - 400000x \\ &-1 + 100000y^4 - 400000y^3 + 600000y^2 - 400000y) \\ g &= \mathrm{diff}(f, x) \end{aligned}$$

Appendix B

Sample Usage

For a more up-to-date coverage of the capabilities of the SLV library the reader is urged to visit `http://www.di.uoa.gr/~erga/soft/SLV_index.html` which is the official homepage of the library. SLV library requires a definition for variable `LIBPATH` which should point on the appropriate path where the source code is stored in your system. On the following, we assume that SLV is located under `/opt/AlgebraicLibs/SLV/`. The following is an example for univariate solving:

```
> LIBPATH := "/opt/AlgebraicLibs/SLV/":
> read cat ( LIBPATH, "system.mpl" ):
> f := 3*x^3 - x^2 - 6*x + 2:
> sols := SLV:-solveUnivariate( f ):
> SLV:-display_1 ( sols );
  < x^2-2, [ -93/64, -45/32], -1.414213568 >
  < 3*x-1, [ 1/3, 1/3], 1/3 >
  < x^2-2, [ 45/32, 93/64], 1.414213568 >
```

Note, that the multiplicities of the roots do not appear, although they have been computed. Instead, the third argument of each component in the *printed* list is an approximation of the root. However, whenever possible we provide rational representation of the root.

The following is an example for bivariate solving, where the second root lies in $\mathbb{Z}^2$:

```
> LIBPATH := "/opt/AlgebraicLibs/SLV/":
> read cat ( LIBPATH, "system.mpl" ):
> f := 1+2*x+x^2*y-5*x*y+x^2:
> g := 2*x+y-3:
> bivsols := SLV:-solveGRID ( f, g ):
> SLV:-display_2 ( bivsols );
  < 2*x^2-12*x+1, [ 3, 7], 5.915475965 > ,
  < x^2+6*x-25, [ -2263/256, -35/4], -8.830718995 >

  < x-1, [ 1, 1], 1 > , < x-1, [ 1, 1], 1 >

  < 2*x^2-12*x+1, [ 3/64, 3/32], .8452400565e-1 > ,
  < x^2+6*x-25, [ 23179/8192, 2899/1024], 2.830943108 >
```

Again, just like in the case of univariate solving, the third argument that is printed on the component that describes each algebraic number is an approximation of the number and not the multiplicity of the root. Similarly, one could have used one of the other solvers on the above example by referring to their names, i.e. call the solvers with one of the following commands:

```
> bivsols := SLV:-solveMRUR ( f, g ):
> bivsols := SLV:-solveGRUR ( f, g ):
```

For those interested in the numerical values or rough approximations of the solutions one can get the appropriate output via `display_float_1` and `display_float_2` procedures. Hence, for the above examples we have:

```
> SLV:-display_float_1 ( sols );
  < -1.4142136 >
  <  0.3333333 >
  <  1.4142136 >
> SLV:-display_float_2 ( bivsols );
  [   5.9154759, -8.8309519, ]
  [   1.0000000,  1.0000000, ]
  [   0.0845241,  2.8309519, ]
```

Consider the list `sols` of $\mathbb{R}_{alg}$ numbers that was returned in the univariate case above; the following are examples on the usage of the `signAt` function provided by our Filtered Kernel[1]:

```
> FK:-signAt( 2*x + 3, sols[1] );
                                1
> FK:-signAt( x^2*y + 2, sols[3], sols[1] );
                               -1
```

Our class on Polynomial Remainder Sequences[2] exports functions allowing the computation of Subresultant and Sturm-Habicht sequences. Let $f, g \in \mathbb{Z}[x, y]$, then you can use *any* of the following commands in order to compute the desired PRS:

```
L := PRS:-StHa ( f, g, y ):
L := PRS:-StHaByDet ( f, g, y ):
L := PRS:-subresPRS ( f, g, y ):
L := PRS:-SubResByDet ( f, g, y ):
```

`PrintPRS` is used for viewing the PRS. For example, let f, g be those from the example on Bivariate Solving above:

```
> L := PRS:-subresPRS ( f, g, y ):
> PRS:-PrintPRS( L );
                 / 2      \                2
                 \x  - 5 x/ y + 1 + 2 x + x
                        y + 2 x - 3
                     3       2
                   2 x  - 14 x  + 13 x - 1
```

Finally, the variance of the above sequence evaluated at $(1, 0)$ can be computed by:

```
> G := PRS:-Eval ( L, 1, 0 );
                        G := [4, -1, 0]
> PRS:-var( G );
                               1
```

[1] Located in file: `FK.mpl`
[2] Located in file: `PRS.mpl`

Bibliography

[Abb06] J. Abbott. Quadratic interval refinement for real roots. In *ISSAC 2006, poster presentation*, 2006. http://www.dima.unige.it/ abbott/.

[AM88] D. Arnon and S. McCallum. A polynomial time algorithm for the topological type of a real algebraic curve. *JSC*, 5:213–236, 1988.

