A Toolkit for Non-linear Algebra

John Canny
University of California
Berkeley, CA, 94720

August 16, 1993

Abstract

This report describes an implementation, now in progress, of a toolkit for polynomial algebra. The toolkit is being written in C, and it can solve systems of equations over the complex numbers, and inequalities over the reals. There are many applications of such a toolkit, since problems from many branches of science and engineering can be formulated using systems of polynomial equations and inequalities. By several well-known theorems, in particular Tarski's on the decidability of the theory of the reals, it is possible to solve such problems in principle. But the worst case bounds are exponential or doubly-exponential in the number of variables, and no practical system has appeared that can deal with large problems. On the other hand, systems of polynomials may have a special structure, a kind of sparseness, that implies a complexity (measured by the algebraic degree) much lower than the worst case. Indeed, most of the applications we have studied are of this type. Very recently, algorithms have been developed by the author and others that can exploit this structure. This realization is a strong motivation for developing the toolkit at this time. In addition, there have been improvements in algorithms for sign determination and symbolic-numeric computation, and we feel that these methods are now advanced enough to warrant implementation.

1 Introduction

In this report, we describe a collection of techniques which improve the efficiency of solving systems of polynomial equations and inequalities. Together, these techniques are being implemented in an algebraic-geometric toolkit, written in C. The system computes all the solutions, and provides either a symbolic description of them, or numerical approximations to real or complex solutions. The impetus for the development of the toolkit was the realization that (a) Many practical algebraic problems that seem hard are in fact easy, i.e. have low effective degree (b) Algebraic algorithms have improved over the last few years to the point where we can achieve an overall complexity which is low-order polynomial in this effective degree.

The equation-solving problems we are most interested in come from robotics, computer vision, and geometric modeling. The equations are often polynomial because they arise from:

- Representing orientation of objects in 3D. Both quaternions and rotation matrix coefficients are polynomial descriptions.
- The most popular descriptions of smooth shapes are as tensor product surfaces, or as CSG models, both of which are algebraic surfaces.

- Geometric constraints, like contact, colinearity, distance, and lower pairs, are algebraic.

The theory we use to exploit the low effective degree of polynomial systems is called *sparse elimination theory.* In the late 1980's, Gel'fand and his colleagues began the study of discriminants and resultants of sparse polynomial systems. Sparseness leads to a lowering of effective degree, and the sparse theory provides a simple direct method for proving bounds on the number of solutions. Sparseness can also be exploited to speedup equation solving and elimination of variables. Algorithms to do this have appeared in the last year. An efficient homotopy algorithm for sparse systems was described in [HS92]. The first efficient algorithms for the sparse resultant were described by the author and a collaborator in [CE93], [EC93].

Development in symbolic computation on polynomials has a longer history. Since the work of Collins in 1975, and Schwartz and Sharir and Grigor'ev et al., in the early 1980s, there has been steadily increasing interest in algorithms for the first-order theory of the reals. Formulae in this language have real quantified variables, and predicates which are boolean functions of polynomial inequalities. Of particular interest to us are formulae with existential quantification only. Some readers may have seen the term *constraint satisfaction* applied to this family of problems. The theory of the reals is very powerful because it allows declarative description of an object via constraints, and leaves it to the decision algorithm to find an instance of the object.

Much recent work [HRS90], [Ren92] [GV92] has focussed on improving the asymptotic complexity bounds for the theory of the reals, but unfortunately, most of it ignores the complexity in practice. The work of the author [Can88a], [Can91b], [Can91a], [Can93b], [Can93a] on the other hand, has been specifically directed at practical algorithms. The collection of techniques developed in those papers permits symbolic calculation (i.e. exact calculation, even with singular inputs) whose complexity is polynomial in the effective degree mentioned earlier. The main ideas of those techniques will be described in this report. In addition, we describe a new technique for performing arithmetic operations on very large integers with expected constant cost.

The report is arranged as follows: The theory of sparse systems is at the heart of both the symbolic and numerical toolkits. So we begin in section 2 with a short introduction to this theory. Then in section 3 we give an overview of the numerical kit. The symbolic kit, which is considerably larger, occupies the bulk of the paper in section 4. Finally, in section 5 we describe some future projects.

2 Sparse Systems

In a moment we will be able to say precisely what we mean by a sparse polynomial system. But even before that, we should answer the question: Why study sparse systems? For us, the reason is that sparse systems are ubiquitous in robotics, vision and modeling. This is not a theorem that we can prove, but our experience shows that almost all the systems that arise there are sparse. We can exploit this sparseness in two ways: (i) To prove tight or tighter bounds for the number of solutions and (ii) To compute those solutions in a time that depends on the sparse bounds, not on the classical degree bound (Bezout's bound) which is usually much larger.

Typical examples of sparse systems are those that describe the inverse kinematics for a 6R robot [MC92b], forward kinematics for the Stewart platform [Mer92], camera motion from point matches [FM90], and geometric constraints describing two- or three-dimensional objects [Owe91]. As the dimension of the problem increases, the difference between the sparse and non-sparse bounds

increases dramatically. Very few algebraic problems with more than 4 variables can be solved with classical resultant methods, which ignore sparseness, but many practical problems which are much larger can be solved fast using a custom elimination formula.

As an example, we can take the 6R inverse kinematics problem from robotics. Given a robot with 6 rotational joints, and a placement of the gripper, this problem asks to find the six joint angles $\theta_1, \ldots, \theta_6$ that place the gripper in the wanted pose. The problem can be stated in terms of 4×4 matrices, which can represent any rigid transformation in 3D. We get

$$T_1(\theta_1) \cdots T_6(\theta_6) = T_e$$

where each T_i represents the transformation between links caused by rotation of joint i, and T_e is the transformation of the gripper. It is better to use a parametrization in terms of $t_i = \tan(\theta_i/2)$ rather than θ_i directly. Both sine and cosine are rational functions of t_i, and the matrix $T_i(t_i)$ then contains only rational functions of t_i. A little algebra shows that the inverse $T_i^{-1}(t_i)$ also contains only rational entries.

Now we have $T_1(t_1) \cdots T_6(t_6) = T_e$. The RHS matrix has 16 entries, so we have a system of 16 equations to be satisfied. But there are only 6 variables, and only 6 of the equations can be independent. Suppose we choose such an independent set. The matrix entries have degree 2 in the t_i's so the total degree of each equation is 12. We can reduce this degree by moving some of the joint transformations to the RHS:

$$T_1(t_1)T_2(t_2)T_3(t_3) = T_e T_6^{-1}(t_6)T_5^{-1}(t_5)T_4^{-1}(t_4) \tag{1}$$

From this we can choose 6 equations of degree 6. By Bezout's theorem, the number of solutions of such a system is $6^6 = 46656$. That is, we might have 46,000 tuples of angles $(\theta_1, \ldots, \theta_6)$ which satisfy the gripper pose constraint. This would be a very difficult problem to solve, if there really were this many solutions. But it has been shown that there are only 16, and [LL88] and [RR89] gave constructive proofs. A real-time solver based on [RR89] is described in [MC92b].

A large gap between the Bezout bound and the actual number of solutions is not unusual for geometric problems, although it is not always as dramatic as for inverse kinematics. Bezout's theorem gives an exact count of the number of solutions in projective space, so most of these solutions are at infinity. The problem with trying to solve a system like this is that most methods have a complexity that depends on the Bezout bound. The Bezout bound is exact if all the coefficients of a polynomial system of some given degree are *generic*. Genericity is the requirement that the coefficients do not satisfy a set of algebraic relations. For example, random coefficients would be generic with high probability. But systems that arise in inverse kinematics and other geometric problems are not generic, even if the robot design parameters were. For inverse kinematics, it is easy to see why. The long series of matrix multiplications leads to many common subexpressions. Each T_i matrix is determined by 4 parameters (called Denavit-Hartenberg parameters [SV89]), and with 6 joints, the whole robot is described by 24 parameters. The gripper has 6 degrees of freedom, so the T_e matrix depends on 6 parameters. Each inverse kinematics system is determined by $24 + 6 = 30$ parameters. But the 6×6 system of equations that we actually solve has hundreds of coefficients, all determined by those 30 parameters. Clearly the coefficients are strongly dependent.

The methods in this report exploit the relation between solution count and *sparseness of the equations*. A polynomial is sparse if many of its coefficients, compared to a generic polynomial of that degree, are zero. A bound derived from the set of non-zero coefficients is called a Bernstein

bound. Bernstein showed that his bound is exact if all the coefficients of the polynomial system are generic [Ber75]. In fact they are exact under much weaker assumptions: Only the coefficients of terms that lie at the vertices of the Newton polytopes (defined later in this section) need to be generic for the bounds to be exact [CR91]. So if we can write down a system in a form where these coefficients are independent, we know how to correctly count the solutions. Using sparse homotopy and resultant methods, we know also how to compute the solutions in a running time that depends on their number.

In contrast with sparse methods, which have appeared in the last year, most equation-solving approaches do not exploit the paucity of solutions, and instead have a complexity that depends on the Bezout bound. This has been true both of homotopy methods [TM85] and general homogenous resultants [Mac02]. Some progress has been made in homotopy methods in the last few years, [MS87] and [VC92]. These methods take advantage of some but not all forms of sparseness. Gröbner basis algorithms have a complexity that depends on the effective degree, and so they work well on systems with few roots. This is one reason they have been considered seriously as a practical equation-solving tool. But they also have high overhead, require arbitrary precision integer arithmetic to work over $\mathbb{R}$ or $\mathbb{C}$, and are difficult to parallelize. When their complexity is measured as a function of the number of solutions, their performance is poor. This has been clear for specific systems for which a sparse resultant was already known. An example is the Dixon resultant [Dix08] for tensor product surface implicitization [MC92a]. The solution using resultants can be computed in 1/100 the time of a Gröbner solution [Hof90].

The value of the special purpose resultants like Dixon's has been clear for some time. Certainly, one would like an analogue for polynomials with any given structure (the set of exponents appearing in the polynomials). These general resultants were defined first by Gel'fond et al., as a special case of an *A-discriminant* [GKZ90]. We will term them *sparse resultants.* A *Poisson formula* for the sparse resultant was given in [PS91]. The Poisson formula expresses the resultant in terms of symmetric functions, and makes it easy to find a sparse resultant's degree.

The most convenient description of a resultant is using a *Sylvester formula.* The Sylvester formula expresses the resultant as the determinant of a matrix whose elements are the polynomial coefficients or zero. Sylvester formulae were given for a class of multi-homogeneous systems in [SZ93]. In general, sparse resultants cannot be computed via a Sylvester formula, but they all have a *determinantal formula*, which expresses them as a factor of a matrix determinant. The determinantal formula is important for several reasons. Firstly, it is efficient. The matrix size is small and under reasonable assumptions, polynomial in the sparse resultant degree. Secondly, as we describe in section 3, it allows us to solve non-linear equations using linear algebra tools, such as eigenvalue and characteristic polynomial routines. Thirdly, linear algebra algorithms are easily parallelizable, and we inherit this property when we transform from non-linear to linear with a determinantal formula. The first determinantal formulae for the sparse resultant were given by the author and a collaborator in [CE93] and [EC93]. We will refer to those later in this section.

Sparse equation solving is still a developing field, and sparse methods are not the full answer to exploiting low solution count. For systems with non-generic coefficients, the Bernstein bounds may still be poor. Returning to the inverse kinematics problem, recall that the matrix product gives us 6 equations of degree 6. These polynomials have only 53 non-zero coefficients, whereas a general polynomial of degree 6 in 6 variables would have 1716 coefficients. The Bernstein bound for the kinematics system written in this form is 2,304. Much less than Bezout at 46,000 but still

excessive. But the Bernstein bound drops rapidly if we rewrite the equations in a form where there are fewer dependencies. For example, using some of the equations of [RR89] gives a Bernstein bound of 384. When the coefficient relations come from common subexpressions, there is a systematic way to remove them by introducing new variables. An example of this is given later in section 5.1. Since the kinematics equations do contain many common subexpressions, this method should give us a system with Bernstein bound close to 16. We expect to put a lot of future effort into dealing with coefficient dependencies, and some preliminary ideas are given in section 5.1.