[BK86] E. Brieskorn and H. Knörrer. *Plane Algebraic Curves*. Birkhäuser, Basel, 1986.

[BPM06] S. Basu, R. Pollack, and M-F.Roy. *Algorithms in Real Algebraic Geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer-Verlag, 2nd edition, 2006.

[Can87] J. Canny. *The Complexity of Robot Motion Planning*. ACM – MIT Press Doctoral Dissertation Award Series. MIT Press, Cambridge, MA, 1987.

[Can88] J. Canny. Some algebraic and geometric computations in PSPACE. In *Proc. STOC*, pages 460–467, 1988.

[CFPR06] Frédéric Cazals, Jean-Charles Faugère, Marc Pouget, and Fabrice Rouillier. The implicit structure of ridges of a smooth parametric surface. *Comput. Aided Geom. Des.*, 23(7):582–598, 2006.

[DET07a] D. I. Diochnos, I. Z. Emiris, and E. P. Tsigaridas. On the complexity of real solving bivariate systems. In *International Symposium on Symbolic and Algebraic Computation*, 2007.

[DET07b] D. I. Diochnos, I. Z. Emiris, and E. P. Tsigaridas. On the complexity of real solving bivariate systems. Research Report 6116, INRIA, 02 2007. https://hal.inria.fr/inria-00129309.

[DSY05] Z. Du, V. Sharma, and C. K. Yap. Amortized bound for root isolation via Sturm sequences. In D. Wang and L. Zhi, editors, *Int. Workshop on Symbolic Numeric Computing*, pages 81–93, Beijing, China, 2005.

[EKK+05] A. Eigenwillig, L. Kettner, W. Krandick, K. Mehlhorn, S. Schmitt, and N. Wolpert. A descartes algorithm for polynomials with bitstream coefficients, 2005.

[EKW07] A. Eigenwillig, M. Kerber, and N. Wolpert. Fast and exact geometric analysis of real algebraic plane curves. In *International Symposium on Symbolic and Algebraic Computation*, 2007.

[Emi95] I.Z. Emiris. A general solver based on sparse resultants, March 1995. Available also as Tech. Report 3110, INRIA Sophia-Antipolis, Jan. 1997.

[EMT07] I. Z. Emiris, B. Mourrain, and E. P. Tsigaridas. Real Algebraic Numbers: Complexity Analysis and Experimentation. In P. Hertling, C. Hoffmann, W. Luther, and N. Revol, editors, *Reliable Implementations of Real Number Algorithms: Theory and Practice*, LNCS (to appear). Springer Verlag, 2007. also available in www.inria.fr/rrrt/rr-5897.html.

[EP99] I.Z. Emiris and V.Y. Pan. Applications of FFT. In M.J. Atallah, editor, *Handbook of Algorithms and Theory of Computation*, chapter 17. CRC Press, Boca Raton, Florida, 1999.

[ESY06] A. Eigenwillig, V. Sharma, and C. K. Yap. Almost tight recursion tree bounds for the descartes method. In *ISSAC*, pages 71–78, New York, NY, USA, 2006. ACM Press.

[ET05] I. Z. Emiris and E. P. Tsigaridas. Real solving of bivariate polynomial systems. In V. Ganzha and E. Mayr, editors, *Proc. Computer Algebra in Scientific Computing (CASC)*, volume 3718 of *LNCS*, pages 150–161. Springer, 2005.

[EV99] I.Z. Emiris and J. Verschelde. How to count efficiently all affine roots of a polynomial system. *Discrete Applied Math., Special Issue on Comput. Geom.*, 93(1):21–32, 1999.

[GVEK96] L. González-Vega and M. El Kahoui. An improved upper complexity bound for the topology computation of a real algebraic plane curve. *J. Complexity*, 12(4):527–544, 1996.

[GVLRR89] L. González-Vega, H. Lombardi, T. Recio, and M-F. Roy. Sturm-Habicht Sequence. In *ISSAC*, pages 136–146, 1989.

[GVN02] L. Gonzalez-Vega and I. Necula. Efficient topology determination of implicitly defined algebraic plane curves. *Computer Aided Geometric Design*, 19(9):719–743, December 2002.

[Klo95] J. Klose. Binary segmentation for multivariate polynomials. *J. Complexity*, 11(3):330–343, 1995.

[KSP05] K.H. Ko, T. Sakkalis, and N.M. Patrikalakis. Resolution of multiple roots of nonlinear polynomial systems. *Int. J. of Shape Modeling*, 11(1):121–147, 2005.

[LL91] Y.N. Lakshman and D. Lazard. On the complexity of zero-dimensional algebraic systems. In T. Mora and C. Traverso, editors, *Effective Methods in Algebraic Geometry*, volume 94 of *Progress in Mathematics*, pages 217–225, Boston, 1991. Birkhäuser. (Proc. MEGA '90, Livorno, Italy).

[LR01] T. Lickteig and M-F. Roy. Sylvester-Habicht Sequences and Fast Cauchy Index Computation. *JSC*, 31(3):315–341, 2001.