2.0.1 Definitions

Suppose we are given m polynomials $f_1, \ldots, f_m$ in $x_1, \ldots, x_n$ with complex coefficients. We use x^e to denote the monomial $x_1^{e_1} \cdots x_n^{e_n}$, where $e = (e_1, \ldots, e_n) \in \mathbf{Z}^n$ is a multi-exponent. Let $\mathcal{A}_i = \{a_{i1}, \ldots, a_{im_i}\} \subseteq \mathbf{Z}^n$ denote the set of exponents occurring in f_i, then

$$f_i = \sum_{j=1}^{m_i} c_{ij} x^{a_{ij}} \ , \qquad \text{for } i = 1, \ldots, m \ , \tag{2}$$

and we suppose $c_{ij} \neq 0$ so that $\mathcal{A}_i$ is uniquely defined given f_i. We term the study of such systems *sparse elimination theory* because we consider the actual set of exponents $\mathcal{A}_i$ occuring in f_i rather than just the degree of f_i.

One unusual aspect of the theory of sparse systems is that we specifically discount solutions having a coordinate $x_i = 0$. That is, we count only solutions $x = \xi$ with $\xi \in (\mathbf{C}^*)^n$, where $\mathbf{C}^* = \mathbf{C} - \{0\}$. This point often causes confusion to readers seeing it for the first time. There are two natural questions to ask: (i) Why not count all the affine solutions rather than those in $(\mathbf{C}^*)^n$? (ii) Some genuine solutions having some $\xi_i = 0$ will be missed, how can they be recoved?

In answer to the first question: The most natural space to consider for the solutions of a polynomial system is the projective space $\mathbf{PC}^n$, in which Bezout's theorem holds exactly. This space has coordinates $(x_0, \ldots, x_n)$ with scalar multiples identified, so $x \equiv \lambda x$ for all $\lambda \in \mathbf{C}^*$. The affine space $\mathbf{C}^n$, which is the one we are interested in in most applications of algebraic geometry, is obtained from $\mathbf{PC}^n$ by removing the plane at infinity $x_0 = 0$. But this is not a "natural" space and Bezout's theorem appears only in a very weakened form. But if we remove all the coordinate planes $x_i = 0$ from $\mathbf{PC}^n$, we obtain again a space with an exact degree theorem, this time Bernstein's theorem, which we state later in this section. Because of the removal of zero from the solution space, we can consider the more general case of f_i's which are polynomials in the x_i and their reciprocals, the *Laurent* polynomials $\mathbf{C}[x_1, x_1^{-1}, \ldots, x_n, x_n^{-1}]$.

The second question was how to recover the missing solutions having $x_i = 0$. This is a straightforward extension. We simply set $x_i = 0$ in the system of polynomials f, giving us a new system $f|_{x_i=0}$, and apply Bernstein's theorem to this system. We can do this for each $i = 1, \ldots, n$. Let the total number of roots found this way be N_1. There is a possibility of counting the same root twice, so we must also count roots where both $x_i = 0$ and $x_j = 0$ for each pair i, j. Let this number of roots be N_2. N_i is defined similarly by considering all i-tuples of polynomials. Finally, let N_0 be the number of roots of $f = 0$. Then applying the inclusion/exclusion principle, if N is the total number of roots of the system assuming generic coefficients,

$$N = N_0 + N_1 - N_2 + N_3 + \cdots$$

Note that the dimension of the Newton polytopes is at most $n-1$ when we set $x_i = 0$. Because of this, unless one of the f_j's is zero after the specialization (meaning that it was divisible by x_i), the specialized system $f|_{x_i=0}$ will be overconstrained. So unless some f_j is divisible by x_i, the system generically will have no roots with $x_i = 0$.

In short, unless some of the polynomials in a system $f = 0$ are divisible by an x_i, all the affine roots will generically have non-zero coordinates. This is the more common situation in practice, and in this case the Bernstein bounds already count all the affine roots.

2.0.2 Newton Polytopes and Bernstein's Theorem

Definition 2.1 *The finite set $\mathcal{A}_i \subset \mathbb{Z}^n$ of all monomial exponents appearing in f_i is the* support *of f_i. The* Newton Polytope *of f_i is $Q_i = \mathrm{Conv}(\mathcal{A}_i) \subset \mathbb{R}^n$, the convex hull of $\mathcal{A}_i$.*

Polynomials so defined are called *sparse* because we consider a general set of exponents $\mathcal{A}_i$ rather than all exponents of some degree.

Definition 2.2 *The* Minkowski Sum *$A + B$ of convex polytopes A and B in $\mathbb{R}^n$ is the set*

$$A + B = \{a + b | a \in A, b \in B\} \ .$$

$A + B$ is a convex polytope. Let $\mathrm{Vol}(A)$ denote the usual n-dimensional volume of A:

Definition 2.3 *Given convex polytopes $A_1, \ldots, A_n \subseteq \mathbb{R}^n$, there is a unique real-valued function $MV(A_1, \ldots, A_n)$ called the* Mixed Volume *which is multilinear with respect to Minkowski sum, such that $MV(A_1, \ldots, A_1) = n!\mathrm{Vol}(A_1)$. Equivalently, if $\lambda_1, \ldots, \lambda_n$ are scalars, then $MV(A_1, \ldots, A_n)$ is precisely the coefficient of $\lambda_1 \lambda_2 \cdots \lambda_n$ in $\mathrm{Vol}(\lambda_1 A_1 + \cdots + \lambda_n A_n)$ expanded as a polynomial in $\lambda_1, \ldots, \lambda_n$.*

The Newton polytopes capture the combinatorial properties of the system in a remarkable way. We have the following bound on the number of roots of a system of $m = n$ polynomials in n variables, see [Ber75], [Kus76], [Kho78].

Theorem 2.4 *(Bernstein's Theorem) Let $f_1, \ldots, f_n \in \mathbb{C}[x_1, x^{-1}, \ldots, x_n, x_n^{-1}]$. The number of common zeros in $(\mathbb{C}^*)^n$ is either infinite, or does not exceed $MV(Q_1, \ldots, Q_n)$. For almost all specialization of the coefficients c_{ij} the number of solutions is exactly $MV(Q_1, \ldots, Q_n)$.*

2.0.3 Sparse Resultants

For systems of $m = n+1$ polynomials in n unknowns, there are generically no solutions, and there is an algebraic condition on the coefficients for a solution to exist. That is, a solution exists whenever a certain polynomial in the coefficients of the system is zero. This polynomial is called the *resultant* of the system. We use the term *sparse resultant* to refer to the resultant of a system with particular supports, to distinguish from the term "resultant" which has traditionally meant either the Sylvester resultant or the resultant of a homogeneous system. The sparse resultant is the more general object, and includes the others as special cases.

The simplest and most efficient way to define the resultant is as the determinant of matrix whose entries are the coefficients of the polynomials, or zero. This is a generalization of the classical

Sylvester formula for the resultant of two polynomials. In general, the sparse resultant cannot be expressed as the determinant of a single matrix, but as a factor of this determinant. But for our purposes, is just as good to define a matrix whose determinant is a multiple of the resultant, and to discard any extraneous factors.

Once a resultant matrix is defined, it can be used for numerical equation solving or for symbolic variable elimination. For the first use, we construct from the resultant matrix another matrix whose eigenvectors define the solutions of the system [MC93]. This method is based on the use of generalized companion matrices, and provides a particularly simple way to deal with non-linear systems. The resultant matrix is the same for all polynomial systems with a given set of exponents, and can be computed offline. Given this matrix in symbolic form, each particular system can be solved online with only an eigenvalue routine. See section 3.1.

To use a resultant for variable elimination, we repeatedly evaluate the resultant for specializations of the other variables (those not to be eliminated), and use Chinese remaindering and sparse interpolation to reconstruct the answer. This method is applied to surface implicitization in [MC92a].

2.1 The Sparse resultant matrix

For the sparse resultant, we assume we have $m = n+1$ polynomials in n variables. We wish to define a square matrix M whose determinant is divisible by the sparse resultant of $f_1, \ldots, f_{n+1}$. Let Q_i be the Newton polytope of f_i. We need the notion of a mixed subdivision of $Q = Q_1 + \cdots Q_{n+1}$. Mixed subdivisions are defined fully in the appendix (section 7), but a short summary is helpful here:

Definition 2.5 *A* mixed subdivision Δ *of* $Q = Q_1 + \cdots + Q_m$ *is a polyhedral subdivision such that every element* $F \in \Delta$ *is of the form* $F = F_1 + \cdots + F_m$, *with* F_i *a face of* Q_i. *Furthermore* $\dim(F) = \sum \dim(F_i)$.

So a mixed subdivision may be thought of as a decomposition of the Minkowski sum $Q_1 + \cdots + Q_m$ into elementary Minkowski sums $F_1 + \cdots + F_m$. Certain faces of the subdivision play a special role:

Definition 2.6 *A face* $F \in \Delta$ *is called a* mixed facet *if* $\dim(F) = n$ *and every* F_i *has dimension* ≤ 1.

If $m = n$, then all the F_i in a mixed facet F have dimension exactly one. The mixed volume $MV(Q_1, \ldots, Q_n)$ is the sum of the volumes of the mixed facets of Δ in the mixed subdivision Δ of Q. The mixed subdivision, and hence mixed volume, can be computed effectively using a convex hull routine on a lifted polytope as described in the appendix. If only the mixed volume is needed, it can be computed by enumerating the mixed cells as described in section 3.2.1.

Returning to resultants, we assume from now on that $m = n + 1$. The rows and columns of M will be indexed by integer lattice points contained in Q, and the subdivision Δ will be used to select which coefficient c_{ij} appears in each matrix element.

For the selection to be well-defined, we must perturb the Minkowski sum slightly so that each integer lattice point lies in the *interior* of a facet of Δ. Thus we choose a generic vector $\delta \in \mathbf{Q}^n$, and the set of exponents that index rows and colums of M will be

$$\mathcal{E} = \mathbf{Z}^n \cap (\delta + Q)$$

If Δ_δ denotes the subdivision obtained by shifting all faces of Δ by δ, the choice of δ is satisfactory if every $p \in \mathcal{E}$ lies in the interior of a facet of Δ_δ.

We can now define our selection rule for elements of M. We define a function $RC : \mathcal{E} \to \mathbf{Z}^2$ (for row contents) as follows

Definition 2.7 (Row content function) Let $p \in \mathcal{E}$ be an exponent. It lies in the interior of a facet $\delta + F_1 + \cdots + F_{n+1}$ of Δ_δ. Let i be the largest integer such that F_i is a vertex, so $F_i = a_{ij}$ for some j. Then $RC(p) = (i, j)$.

The row of M indexed by $p \in \mathcal{E}$ contains the coefficients of f_i, and represents a multiple of f_i which is

$$x^{(p-a_{ij})} f_i \tag{3}$$

where $(i, j) = RC(p)$. The coefficient of x^q in $x^{(p-a_{ij})} f_i$ appears in the q^{th} column. More explicitly, the matrix M is constructed as

Definition 2.8 M is an $|\mathcal{E}| \times |\mathcal{E}|$ matrix whose rows and columns are indexed by elements of $\mathcal{E}$, and whose elements are

$$M_{pq} = \begin{cases} c_{ik} & \text{if } q - p + a_{ij} = a_{ik} \text{ for some } k, \text{ where } (i, j) = RC(p) \\ 0 & \text{if } q - p + a_{ij} \notin \mathcal{A}_i \end{cases}$$

and therefore $M_{pp} = a_{ij}$ where $(i, j) = RC(p)$. For the matrix to be well-defined all exponent vectors of (3) must lie within $\mathcal{E}$, which we show in the next section.

2.2 A Sample Sparse Resultant

The construction is illustrated for a system of 3 polynomials in 2 unknowns;

$$\begin{aligned} f_1 &= c_{11} + c_{12}xy + c_{13}x^2y + c_{14}x \\ f_2 &= c_{21}y + c_{22}x^2y^2 + c_{23}x^2y + c_{24}x \\ f_3 &= c_{31} + c_{32}y + c_{33}xy + c_{34}x \end{aligned}$$

Pick generic functions

$$\begin{aligned} l_1(x, y) &= L^5x + L^4y \\ l_2(x, y) &= L^3x + L^2y \\ l_3(x, y) &= Lx + y \end{aligned}$$

where L is a sufficiently large integer. These functions define a mixed subdivision of Q as described in the appendix. The input Newton polytopes are shown in figure 1 and the mixed subdivision $\Delta + \delta$ is shown in figure 2.

To illustrate the construction, we will generate one row of the matrix. Choose any point in the subdivision of figure 2, say the point $(1, 2)$, which represents the monomial xy^2. We will fill the

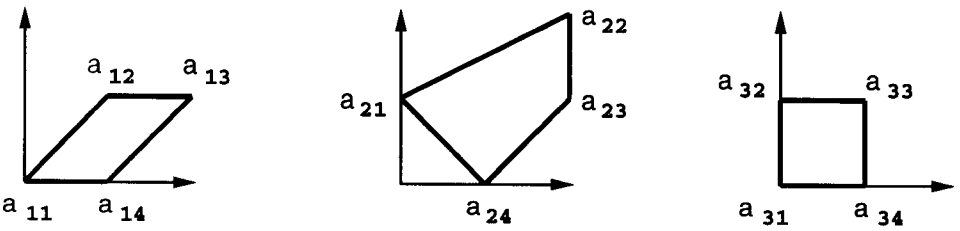

Figure 1: The Newton polytopes and the exponents a_{ij}.

matrix row indexed by $(1,2)$. This point lies in a face $F+\delta$ labelled "a_{11}, a_{33}". This means that F is the Minkowski sum $F_1 + F_2 + F_3$ with $F_1 = a_{11}$ and $F_3 = a_{33}$ both vertices, and F_2 a non-vertex. In this case $F_2 = Q_2$. Either of the points a_{11} or a_{33} would define a suitable row, but the row contents function is defined to choose the larger i value, so $RC(1,2) = (3,3)$.

Now $RC(1,2) = (3,3) = (i,j)$ where i is the number of the polynomial whose multiple fills row $(1,2)$. The multiplier is chosen so that the coefficient c_{33} lies on the leading diagonal. The exponent of the multiplier is $p - a_{33} = (0,1)$. So row $(1,2)$ is filled with yf_3.

$$
\begin{array}{c|cccccccccccccc}
 & 1,0 & 2,0 & 0,1 & 1,1 & 2,1 & 3,1 & 0,2 & 1,2 & 2,2 & 3,2 & 4,2 & 1,3 & 2,3 & 3,3 & 4,3 \\
1,0 & c_{11} & c_{14} & 0 & 0 & c_{12} & c_{13} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2,0 & c_{31} & c_{34} & 0 & c_{32} & c_{33} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0,1 & c_{24} & 0 & c_{21} & 0 & c_{23} & 0 & 0 & 0 & c_{22} & 0 & 0 & 0 & 0 & 0 & 0 \\
1,1 & 0 & 0 & 0 & c_{11} & c_{14} & 0 & 0 & 0 & c_{12} & c_{13} & 0 & 0 & 0 & 0 & 0 \\
2,1 & 0 & 0 & 0 & 0 & c_{11} & c_{14} & 0 & 0 & 0 & c_{12} & c_{13} & 0 & 0 & 0 & 0 \\
3,1 & 0 & c_{24} & 0 & c_{21} & 0 & c_{23} & 0 & 0 & 0 & c_{22} & 0 & 0 & 0 & 0 & 0 \\
0,2 & 0 & 0 & 0 & 0 & 0 & 0 & c_{11} & c_{14} & 0 & 0 & 0 & c_{12} & c_{13} & 0 & 0 \\
1,2 & 0 & 0 & c_{31} & c_{34} & 0 & 0 & c_{32} & c_{33} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2,2 & 0 & 0 & 0 & c_{31} & c_{34} & 0 & 0 & c_{32} & c_{33} & 0 & 0 & 0 & 0 & 0 & 0 \\
3,2 & 0 & 0 & 0 & 0 & c_{31} & c_{34} & 0 & 0 & c_{32} & c_{33} & 0 & 0 & 0 & 0 & 0 \\
4,2 & 0 & 0 & 0 & 0 & 0 & c_{24} & 0 & 0 & c_{21} & 0 & c_{23} & 0 & 0 & 0 & c_{22} \\
1,3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & c_{31} & c_{34} & 0 & 0 & c_{32} & c_{33} & 0 & 0 \\
2,3 & 0 & 0 & 0 & c_{24} & 0 & 0 & c_{21} & 0 & c_{23} & 0 & 0 & 0 & c_{22} & 0 & 0 \\
3,3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & c_{31} & c_{34} & 0 & 0 & c_{32} & c_{33} & 0 \\
4,3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & c_{31} & c_{34} & 0 & 0 & c_{32} & c_{33}
\end{array}
\tag{4}
$$

The matrix M is given in equation (4) with rows and columns indexed by exponent vectors from $\mathcal{E}$. Note that the leading diagonal is always non-zero. This property is important in proving that the determinant of M is non-zero in [CE93].

2.2.1 The Size of M

M is an $|\mathcal{E}| \times |\mathcal{E}|$ matrix. The cardinality of $|\mathcal{E}|$ to a good approximation equals the volume of Q. Ideally, we would like the size of M to be the total degree of the resultant, which can be shown to be $MV(Q_1, \ldots, Q_{n+1})$ or $MV(Q)$ by a slight abuse of notation. So our construction, while it does depend on the Newton polytopes, is suboptimal by a factor of

$$\frac{\text{Vol}(Q)}{MV(Q)}$$

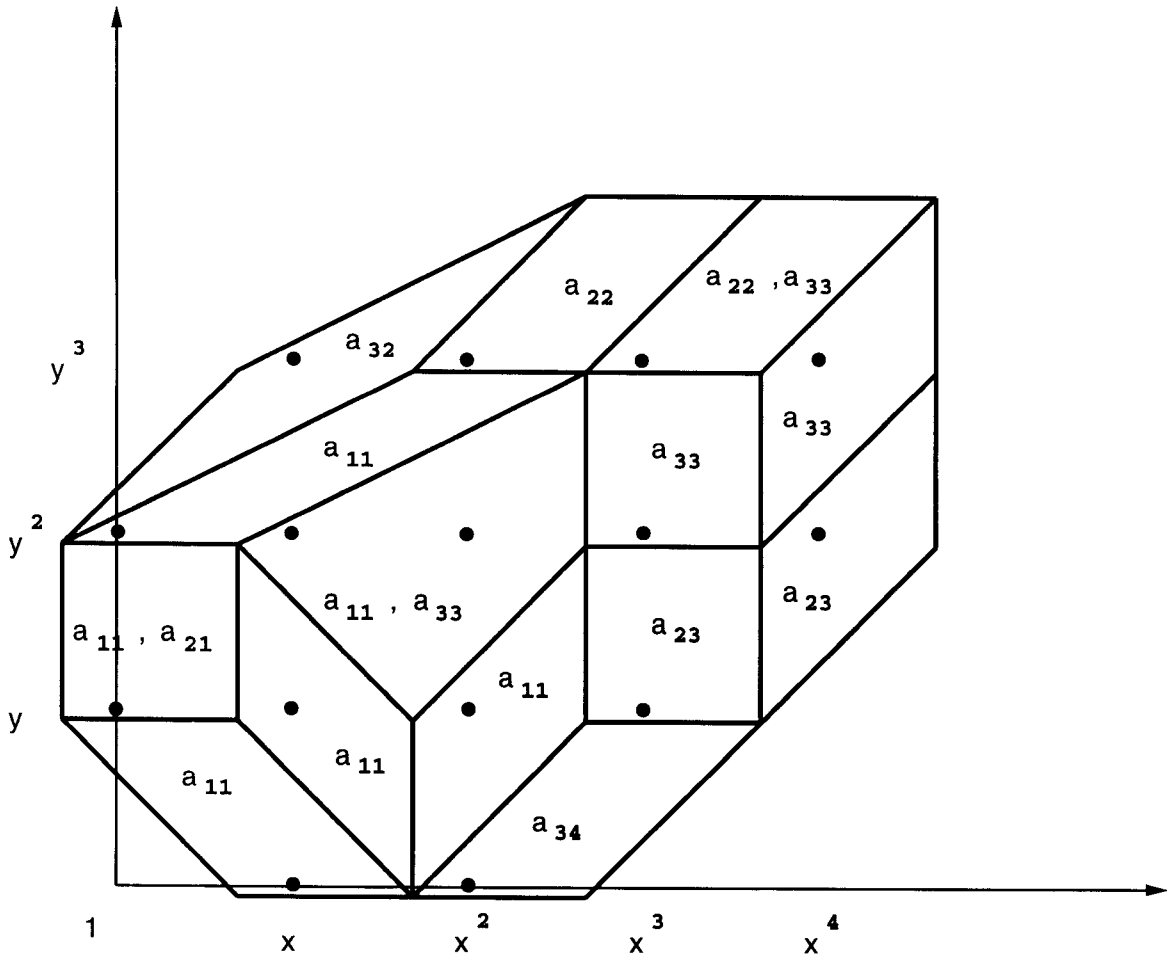

Figure 2: The mixed subdivision Δ_δ of $Q+\delta$. Each facet is labeled with the vertices which contribute to optimal sums within that facet.

In general, this ratio is difficult to determine. Worse than that, there are collections of polytopes whose mixed volume is zero, but whose Minkowski sum volume is finite. For these systems, the suboptimality ratio is infinite.

For reasonable cases, the ratio is better. As an example, suppose that all the Newton polytopes are identical:

$$Q_1 = \cdots = Q_{n+1} \ .$$

Then the total degree of the resultant $MV(Q)$, is the sum of all $n+1$ n-fold Mixed Volumes, each being equal to $n!\mathrm{Vol}(Q_1)$. Hence

$$\deg R = (n+1)!\,\mathrm{Vol}(Q_1) \ .$$

The Minkowski Sum has volume $\mathrm{Vol}(Q) = (n+1)^n \mathrm{Vol}(Q_1)$ and the number of lattice points in it is asymptotically the same [Kan92]. Then $|\mathcal{E}| = \mathcal{O}\left(\frac{n^n \deg R}{(n+1)!}\right)$. Using Sterling's approximation and letting e be the base of natural logarithms, we arrive at

Lemma 2.9 *For unmixed systems*

$$|\mathcal{E}| = \mathcal{O}(e^n \deg R) \ .$$

This is exponential in n, but of course, the mixed volume itself typically grows exponentially with dimension. e.g. if we posit a family of systems with Newton polytopes all equal to dQ_1 for integer d, then the resultant degree is $O(d^n)$. Our resultant matrix size is $O((de)^n)$, and so is polynomial in the resultant degree considered either as a function of d or a function of n. But there is still room for considerable improvement.

2.2.2 A smaller sparse resultant matrix

An improved construction is described in [EC93]. That paper defines a matrix M' whose determinant is a multiple of the resultant, and whose size is at most the size of M. In general, it is known to be impossible to construct a matrix whose determinant is exactly the sparse resultant. So M' has to be larger than optimal. But we claim the construction of M' leads to close-to-optimal size. We cannot make this statement quantitative yet, but have found some strong supporting evidence. Namely, in all the cases where optimal-size matrices are known to exist (enumerated in [SZ93]), the matrix M' has optimal size. This is the algorithm we plan to use in the toolkit for the elimination task. Even if we cannot prove better bounds on its size, we will gather plenty of empirical data on its effectiveness.

3 Toolkit Overview: Numerical Kit

Figure 3 shows the breakdown of the numerical part of the toolkit. It is actually two independent systems. One is based on resultants and eigenvalues, and the other on homotopy methods. These two methods are clear front-runners for numerical equation-solving, both in terms of simplicity and sheer speed, over other current techniques. Between the two, there is no clear winner, and their virtues are complementary, so both are included in the toolkit. Homotopy methods are faster in practice for very large problems. Their complexity for well-conditioned inputs is linear in the number of solutions N of $f = 0$. Their weakness is on singular problems, where they may diverge or run intolerably slowly. Resultant-eigenvalue methods work very well on small to medium sized problems, and they are simple to implement. Because of extensive study of singular eigenvalue problems, a resultant-eigenvalue method can provide more information about singular solutions, and can compute them accurately if their multiplicity is known. For now, the complexity of the resultant-eigenvalue method is $O(N^3)$, assuming an $O(N \times N)$ resultant matrix is available. But because this matrix is very sparse, it may be possible to use more efficient eigenvalue algorithms whose complexity is quadratic or even pseudo-linear in N. This is a topic that warrants further study.