[LRSED00] H. Lombardi, M-F. Roy, and M. Safey El Din. New Structure Theorem for Subresultants. *JSC*, 29(4-5):663–689, 2000.

[Mil92] P.S. Milne. On the solution of a set of polynomial equations. In B. Donald, D. Kapur, and J. Mundy, editors, *Symbolic and Numerical Computation for Artificial Intelligence*, pages 89–102. Academic Press, 1992.

[Mou96] B. Mourrain. Enumeration problems in geometry, robotics and vision. In L. Gonzalez-Vega and T. Recio, editors, *Effective Methods in Algebraic Geometry*, Progress in Mathematics. Birkhäuser, 1996. (Proc. MEGA '94, Santander, Spain).

[Mou99] B. Mourrain. A new criterion for normal form algorithms. In M. Fossorier, H. Imai, Shu Lin, and A. Poli, editors, *Proc. AAECC*, volume 1719 of *LNCS*, pages 430–443, 1999.

[MP97] B. Mourrain and V.Y. Pan. Solving special polynomial systems by using structured matrices and algebraic residues. In F. Cucker and M. Shub, editors, *Proc. Workshop on Foundations of Computational Mathematics*, pages 287–304, Berlin, 1997. Springer-Verlag.

[MP98] B. Mourrain and V.Y. Pan. Asymptotic acceleration of solving polynomial systems. In *Proc. ACM Symp. Theory of Computing*, pages 488–496. ACM Press, New York, 1998.

[MP05] B. Mourrain and J-P. Pavone. Subdivision methods for solving polynomial equations. Technical Report RR-5658, INRIA Sophia-Antipolis, 2005.

[MPS+06] Bernard Mourrain, Sylvain Pion, Susanne Schmitt, Jean-Pierre Técourt, Elias Tsigaridas, and Nicola Wolpert. Algebraic issues in computational geometry. In Jean-Daniel Boissonnat and Monique Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, pages 117–155. Springer-Verlag, 2006.

[MS99] M. Mignotte and D. Stefanescu. *Polynomials: An algorithmic approach.* Springer, 1999.

[MT00] B. Mourrain and P. Trébuchet. Solving projective complete intersection faster. In C. Traverso, editor, *Proc. Intern. Symp. on*

Symbolic and Algebraic Computation, pages 231–238. New-York, ACM Press., 2000.

[Neu90] Arnold Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, 1990.

[Pan02] V.Y. Pan. Univariate polynomials: Nearly optimal algorithms for numerical factorization and rootfinding. *JSC*, 33(5):701–733, 2002.

[PRS93] P. Pedersen, M-F. Roy, and A. Szpirglas. Counting real zeros in the multivariate case. In F. Eyssette and A. Galligo, editors, *Computational Algebraic Geometry*, volume 109 of *Progress in Mathematics*, pages 203–224. Birkhäuser, Boston, 1993.

[Rei97] D. Reischert. Asymptotically fast computation of subresultants. In *ISSAC*, pages 233–240, 1997.

[Ren89] J. Renegar. On the worst-case arithmetic complexity of approximating zeros of systems of polynomials. *SIAM J. Computing*, 18:350–370, 1989.

[Rou99] F. Rouillier. Solving zero-dimensional systems through the rational univariate representation. *J. of AAECC*, 9(5):433–461, 1999.

[Sak89] T. Sakkalis. Signs of algebraic numbers. *Computers and Mathematics*, pages 131–134, 1989.

[SF90] T. Sakkalis and R. Farouki. Singular points of algebraic curves. *JSC*, 9(4):405–421, 1990.

[TE06] E. P. Tsigaridas and I. Z. Emiris. Univariate polynomial real root isolation: Continued fractions revisited. In Y. Azar and T. Erlebach, editors, *In Proc. 14th ESA*, volume 4168 of *LNCS*, pages 817–828, Zurich, Switzerland, 2006. Springer Verlag.

[Tsi06] Elias P. Tsigaridas. *Algebraic Algorithms and Applications to Geometry*. PhD thesis, Dept. of Informatics and Telecommunications, Univarsity of Athens, 2006.

[vHM02] M. van Hoeij and M. Monagan. A modular GCD algorithm over number fields presented with multiple extensions. In *ISSAC*, pages 109–116, July 2002.

[vzGG03] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge Univ. Press, Cambridge, U.K., 2nd edition, 2003.

[vzGL03] J. von zur Gathen and T. Lücking. Subresultants revisited. *TCS*, 1-3(297):199–239, 2003.

[Wol02] N. Wolpert. *An Exact and Efficient Approach for Computing a Cell in an Arrangement of Quadrics*. PhD thesis, MPI fuer Informatik, October 2002.

[WS05] N. Wolpert and R. Seidel. On the Exact Computation of the Topology of Real Algebraic Curves. In *SoCG*, pages 107–115. ACM, 2005.

[Yap00] C.K. Yap. *Fundamental Problems of Algorithmic Algebra*. Oxford University Press, New York, 2000.