3.1 Equation-Solving with Resultants and Eigenvalues

The idea of solving a non-linear polynomial system with resultants and eigenvalues is both general and simple. We believe it will find very wide application in the future, and along with homotopy methods will become the methods of choice for polynomial equation solving. It is described more fully in [MC91a] and [MC93], and applied to a robotics problem in [MC92b] and to geometric modeling in [MC91b].

In the last section, we summarized the theory of sparse systems and defined the resultant of a system of $n + 1$ equations in n unknowns. Before this general formula was known, there were many

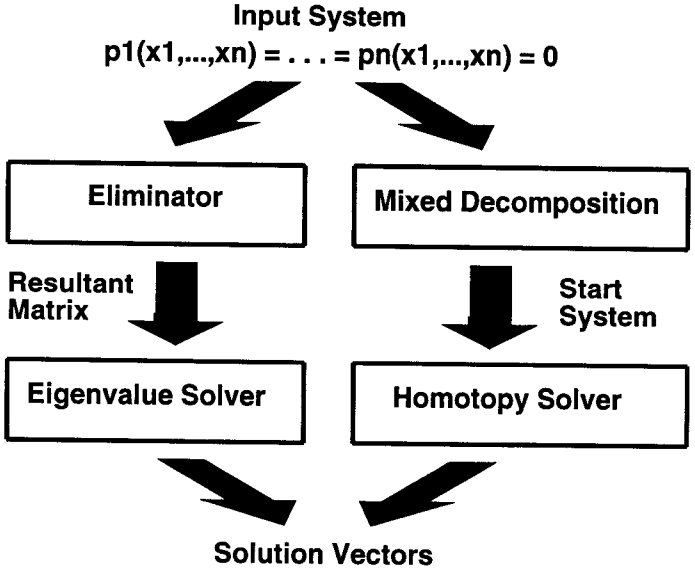

Figure 3: Numerical subsystem components

resultant formulae for specific systems. Almost all these formulae also express the resultant via matrix determinants. e.g. For a pair of polynomials $f(x)$ and $g(x)$ in a single variable, the resultant is the determinant of the Sylvester matrix:

$$\begin{bmatrix} f_n & \cdots & \cdots & f_0 & \cdot & 0 \\ \cdot & \ddots & & & \ddots & \cdot \\ 0 & \cdot & f_n & \cdots & \cdots & f_0 \\ g_m & \cdots & \cdots & g_0 & \cdot & 0 \\ \cdot & \ddots & & & \ddots & \cdot \\ 0 & \cdot & g_m & \cdots & \cdots & g_0 \end{bmatrix}$$

The form of the resultant matrix in the sparse case [CE93] is analogous, although in the multivariate case, the structure is more varied, as can be seen in the example resultant from section 2.2. But one important common feature is that each row of the matrix represents a multiple of one of the polynomials. That is, all the entries in that row are coefficients of that polynomial (or zero). This form is particularly convenient because it allows us to reduce the non-linear equation-solving problem directly to an eigenvalue problem.

3.1.1 Method 1: Adding a linear polynomial

The simplest way to perform the reduction is to add a generic linear polynomial. Suppose we are given $f_1, \ldots, f_n$ as polynomials in $x_1, \ldots, x_n$. We add the linear (in x) polynomial

$$r(s, x) = s - l(x)$$

where $l(x)$ is linear in x, and s is a new variable. We now have $n + 1$ equations in n variables, so the resultant is well-defined. It will be a polynomial $R(s)$ in the new variable s.

$R(s)$ must vanish when all the equations have a common solution, and this is precisely when $s = l(\xi)$, where $x = \xi$ is a solution of $f = 0$. By ordering the rows of the resultant matrix A so rows containing $r(s, x)$ appear after all the others, A can be written in block form as:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21}(s) & A_{22}(s) \end{bmatrix}$$

where the elements of A_{21} and A_{22} are linear in s. After some elementary row operations, we obtain:

$$\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22}(s) - A_{21}(s)A_{11}^{-1}A_{12} \end{bmatrix}$$

whose determinant differs by only a constant factor ($\det(A_{11})$) from the determinant of $B(s) = A_{22}(s) - A_{21}(s)A_{11}^{-1}A_{12}$. Now $B(s)$ is again a matrix whose elements are linear in s. So it can be written $B(s) = sB_1 + B_0$. Multiplying through by B_1^{-1}, we obtain

$$B'(s) = sI + B_0B_1^{-1}$$

and considered as a polynomial in s, the determinant of $B'(s)$ has the same roots as $\det(B(s))$ and $\det(A(s))$ which both differ from it by constant factors. The roots of $B'(s)$ are the eigenvalues of $-B_0B_1^{-1}$.

By the construction, these eigenvalues are the values of $l(\xi)$ for various roots $x = \xi$ of $f = 0$. So, for example, we could set $l(x) = x_1$, and the eigenvalues of $-B_0B_1^{-1}$ will be the x_1 coordinates of the solutions ξ. But a better approach is to choose a generic linear polynomial for $l(x)$, say by specifying $l(x)$ with n random numbers. Then, with high probability, the values of $l(x)$ will be distinct for distinct solutions ξ and ξ'. So long as the solution $x = \xi$ has multiplicity one, $l(\xi)$ will be a simple eigenvalue. In this case, all the coordinates of the solution ξ can be recovered from the eigenvector corresponding to the eigenvalue $l(\xi)$.

This method is even simpler if the sparse resultant matrix of [CE93] is used. It is possible to define the matrix M of section 2.1 in such a way that the constant coefficient of $g(s, x)$, which is the only coefficient depending on s, falls always on the leading diagonal. Consequently, the matrix B_1 is diagonal, in fact the identity matrix, so no inversion of it is needed.

3.1.2 Method II: Generalized Companion Matrices

Suppose we have the same system of polynomials $f_1, \ldots, f_n$ in $x_1, \ldots, x_n$ and we would like to solve $f = 0$. Rather than adding another polynomial, we can reduce the number of variables by hiding one in the base field. So we rewrite $f_1, \ldots, f_n$ as polynomials in $x_2, \ldots, x_n$ whose coefficients are now polynomials in x_1. This time, we have n equations in $n - 1$ variables, so a resultant is well-defined. Again, let A be the matrix whose determinant is the resultant. The elements of A are coefficients of f and so are polynomials in x_1. We can arrange A in powers of x_1 as

$$A(x_1) = x_1^d A_d + \cdots + A_0$$

Assuming A_d is non-singular, we can define

$$A'(x_1) = A_d^{-1}A(x_1)$$

whose determinant has the same roots as $\det(A(x_1))$. Then we use theorem 1.1 [GLR82] to construct a *generalized companion matrix* of the form

$$C = \begin{bmatrix} 0 & I_n & 0 & \dots & 0 \\ 0 & 0 & I_n & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & I_n \\ -A'_0 & -A'_1 & -A'_2 & \dots & -A'_{d-1} \end{bmatrix}, \tag{5}$$

such that the eigenvalues of C correspond exactly to the roots of $\det(A(x_1)) = 0$. C is a numeric matrix of order dN, where N was the original size of A.

In this case, the eigenvalues will be the x_1-coordinates of the root vectors ξ. This is true because we have hidden x_1 in the coefficient field in order that the system $f = 0$ be overconstrained and have a resultant. But this doesnt change the solutions of the system. So there can only be a sequence of values $(x_2, \dots, x_n)$ satisfying $f = 0$ when x_1 is specialized to the first coordinate of a solution. The existence of a solution $(x_2, \dots, x_n)$ for some value of x_1 implies that the resultant must vanish for that value of x_1, and by the construction above, we have shown that the roots of the resultant are the eigenvalues of C.

3.2 Sparse Homotopies

The second subsystem of the numerical kit is based on homotopies. For those not familiar with the homotopy method, a very short explanation follows. Suppose we have a system $f = 0$ that we would like to solve. Suppose also, that we have another system $g = 0$, and suppose both systems comprise n polynomials in n variables. Now consider

$$(1 - \lambda)g + \lambda f = 0$$

For $\lambda = 0$, this reduces to the system $g = 0$. As λ increases from 0 to 1, the solutions vary continously as functions of λ, and when $\lambda = 1$ they must have either diverged to infinity or converged to solutions of $f = 0$. The tracking of solutions is typically done with Newton iterations, taking small discrete steps in λ, from $\lambda_0 = 0$ to $\lambda_N = 1$. Each solution of the system $(1 - \lambda_{i-1})g + \lambda_{i-1}f = 0$ is used as a seed for solution of $(1 - \lambda_i)g + \lambda_i f = 0$.

Under appropriate conditions, we can choose a system $g = 0$ whose solutions converge to all of the solutions of $f = 0$. For example, if f_i and g_i both have the same set of exponents for each i, g has distinct roots, and if we track solutions in complex projective space, we will obtain all the solutions of $f = 0$, provided we dont run into a singularity along the way. This is very useful for sparse systems, because both $f = 0$ and $g = 0$ have a number of solutions given by Bernstein bounds. And the complexity of homotopy methods is very good, in practice almost linear in the number of solutions. So it would seem that homotopy methods immediately offer a way to fully exploit sparseness.

The story is not so simple, because there is no way known in general to design a system $g = 0$ which has given exponents and known, distinct roots. On the other hand if one is always solving systems with the same structure, i.e. set of exponents, one can perform a computation offline to find the roots of one such system $f = 0$ from a somewhat larger start system $g = 0$. Then the roots of $f = 0$ can be saved, and subsequent calculations can use these roots and the system $f = 0$ as the start system. For example, a suitable start system is $g_i = x_i^{d_i} - 1$, for $i = 1, \dots, n$, where d_i is a

bound on the degree of f_i. The number of solutions of this system is given by Bezout, rather than Bernstein bounds so the first calculation may be very expensive. Subsequent calculations will only need to track the Bernstein number of solutions.

But this is a poor solution when one has to deal with polynomial systems that do not all have the same structure. Fortunately, the problem of finding start systems was recently solved in [HS92]. The solution is not in the form of a single start system $g = 0$, but a family of start systems $f|_{\mathcal{F}} = 0$, where $\mathcal{F}$ ranges over the mixed facets of a mixed subdivision. Specifically, in the notation of section 7, let Q_i be the Newton polytope of f_i, and let Δ denote a mixed subdivision of $Q_1 + \cdots + Q_n$. Let $\mathcal{F} = F_1 + \cdots + F_n$ be an n-dimensional face of Δ. Then each F_i is a face of Q_i, and the sum of the dimensions of the F_i will be n. $\mathcal{F}$ will be a mixed cell if and only if all the faces F_i are one-dimensional.

The start system $f|_{\mathcal{F}}$ consists of the equations

$$f_i|_{F_i} = 0, \qquad \text{for } i = 1, \ldots, n$$

where $f_i|_{F_i}$ is the polynomial f_i with all terms set to zero except those whose exponents lie in F_i. Thus the Newton polytope of $f_i|_{F_i}$ is F_i. While it has a one-dimensional Newton polytope, $f_i|_{F_i}$ is not a univariate polynomial, and finding the roots of $f|_{\mathcal{F}} = 0$ is still non-trivial. But after a monoidal change of variables $x_i \mapsto y^{b_i}$, where $y = (y_1, \ldots, y_n)$, $b_i \in \mathbb{Z}^n$, we obtain polynomials $f_i'|_{F_i}(y_i)$ which are univariate, and which are easily solved. The number of roots of $f|_{\mathcal{F}} = 0$ either before or after the monoidal transformation is exactly the volume of the polytope $\mathcal{F}$. Applying the monoidal transformation to the y-solutions obtained by solving the univariate system gives us the x-solutions of $f|_{\mathcal{F}}(x) = 0$. We track these solutions from $f|_{\mathcal{F}}(x) = 0$ to $f = 0$, using a homotopy method.

There is a very satisfying proof in [HS92] which shows that every root of $f = 0$ is the endpoint of a path begining at a root $f|_{\mathcal{F}} = 0$ for some mixed facet $\mathcal{F}$. The number of roots of $f = 0$ is therefore the total of the numbers of roots of the $f|_{\mathcal{F}} = 0$. This is exactly the total volume of all the faces $\mathcal{F}$, which is exactly the mixed volume of $Q_1, \ldots, Q_n$. In this way, [HS92] provides a constructive proof of Bernstein's theorem.

3.2.1 Computing Mixed Cells and Volumes

The idea of a mixed subdivision of a Minkowski sum of polytopes is described in an appendix (section 7). We hinted in the previous section how the mixed subdivision furnishes information about a family of homotopy start systems, one for each mixed cell of the subdivision. To complete the implementation of a homotopy solver for sparse systems then, we need an efficient algorithm for computing either mixed subdivisions, or enumerating directly the mixed cells of a subdivision.

Because of the complexity of a mixed subdivision in high dimensions, we have found it much more effective to enumerate only the mixed cells. Let $\mathcal{F} = F_1 + \cdots + F_n$ be a mixed cell of a subdivision Δ of the Minkowski sum $Q_1 + \cdots + Q_n$. Then every F_i is an edge of Q_i. We could enumerate all possible mixed cells by enumerating all n-tuples of edges, one from each polytope, and checking them, but this method soon runs into a computational brick wall. Imagine for example, that $n = 6$ and each polytope has 20 edges - there would be 65 million possibilities to check. Fortunately, there is a much better search strategy.

Define the Minkowski sum of the first i faces contributing to $\mathcal{F}$:

$$\mathcal{F}_{\leq i} = F_1 + \cdots + F_i$$

Similarly, in the space of lifted polytopes, $\mathbb{R}^{n+1}$, we have

$$\hat{\mathcal{F}}_{\leq i} = \hat{F}_1 + \cdots + \hat{F}_i$$

A necessary condition for $\mathcal{F}$ to be a mixed facet is that every $\hat{\mathcal{F}}_{\leq i}$ is a facet of the partial Minkowksi sum $\hat{Q}_1 + \cdots + \hat{Q}_i$, for $i = 1, \ldots, n$. This immediately suggests a search strategy:

Our mixed volume algorithm constructs a tree of candidate $\hat{\mathcal{F}}_{\leq i}$'s. The root of the tree is level zero. The first level of the tree contains all lifted edges of $\hat{Q}_1$. The k^{th} level of the tree contains candidate k-fold sums of edges. Let $\hat{\mathcal{F}}_{\leq k}$ be the k-fold sum corresponding to tree node v. We explore v by checking whether $\hat{\mathcal{F}}_{\leq k}$ lies on the boundary of $\hat{Q}_1 + \cdots + \hat{Q}_k$ using linear programming. If not, then no further expansion of v occurs. If $\hat{\mathcal{F}}_{\leq k}$ is on the boundary, then a new child u of v is created for each edge $\hat{e}$ of $\hat{Q}_{k+1}$. The facet at u is the Minkowksi sum $\hat{\mathcal{F}}_{\leq k} + \hat{e}$. These nodes are explored in the same manner. The search continues down to level n, and those leaves whose facets lie on the boundary of $\hat{Q}$ define the mixed cells.

This search strategy works well in practice, and has allowed us to solve some very large mixed volume calculations, in up to 28 dimensions. As a benchmark, we give its running times for bounding the number of *cyclic n-roots* for various n. The cyclic n-roots problem is of some independent interest, but it has primarily served as a computational benchmark for algebraic algorithms. More details and references are given in [BF91].

The equations themselves arise in Fourier analysis. There are n equations in the variables $x_1, \ldots, x_n$, and they have the form:

$$\begin{aligned} x_1 + \cdots + x_n &= 0 \\ x_1x_2 + x_2x_3 + \cdots + x_nx_1 &= 0 \\ x_1x_2x_3 + x_2x_3x_4 + \cdots + x_nx_1x_2 &= 0 \\ &\vdots \\ x_1x_2\cdots x_n &= 1 \end{aligned} \tag{6}$$

Using our mixed volume algorithm, we computed the Bernstein bounds for cyclic systems for various values of n, and these are shown in figure 3.2.1. The mixed volume algorithm is written in $ANSI - C$, and the test data is from a SUN Sparc-10 workstation.

The last equation $x_1 \cdots x_n = 1$ forces all the solutions to lie in $(\mathbb{C}^*)^n$. So Bernstein's theorem provides an exact count allowing for multiplicities. In the one case where the number of solutions is known, $n = 7$ [BF91], the multiplicities are all 1, and the Bernstein bound is exact. To our knowledge, the bounds for $n = 9$ and $n = 10$ have not been computed before, and the value for $n = 7$ tooks many hours to compute using a Gröbner algorithm called Bergmann in [BF91]. This is an encouraging sign for the effectiveness of the mixed volume algorithm.

4 Toolkit Overview: Symbolic Kit

As shown in figure 5, the primary function of the symbolic toolkit is to determine solvability of systems of polynomial inequalities over the real numbers. A numerical approximation to the solution, if any, can be easily extracted. The symbolic toolkit can also compute optimal solutions to systems of inequalities, given a polynomial objective function.

n	Bernstein bound	Computing time
7	924	1m 2s
8	2560	9m 37s
9	11016	1h 42m 50s
10	36650	20h 22m 1s

Figure 4: Bernstein bounds for the cyclic n-roots problem

We plan also to use various modules of the toolkit for other purposes. For example, computing connected components of algebraic curves, and ordering points along those curves. This leads ultimately to an algorithm for computing connected components [Can88a]. We also plan to implement some fast query algorithms for point location in arrangements of polynomial surfaces. The algebraic algorithms needed in these applications are all contained in the toolkit. This explains the toolkit moniker, and the module-by-module description in this report. But in this report, for reasons of space, the only calculation described will be solution of systems of inequalities.

The input is a predicate that specifies a *semi-algebraic set.* The predicate can be a general boolean combination, rather than simply a conjunction, of polynomial inequalities:

Definition 4.1 *A semi-algebraic set S is the set of points in $\mathbf{R}^n$ satisfying a predicate of the form $B(A_1, \ldots, A_k)$ where $B : \{0,1\}^k \to \{0,1\}$ is a boolean function and each A_i is an atomic formula of one of the following types:*

$$(f_i = 0),\ (f_i \neq 0),\ (f_i > 0), (f_i < 0),\ (f_i \geq 0),\ (f_i \leq 0) \tag{7}$$

with each f_i a polynomial in $x_1, \ldots, x_n$ with rational (for our computational purposes) coefficients.

Semi-algebraic (SA) sets are a versatile class that includes the forms of most familar objects in $\mathbb{R}^3$, such as cones, cylinders, spheres, and combinations of these. SA sets in 3D are slightly more general than CSG (constructive solid geometry) models, for example, SA sets include sets of mixed dimension, and sets which are not topologically closed. Both of these are forbidden in CSG.

SA sets are defined in any dimension, and can also be used to represent the set of legal (e.g. obstacle-avoiding) configurations of a mechanical system, such as a robot, or the set of camera transformations consistent with some geometric constraints, or the set of positions at a given distance from a surface to guide a milling machine. The defining formulae for SA sets are the real analogue of SAT formulae, and they provide the ability to search over a space of real values for a solution satisfying some property.

The goal of the inequality solver then, is to decide if a semi-algebraic set S is non-empty, and to produce a sample point if it is. We describe each module of the solver in the following sections. The next section, on the sample point enumerator, gives also an overview of the whole system.

4.1 Sample point enumerator

In order to decide if the predicate is satisfiable, that is, if the semi-algebraic set S it defines is non-empty, the solver tries to find a witness point $v \in S$. The existence of such a point is a proof of

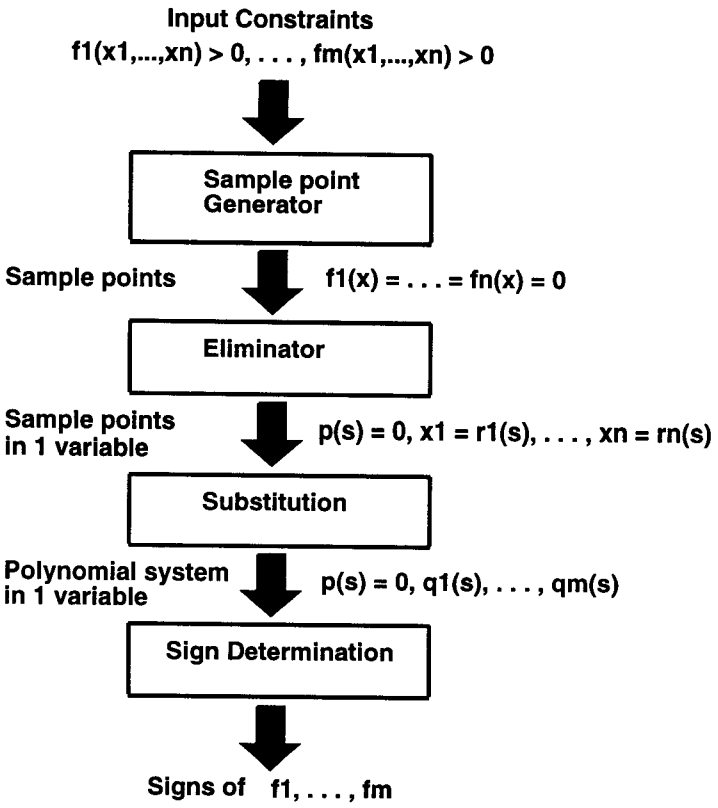

Figure 5: Symbolic subsystem modules

satisfiability. It is also possible to prove unsatisfiability by enumerating sufficiently many potential witness points, one of which must lie in S if S is non-empty.

For simplicity of explanation, we will assume that the set S is closed and bounded, and that the surfaces $f_i = 0$ are in *general position.* These assumptions are not necessary, and methods for dealing with other cases are given in [Can93b].

Since we know the polynomials f_i are in general position, the intersection of any $n+1$ of them in n dimensions will be null. Any $j \leq n$ of them will intersect in a manifold of dimension $n-j$. Let P be an extremal point of π in the set S. Then P is also an extremal of π in a manifold M which is the set of zeros of some polynomials $f_{i_1}, \ldots, f_{i_j}$. These polynomials are precisely the f_i which are zero at P.

So to enumerate all potential witness points, we enumerate the critical points of π on the set of common zeros of $f_{i_1}, \ldots, f_{i_j}$, for every set of $j \leq n$ polynomials. First we define a polynomial

$$g = \sum_{l=1}^{j} f_{i_l}^2$$

and solve the system

$$g = \mu \qquad \frac{\partial g}{\partial x_2'} = \cdots = \frac{\partial g}{\partial x_n'} = 0 \tag{8}$$

"in the limit" as $\mu \to 0$. The coordinates $x'_2, \ldots, x'_n$ are a basis for the $n-1$ dimensional linear space which is the kernel of π. The process of solving this system in the limit is described in [Can88b] and [Can90], and involves computing the resultant of the system, arranging it in powers of μ, and retaining the lowest degree coefficient. The result is a polynomial $p(s)$ and rational functions $(r_1(s), \ldots, r_n(s))$ such that the solutions to the system (8) are all the tuples $(r_1(\alpha_i), \ldots, r_n(\alpha_i))$ where α_i are the roots of $p(s) = 0$.

To compute the signs of the other polynomials at these critical points, we substitute $x_i \mapsto r_i(s)$ for $i = 1, \ldots, n$, giving $q_i(s) = f_i(r(s))$ and the set of signs we are looking for is precisely the sign sequences of

$$(q_1(s), \ldots, q_k(s)) \text{ at roots of } p(s) = 0$$

and to find these we simply apply the sign determination algorithm of the last section (to numerator and denominator, since the q_i here are rational functions).

4.1.1 Summary of the inequality solver

We first choose a generic linear map $\pi : \mathbb{R}^n \to \mathbb{R}$ by selecting n random integers $\pi_1, \ldots, \pi_n$. Then the algorithm proceeds as follows:

- **Sample point enumeration:** Enumerate all subsets of $j \leq n$ polynomials $\{f_{i_1}, \ldots, f_{i_j}\}$. Do this in order of increasing j, so that "easy" witness points will be found early.
- **Elimination:** For each subset, construct a representation of the critical points of π as a polynomial $p(s)$ and rational functions $r_1(s), \ldots, r_n(s)$.
- **Substitution:** Substitute $r_i(s)$ for x_i in the other polynomials, giving $q_i(s) = f_i(r(s))$ for $i = 1, \ldots, k$.
- **Sign Determination:** Determine the signs of the $q_i(s)$ at roots of $p(s) = 0$ using the algorithm of section 4.3. Substitute these signs into the formula B to check if the corresponding critical point lies in S. If yes, S is non-empty, so return "true".
- Continue until no more sample points, and then return "false".

4.2 Reduction to a univariate problem

The function of this module is to take a system of n polynomials in n variables

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

which has solutions $\xi^{(i)} \in \mathbb{C}^n$, for $i = 1, \ldots, N$, and produce from it a univariate polynomial $p(s)$ and rational functions $r_1(s), \ldots, r_n(s)$. The pair (p, r) is a symbolic description of the roots $\{\xi^{(i)}\}$ in the following sense: If the roots of $p(s) = 0$ are $\alpha^{(i)} \in \mathbb{C}$, for $i = 1, \ldots, N$, then

$$\xi^{(i)} = r(\alpha^{(i)})$$

Put another way, $r(s)$ is a parametric curve in $\mathbb{C}^n$ which passes through all the roots $\xi^{(i)}$, $i = 1, \ldots, N$. The values of s at which it arrives at a root are precisely the roots of $p(s) = 0$. The system $f = 0$ will usually be derived from equation 8 of the last section.

There are several ways to construct (p, r) from f. One method uses polynomial GCDs and was described in [Can88b]. A more efficient method based on differentiation was given by Renegar [Ren89], and that is the one we use in the toolkit. Compared to the cost of computing the polynomial $p(s)$, Renegar's method computes all the $r_i(s)$'s in only n times as many operations.

To start, we add to the system $f = 0$, a linear polynomial

$$s - l(x) = 0, \qquad \text{where } l(x) = l_0 + l_1 x_1 + \cdots l_n x_n$$

and compute the resultant $R(s, l)$. Suppose that l is specialized to a random linear polynomial L, then with probability one, the values $l(\xi^{(i)})$ will be distinct for distinct $\xi^{(i)}$. The roots of the resultant $R(s, l = L)$ will be $s = L(\xi^{(i)})$, and will also be distinct. We set $p(s) = R(s, l = L)$ and we are halfway to our goal.

We obtain the functions r_i by differentiating $R(s, l)$. We define r_i as

$$r_i(s) = \left. \frac{\left(\frac{dR(s,l)}{dl_i}\right)}{\left(\frac{dR(s,l)}{dl_0}\right)} \right|_{l=L}$$

The reader can verify that the r_i's have the correct values at $s = \alpha^{(j)}$ by computing the derivatives of $R(s, l)$ in its factored form $R(s, l) = \prod_{j=1}^{N}(s - l(\xi^{(j)}))$.

In our implementation, it is particularly easy to compute the derivatives. All the algorithms in the symbolic toolkit use an SLP (straight-line program) representation of arithmetic. See section SLP-arithmetic below. The polynomial $p(s) = R(s, l = L)$ is computed explicitly. That is, we compute SLPs for each of the coefficients $p_0, \ldots, p_m$ of $p(s)$. Because they depend on the coefficients of l, they can be written $p_i(l)$. Then we compute the derivative of the SLP for $p_i(l)$ with respect to some l_j. For each node v of $p_i(l)$, we add a new node representing the derivative of v with respect to l_j. This roughly doubles the size of the SLP (see section 4.5.1). Computing all the derivatives of $R(s, l)$ increases the size of the SLP by a factor of $n + 2$.

This method works even if the system $f = 0$ has roots with multiplicity > 1. When this happens, $p(s)$ will have repeated factors for all choices of l. The partial derivatives above will all be zero. But we can remove the repeated factors by first choosing a random l, setting $p(s) = R(s, l)$ and then defining

$$\hat{p}(s) = p(s)/GCD(p(s), \frac{dp}{ds})$$

This time, we define $r_i = \frac{d\hat{p}}{dl_i} / \frac{d\hat{p}}{dl_0}$. The partial derivatives are again computed using SLPs, and they will be non-zero for almost all choices of l.

4.3 Sign Determination

Most of the recent work on real algebraic algorithms makes use of a sign-determination lemma due to Ben-Or, Kozen and Reif [BOKR86]. This lemma, henceforth called "BKR", takes a univariate polynomial $p(s)$, and polynomials $q_1(s), \ldots, q_k(s)$, and returns m sign sequences $\sigma \in \{-, 0, +\}^k$, where m is the number of real roots of $p(s)$. Each sign sequence σ corresponds to a particular root α_j of $p(s)$, in such a way that $\sigma_i = \text{sign}(q_i(\alpha_j))$.

Given the importance of BKR, it is natural to look for improvements and simplifications. In [Can91b] and [Can93b], we described a faster and simpler version of BKR, which allows purely

symbolic elimination. It eliminates the matrix rank tests of BKR, replacing them with a simple recursive algorithm.

4.3.1 Sign-Determination Algorithm

The input is the polynomial $p(s)$, and polynomials $q_1(s), \ldots, q_k(s)$ with rational coefficients, and of degree at most d. The output is the set of sign sequences $\{(q_1(\alpha_j), \ldots, q_k(\alpha_j))\}$ at the $m \leq d$ real roots α_j of $p(s)$. Note that the set of sign sequences is an unordered set, and the algorithm does not output the sign sequences in the order of the α_j as real numbers.

1. We assume at the i^{th} step that the algorithm knows for each sign sequence of $q_1, \ldots, q_{i-1}$, how many roots of $p(s)$ produce this sign sequence. Most of these are zero, and the algorithm only stores the sign sequences with at least one root, and the number of these is $m_{i-1} \leq d$.

2. There are $3m_{i-1}$ possible sign sequences for $q_1, \ldots, q_i$, and each of these defines a column of the matrix $K_{i,3m_{i-1}}$. These columns are linearly independent, so there are $3m_{i-1}$ rows which together with the specified columns, define a square submatrix J_i of $K_{i,3m_{i-1}}$.

3. We solve the $3m_{i-1} \times 3m_{i-1}$ system corresponding to this matrix, to find the actual sign sequences of $q_1, \ldots, q_i$, and repeat the above steps for $i = 1, \ldots, k$.

The first task then, is to give a procedure that accepts a list of m columns of the matrix $K_{n,m}$, and returns a list of m rows, such that the resulting $m \times m$ matrix is non-singular. This procedure needs to run in polynomial time in n and m and not the size of $K_{n,m}$.

The second task, which is very easy, is to determine the entries of this submatrix of $K_{n,m}$. Again this must be in polynomial time in m, so we cannot aford to construct all of $K_{n,m}$. This task reduces to determining the value of a single element of $K_{n,m}$ given its row and column indices. Both steps can be solved with simple recursive procedures whose running time is $O(nm)$ and $O(n)$ respectively. Refer to [Can91b] and [Can93b] for details.

Let m denote the number of real roots of $p(s)$, then m is not more than d, the degree of $p(s)$. The overall running time is:

$$O(n(md^2 \log^2 m + m^3)) \tag{9}$$

assuming naive algorithms for polynomial arithmetic. The improvement over the original BKR is that we have reduced the maximum number of polynomials in a Sturm query from n to $\log m$. Since the actual number of real roots of a polynomial is small compared to its degree, the complexity of BKR without this improvement would typically be nm^3d^2 times some log factors. With the improvement, we typically get nmd^2 times some logs.

4.4 Efficient Arithmetic

There are traditionally two routes to take when writing code to solve algebraic problems (i) Using floating point, and (ii) using exact arithmetic and arbitrary precision integers. Neither approach is satisfactory for large problems.

If a single floating point number is used to approximate a real number, then approximate tests for equality, based on an "epsilon" parameter, have to be used. Two numbers that are nearly equal are delared to be equal, and there is no way to tell if this is a false assumption. The author experimented

with this approach in the mid 1980s and found that even with double precision arithmetic, Sturm sequence calculations were unreliable for polynomials of degree > 10. Conversations with other researchers since have supported this conclusion. A refinement of this approach is to use interval arithmetic. A real number r is represented as a pair of floats (ϕ_1, ϕ_2), such that $\phi_1 < r < \phi_2$, and $|\phi_1 - f_2|$ is as small as possible. This eliminates some types of error, but if two intervals overlap, it is still impossible to tell if the real numbers they represent are equal or not.

Finally, it is possible to construct "arbitrary precision" floating point, which works at fixed precision until an operation is performed which is a test for zero (or equality). Then the precision is automatically increased until the result of the test is known for certain. An arbitrary precision floating point system is described in [Pri91]. This approach is as good as a symbolic approach and guarantees correct results. The problem is that if the two quantities to be tested really are equal, the precision must be extended all the way to the equivalent of exact integer calculation. In the applications we have in mind, this may involve integers of hundreds of words in length. Arithmetic on such integers takes quadratic time in practice, so the calculation of the two values to be compared, and all those they depend on, slows down by a factor of 10^4 to 10^5 compared to double-precision floating point.

Exact calculation requires arbitrary precision integer arithmetic, and to be efficient, it also requires finite field arithmetic and chinese remaindering. Modular integer arithmetic using chinese remaindering has complexity $O(cp + ap^2)$ with naive algorithms, where c is the number of intermediate arithmetic steps, a is the number of integers in the result, and p is the precision in bits. Often cp dominates ap^2. For example, for solving an $n \times n$ system of linear equations, c is $O(n^3)$, a is n, and p for an exact result is $O(n)$. In these cases chinese remaindering offers linear complexity in p compared to quadratic complexity if arbitrary precision arithmetic is done at the intermediate steps. The only problem is that p may not be known exactly, and using an a priori bound on p often leads to unnecessarily high precision. To get around this problem, in [MC93] we used a probabilistic scheme that does chinese remainder lifting incrementally, and stops when it has sufficient precision. At this point the lifted integers stop changing with from one iteration to the next.

Even though incremental chinese remaindering method brings down the complexity of large integer arithmetic, it suffers from the same problems as arbitrary precision floating point. For our problems of interest, the integers may be hundreds of words long. Even with chinese remaindering, this slows the algorithm down by a factor of several hundred compared to fixed-precision floating point. This is still too large a penalty to pay for exact computation.

4.4.1 A Mixed Approach

We will take a new approach, based on the use of both finite field and floating point arithmetic, to achieve constant cost per arithmetic step. The idea is to use finite fields to test for equality, and then floating point to compute the approximate values. For example, in Sturm sequence or polynomial GCD calculation, it is critical to be able to tell if the leading coefficient of a remainder is zero. But to use the sequence, all that is required is the signs of the actual leading coefficients, and these are usually far from zero. In what follows, we assume the numbers we are trying to represent are (large) integers. It is possible to extend the method, with some effort, to rational numbers, but all the calculations we forsee in the toolkit require integers only.

In our method, we represent a (possibly very large) integer k as a structure with two values. One is the fixed precision integer $k \bmod p$, and the other is a floating point approximation ϕ_k, to k. An

arithmetic operation on arguments of this type consists of separate operations on the mod p and float fields. To check two mixed numbers for equality, we check if their mod p values are equal. If so, we declare them to be equal. The probability of an incorrect answer is very small, and depends on the size of p. If we require the relative order of two mixed arguments, we compare first the mod p values to check for equality, and then their float values, and order them accordingly.

In the algorithms we use, equality comparisons are much more frequent than ordering (e.g. every addition or subtraction of polynomials includes a check for cancellation of the leading coefficient). It is still possible using mixed arithmetic that we try to compare two unequal numbers, and find that their floating point descriptions are too close for us to be able to order them. But because the ordering comparisons are much less common, the chances of this happening are much lower than if we tried to use floating point alone for equality testing. Nonetheless, ordering tests are a critical part of algorithms such as the sign determination algorithm, and they cannot be done with the mod p representation alone. For slightly greater cost than either, a mixed representation give us correct sign information, and very low probability of failure.

For simplicity, we have described the mixed representation using a mod p integer and a floating point number. But we also plan to try a mixed representation with an integer mod p and an interval of two floats. This is a safer representation since the ordering check can determine if the float descriptions are too close to order, and flag failure rather than producing an incorrect ordering.

4.5 Computing with Infinitesimals

In the algorithms of [HRS90], [Ren92], [GV92] and [Can91a], various singularities are dealt with by perturbing the input polynomials with infinitesimals. This moves the problem away from the singularity, and when done carefully preserves the important properties (like connectivity or non-emptiness) of the input. Computations with an infinitesimal ϵ are done in the rational field $\mathbb{Q}(\epsilon)$. That is, each number a or b in this extension field is a rational function (a quotient of polynomials) in ϵ. To perform arithmetic, we use the usual rules for arithmetic on rational functions. To determine the sign of such an element, we exclusive-or the signs of its numerator and denominator, which are polynomials in ϵ. To determine the sign of a polynomial in ϵ, we use the sign of the lowest degree non-zero coefficient.

But it is very expensive to compute with explicit rational functions. For example, in the extension $\mathbb{R}(\mu, \epsilon, \delta, \rho)$ that we have been using, an element of degree 10 would have several hundred coefficients. But the sign of the element, which is all we need for the sign-determination algorithm, is determined by just one of these coefficients. This element is the lowest degree element under the lexicographic ordering $\mu \prec \epsilon \prec \delta \prec \rho$

If we knew that this element was say $\mu^4\epsilon\delta^2\rho$, we could find it by computing modulo the ideal $(\mu^5, \epsilon^2, \delta^3, \rho^2)$, which effectively discards higher-degree terms. Since we dont know the degree, we would have to do some search, gradually increasing degree until we obtain a non-zero term. Rather than doing this repeatedly, we can obtain the lowest degree term by differentiating a straight-line program.

4.5.1 Straight-Line Program Arithmetic

In our approach, we use arithmetic straight-line programs to represent the intermediate values in a calculation, as advocated by Kaltofen et al. in [Kal88, Kal89, FIKY88]. An SLP may be represented

as a directed acyclic graph. Each node of the graph holds an operation type, such as "+" and a value. If the operation is a binary operation, the node will have indegree two, with two incoming edges from the nodes which are arguments to the operation.

Computation with SLPs is split into two phases (i) creation and (ii) evaluation. During creation, unevaluated SLP nodes are created. e.g. If a and b are numbers represented as SLPs, computing $c = a + b$ means adding an SLP node with undefined value, operation type "+" and two edges from the nodes representing a and b. During evaluation of a node, the value field of that node is computed by applying the operation to the value fields of the two argument nodes. Evaluation involves a depth-first search of the SLP graph.

In our implementation, the creation and evaluation phases are transparent, and only normal arithmetic operations are visible. We do this by implementing arithmetic operations with creations, and comparisons with evaluations. This is a form of lazy evaluation, although the SLP sub-graph on which a node depends is always retained, even after the node is evaluated.

The data structure for an SLP node has several essential fields: two argument pointers, which point to a and b in the example above, the operation field (e.g. "+"), and a value field, which holds the value of the node during evaluation. There will usually also be a flag field to indicate that a node has already been evaluated, so that the SLP can be evaluated with depth-first search. And in the toolkit, we will need the values of various derivatives and anti-derivatives of SLP nodes, which adds two additional pointers to the data structure.

Suppose we have computed an element $a \in \mathbb{R}$ from some other real values $b_1, \ldots, b_m$ via a series of arithmetic operations. For example, such a could be a coefficient of $p(s)$ described in section 4.2, or a coefficient of one of the Sturm query polynomials. We can represent a as an SLP rooted at the values $b_1, \ldots, b_m$. Now suppose that b_1 is specialized to the infinitesimal value ϵ, and that the other b_i take on integer values. We would like to know the sign of a. For simplicity we assume that $a = a(\epsilon)$ is a polynomial in ϵ. This is all we need in our applications.

We could substitute $\epsilon = 0$ and evaluate the SLP over the rationals. If we are lucky, $a(0)$ will have a non-zero value, and this gives the sign of $a(\epsilon)$. If not, we can construct an SLP for the derivative $\frac{da(\epsilon)}{d\epsilon}$. This has roughly double the size of the original SLP. Now evaluating this program at $\epsilon = 0$ gives us a_1, the coefficient of ϵ in $a(\epsilon)$. If this is non-zero, it gives us the sign of $a(\epsilon)$, otherwise we compute the second derivative, and continue. The extra program for the k^{th} derivative is about $k + 1$ times the size of the original program, and it uses nodes from the first $k - 1$ derivatives. The *total* program size to compute the k^{th} derivative is $\binom{k+2}{2}$ times the original.

This process generalizes easily to multivariate elements, using randomization. For example, to find the sign of $a(\mu, \epsilon, \delta)$ with $\mu \ll \epsilon \ll \delta$, we first substitute random integer values for ϵ and δ. With high probability, this doesnt change the degree of the lowest degree term in μ. Then we apply the procedure above to obtain an SLP for the first non-zero derivative wrt μ at $\mu = 0$. Let $aa(\mu, \epsilon, \delta)$ denote this derivative. Then $aa(0, \epsilon, \delta)$ is the lowest-degree coefficient of a in μ, times the constant $k_1!$, where k_1 is the order of the derivative.

We iterate the process, and set δ to a random integer, μ to zero, and run the univariate procedure on the SLP for aa as a polynomial in ϵ. This gives us an SLP for the first non-zero derivative wrt ϵ at $\epsilon = 0$, which we denote $aaa(\mu, \epsilon, \delta)$.

Finally, we run the univariate routine on the SLP aaa with μ and ϵ both set to zero. Evaluating the resulting program at $\mu = \epsilon = \delta = 0$ gives the sign of the lexicographically first term, which is what we need.

Some simple analysis shows that the SLP for computing the sign when the lowest degree term is $\mu^{k_1}\epsilon^{k_2}\delta^{k_3}$ is $k_1^2k_2^2k_3^2$ times the original. More generally we have

Proposition 4.2 *Let $P(\epsilon_1,\ldots,\epsilon_m)$ be a polynomial, represented as an SLP with L vertices. If $\epsilon_1 \prec \cdots \prec \epsilon_m$ are infinitesimals, and if the lexicographically first term in P is $c\epsilon^{k_1}\cdots\epsilon^{k_m}$, then an SLP for this term can be constructed having size $\leq Lk_1^2\cdots k_m^2$, in the same number of steps.*

We claim this method is useful in practice because the k_is are typically small constants, independent of the degree of a in μ,ϵ,δ. Each infinitesimal is used to perturb away from a possibly singular input, and the degree in that infinitesimal is a measure of the multiplicity of the singularity. Where the input is not singular at all, the degree in that infinitesimal will be zero. Most of the time, we expect small multiplicities, and the cost of working over the infinitesimal extension should be only a small constant factor more than integer arithmetic, this factor being the increase in the SLP size.

4.6 Complexity

We measure the complexity of a predicate with four quantities, the number of polynomials k, the number of variables n, the maximum degree of the polynomials d, and the maximum coefficient length c of the coefficients of the polynomials.

The time complexity of the inequality solver is

$$k^{n+1}d^{(O(n))}c^2$$

arithmetic steps. This is an improvement over the previously published algorithms for the existential theory [Ren92], [HRS90] and [GV92], and is within a factor of k of optimal. More importantly, the algorithm is simple, and there are no large constants hidden in the exponents.

The $d^{O(n)}$ bound can be made more precise. The use of sparse elimination, mixed arithmetic and lazy limit-taking gives a *typical* complexity of $O(d^{3n})$. This assumes that the input is non-singular or has only low-order singularities. Higher order singularities may require many derivatives to be taken, pushing the complexity up as a polynomial in the number of derivatives.

This is a dramatic improvement over other published algorithms [Ren92], [HRS90] and [GV92], all of which give only a "big-O" estimate of the exponents of both k and d, that is, their bounds are of the form $(kd)^{O(n)}$. Based on the number of infinitesimals used, it is unlikely that any of these algorithms has typical complexity less than $O((kd)^{7n})$.

5 Future Work

5.1 Coefficient relations

We have now a good understanding of Newton polytope volumes and their effect on the number of solutions. Given a system with general coefficients, we can compute the sparse resultant, and that is the smallest possible eliminant. But often in practice, the coefficients of the system we are given will satisfy some relations, and the sparse bounds are no longer the tightest possible.

As an example, consider the system $f_1 = 1 + x + y + xy$, and $f_2 = 1 + 2x + 3y + 4xy$, whose Newton polytopes are the same unit square. The mixed volume of the system is 2, indicating two solutions. Now suppose we make a change of coordinates, say $x = u + v$, $y = u - v$. Then the

equations become $f_1' = 1 + 2u + u^2 - v^2$, and $f_2' = 1 + 5u - v + 4u^2 - 4v^2$. Both these equations have Newton polytopes which are simplices of side length 2, and the mixed volume is 4. The number of affine solutions has not changed, but the coefficients are now dependent, and the leading terms (terms of degree two) of the equations now have roots in common. In this situation, Bernstein's theorem is an overestimate. If we used a sparse homotopy method to track the roots, we would find that two of them diverged to infinity. If we compute the sparse resultant of a system of polynomials whose coefficients are dependent, we may find that it is indentically zero. This indicates that there some roots always present, usually at infinity.

This situation is quite common, and perhaps the most striking example is the inverse kinematics of a rotary-joint robot. The simplest way of writing down these equations gives rise to 6 equations of degree 6. The Bernstein bound is about 2,000, while the Bezout bound is 46,000. The number of affine solutions is only 16 [RR89]. The kinematic equations of [RR89] for the same problem give a Bernstein bound of 384.

The large fluctuation in the bounds is caused by the heavy dependencies between the coefficients of the equations. For kinematics, the polynomial system consists of 6 of the elements of the matrix product $A_1(t_1) \cdots A_6(t_6) = B$. Each matrix $A_i(t_i)$ has elements which are quadratic functions of t_i. This would give degree 12 overall, but we can take instead $A_1(t_1)A_2(t_2)A_3(t_3) = BA_6^{-1}(t_6)A_5^{-1}(t_5)A_4^{-1}(t_4)$, because the inverses are also quadratic functions of t_i's. Clearly, the matrix products lead to many common subexpressions in the final polynomials. The dependencies that result reduce the number of solutions, but this is not manifest in the Newton polytopes.

Surprisingly, we can make the structure manifest by *introducing new variables.* We assign these variables to the common subexpressions and replace each occurence of the expression with the corresponding variable. This increases the dimension of the problem, and may or may not reduce the degree. The Bezout bounds will usually increase significantly as a result, but the Bernstein bounds decrease. This runs against the conventional wisdom for dealing with multivariate polynomials, which is to keep the number of variables as low as possible. But by reducing the Bernstein bound, we can use methods based on sparse homotopies or sparse resultants to solve the problem with much better time bounds.

We have applied this idea to a problem in computer vision. The problem, discussed in [FM90], is to determine the camera displacement given the x, y coordinates of a set of points from two views. The equations given there are:

$$(u_2 - u_0)(v_2 - v_1) = (v_2 - v_0)(u_2 - u_1)$$

$$(u_2 e_2 - u_0 e_0)(v_2 e_2' - v_1 e_1') = (v_2 e_2' - v_0 e_0')(u_2 e_2 - u_1 e_1)$$

$$\begin{aligned}(\delta_{01} {u_2}^2 + \delta_{02} {u_1}^2 + 2\delta_0 u_2 u_1)(\delta_2' v_0 v_1 - \delta_{01}' {v_2}^2 - \delta_0' v_1 v_2 - \delta_1' v_0 v_2) = \\ ({\delta_{01}}' {v_2}^2 + {\delta_{02}}' {v_1}^2 + 2\delta_0' v_2 v_1)(\delta_2 u_0 u_1 - \delta_{01} {u_2}^2 - \delta_0 u_1 u_2 - \delta_1 u_0 u_2)\end{aligned}$$

$$\begin{aligned}(\delta_{01} {u_2}^2 + \delta_1 2{u_0}^2 + 2\delta_1 u_2 u_0)(\delta_2' v_0 v_1 - \delta_{01}' {v_2}^2 - \delta_0' v_1 v_2 - \delta_1' v_0 v_2) = \\ ({\delta_{01}}' {v_2}^2 + \delta_1 2p{v_0}^2 + 2\delta_0' v_2 v_0)(\delta_2 u_0 u_1 - \delta_{01} {u_2}^2 - \delta_0 u_1 u_2 - \delta_1 u_0 u_2)\end{aligned}$$

The system is bi-homogeneous in the two sets of variables (u_0, u_1, u_2) and (v_0, v_1, v_2). The main result of the paper [FM90] is that this system has 10 solutions, meaning that there are 10 possiblities for the camera displacement. (In fact this result was proved earlier by Demazure, but the proof was non-elementary, and the proof in [FM90] is straightforward) If we evaluate the Bernstein bound for the system as given, we find that there should be 18 solutions.

But inspection of the equations shows that there are common subexpressions in the last two equations. We can introduce two new variables, x_1 and x_2 to describe these, leading to the apparently equivalent system:

$$(u_2 - u_0)(v_2 - v_1) = (v_2 - v_0)(u_2 - u_1)$$
$$(u_2e_2 - u_0e_0)(v_2e_2' - v_1e_1') = (v_2e_2' - v_0e_0')(u_2e_2 - u_1e_1)$$
$$x_1 = (\delta_2'v_0v_1 - \delta_{01}'v_2^2 - \delta_0'v_1v_2 - \delta_1'v_0v_2)$$
$$x_2 = (\delta_2u_0u_1 - \delta_{01}u_2^2 - \delta_0u_1u_2 - \delta_1u_0u_2)$$
$$(\delta_{01}u_2^2 + \delta_{02}u_1^2 + 2\delta_0u_2u_1)x_1 = (\delta_{01}'v_2^2 + \delta_{02}'v_1^2 + 2\delta_0'v_2v_1)x_2$$
$$(\delta_{01}u_2^2 + \delta_{12}u_0^2 + 2\delta_1u_2u_0)x_1 = (\delta_{01}'v_2^2 + \delta_{12}'v_0^2 + 2\delta_0'v_2v_0)x_2$$

which has a sparse bound of 12, almost optimal. It seems strange that the second system has fewer solutions, but remember that Bernstein's theorem counts roots in $(\mathbb{C}^*)^n$. These solutions have all their coordinates non-zero. By introducing x_1 and x_2, we have forced the corresponding subexpressions to be non-zero for a valid solution. It was the vanishing of these expressions, together with the first two equations, that led to 6 spurious roots in the first system.

This kind of substitution is a useful heuristic for improving bounds, but we would like a better understanding of the phenomenon underlying it. Specifically, we would like to extend the theory of sparse systems from systems $f_1 = \cdots = f_n = 0$ to *compositions of polynomial maps* $F_i : \mathbb{C}^{n_{i-1}} \to \mathbb{C}^{n_i}$, of the form $F_1(F_2(\cdots)) = 0$.

In the example above, the original system can be thought of as a composition of two maps $F_1(F_2())$. The map $F_2 : K^6 \to K^8$ has 8 output values which are $(u_0, u_1, u_2, v_0, v_1, v_2, x_1, x_2)$. The map $F_1 : K^8 \to K^4$ gives, say, the differences between LHS and RHS of the 4 equations. But the equations in F_1 are the reduced equations 5.1 which use x_1 and x_2 instead of the expressions they replace. By performing this decomposition, we have produced polynomial maps F_1 and F_2 with coefficients that are generic, or at least, more likely to be generic. Bounds for this system should be tighter than for the original equations because the dependencies have been removed.

Tight bounds for the number of roots of compositions of maps would have many implications. For example, the kinematics of any mechanism with rotary or sliding joints can be written in that form. The number of solutions for the mechanism could be determined algorithmically. As we have seen, an understanding of sparse bounds often leads to good algorithms for solving the system. If this too were possible for compositions of maps, efficient solvers could be constructed automatically. This is particularly important for new types of complex mechanisms, such as silicon microstructures, which have large numbers of interacting links.

5.2 Pfaffian equations

One of the most important extensions of the theory of the real numbers is to Pfaffian systems. These are systems which satisfy ordinary differential equations in a single dependent variable, time, of the form

$$\dot{X} = A(X)u$$

where X is a vector of states, A is a matrix whose entries are polynomials in X, and u is a vector of control inputs. If $B(X)$ is a matrix whose null rowspace is the column span of A, the system can

be written in equivalent form as

$$B(X)\dot{X} = 0$$

Pfaffian systems include most of the systems studied by control theorists and many dynamic problems in other branches of engineering and physics. The complexity of deciding properties of such systems seems hopeless at first because of the possibility of infinite oscillation, limits cycles etc and chaotic behaviour. But if one places some reasonable constraints on the domain of the state variables, some remarkable properties emerge. This line of work began with the remarkable book by Khovanski [Kho91]. He showed, among other things, that Bezout-like bounds apply to the number of intersections of trajectories ("p-curves") of Pfaffian systems.

Recently there has been progress on extension of the theory of the reals with exponential functions [Ric92], which are the simplest examples of p-curves.

We plan to work on further extensions of the theory of the reals to include Pfaffian constraints, and eventually incorporate this capability into the toolkit.

6 Bibliography

[Ber75] D. N. Bernstein. The number of roots of a system of equations. *(Translated from) Funktsional'nyi Analiz i Ego Prilozheniya*, 9(3):1–4, Jul-Sep 1975.

[Bet92] U. Betke. Mixed volumes of polytopes. *Arch. der Math.*, 58:388–391, 1992.

[BF91] J. Backelin and R. Fröberg. How we proved that there are exactly 924 cyclic 7-roots. In *International Symposium on Symbolic and Algebraic Computation*, pages 103–111, 1991. Bonn.

[BOKR86] M. Ben-Or, D. Kozen, and J. Reif. The complexity of elementary algebra and geometry. *J. Comp. and Sys. Sci.*, 32:251–264, 1986.

[Can88a] J.F. Canny. *The Complexity of Robot Motion Planning.* M.I.T. Press, Cambridge, 1988.

[Can88b] J.F. Canny. Some algebraic and geometric computations in PSPACE. In *ACM Symposium on Theory of Computing*, pages 460–467, 1988.

[Can90] J.F. Canny. Generalized characteristic polynomials. *Journal of Symbolic Computation*, 9(3), 1990.

[Can91a] J.F. Canny. Computing roadmaps of general semi-algebraic sets. In *AAECC-91*, 1991. New Orleans.

[Can91b] J.F. Canny. An improved sign determination algorithm. In *AAECC-91*, 1991. New Orleans.

[Can93a] J. Canny. Computing roadmaps of general semi-algebraic sets. *Computer Journal*, 1993. Special Issue on Quantifier Elimination.

[Can93b] J. Canny. Improved algorithms for sign-determination and existential quantifier elimination. *Computer Journal*, 1993. Special Issue on Quantifier Elimination.

[CE93] J. Canny and I. Emiris. An efficient algorithm for the sparse mixed resultant. In *Conference on Algebraic Algorithms and Error-Correcting Codes*, Puerto Rico, 1993.

[CR91] J.F. Canny and J.M. Rojas. An optimal condition for determining the exact number of roots of a polynomial system. In *International Symposium on Symbolic and Algebraic Computation*, 1991. Bonn, Germany.

[Dix08] A.L. Dixon. The eliminant of three quantics in two independent variables. *Proceedings of London Mathematical Society*, 6:49–69, 209–236, 1908.

[EC93] I. Emiris and J. Canny. A practical algorithm for the sparse mixed resultant. In *International Symposium on Symbolic and Algebraic Computation*, 1993.

[FIKY88] T. S. Freeman, G. Imirzian, E. Kaltofen, and L. Yagati. Dagwood: A system for manipulating polynomials given by straight-line programs. *ACM Trans. Math. Software*, 14(3):218–240, 1988.

[FM90] O. D. Faugeras and S. Maybank. Motion from point matches: Multiplicity of solutions. *International Journal of Computer Vision*, 4(3):225–246, 1990.

[GKZ90] I. M. Gel'fand, M. M. Kapranov, and A. V. Zelevinsky. Discriminants of polynomials in several variables and triangulations of Newton polytopes. *(Translated from) Algebara i Analiz*, 2:1–62, 1990. (English Translation in) *Leningrad Math. J.* 2 (1991).

[GV92] D.Y. Grigor'ev and N.N. Vorobjov. Counting connected components of a semialgebraic set in subexponential time. *Computational Complexity*, 2:133–186, 1992.

[Hof90] C.M. Hoffmann. Algebraic and numeric techniques for offsets and blends. In W. Dahmen, M. Gasca, and C. Micchelli, editors, *Computations of Curves and Surfaces*, pages 499–528. Kluwer Academic Publishers, 1990.

[HRS90] J. Heintz, M.F. Roy, and P. Solerno. Complexité du principe de Tarski-Seidenberg. *Bull. Soc. Math. France*, 118:101–126, 1990.

[HS92] Birkett Huber and Bernd Sturmfels. Homotopies preserving the newton polytopes. Cornell University, manuscript, 1992.

[Kal88] Erich Kaltofen. Greatest common divisors of polynomials given by straight-line programs. *Journal of the ACM*, 35(1):231–264, 1988.

[Kal89] Erich Kaltofen. Factorization of polynomials given by straight-line programs. In S. Micali, editor, *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 375–412. JAI Press, Greenwhich, Connecticut, 1989.

[Kan92] J-M. Kantor. Sur le polynôme associé à un polytope à sommets entiers dans r^n. *C. R. Acad. Sci. Paris*, 314:669–672, 1992.

[Kho78] A.G. Khovanskii. Newton polyhedra and the genus of complete intersections. *(Translated from) Funktsional'nyi Analiz i Ego Prilozheniya*, 12(1):51–61, Jan-Mar 1978.

[Kho91] A. G. Khovanskii. *Fewnomials.* AMS Press, Providence, Rhode Island, 1991.

[Kus76] A.G. Kushnirenko. Newton polytopes and the Bezout theorem. *(Translated from) Funktsional'nyi Analiz i Ego Prilozheniya*, 10(3), Jul-Sep 1976.

[LL88] H.Y. Lee and C.G. Liang. A new vector theory for the analysis of spatial mechanisms. *Mechanisms and Machine Theory*, 23(3):209–217, 1988.

[Mac02] F.S. Macaulay. On some formula in elimination. *Proceedings of London Mathematical Society*, pages 3–27, May 1902.

[MC91a] D. Manocha and J. Canny. Efficient techniques for multipolynomial resultant algorithms. In *ISSAC-91*, 1991. Bonn, Germany.

[MC91b] D. Manocha and J.F. Canny. A new approach for surface intersection. *International Journal of Computational Geometry and Applications*, 1(4):491–516, 1991. Special issue on Solid Modeling.

[MC92a] D. Manocha and J.F. Canny. The implicit representation of rational parametric surfaces. *Journal of Symbolic Computation*, 13:485–510, 1992.

[MC92b] D. Manocha and J.F. Canny. Real time inverse kinematics of general 6R manipulators. In *IEEE Conference on Robotics and Automation*, pages 383–389, 1992.

[MC93] D. Manocha and J. Canny. Multipolynomial resultant algorithms. *Journal of Symbolic Computation*, 15(2):99–122, 1993.

[Mer92] J-P. Merlet. Direct kinematics and assembly modes of parallel manipulators. *International Journal of Robotics Research*, 11(2):150–162, 1992.

[MS87] A.P. Morgan and A.J. Sommese. A homotopy for solving general polynomial systems that respects m-homogeneous structures. *Applied Mathematics and Computations*, 24:101–113, 1987.

[Owe91] J. C. Owen. Algebraic solution for geometry from dimensional constraints. In J. Rossignac and J. Turner, editors, *Symp. on Solid Modeling Foundations and CAD/CAM Applications*, pages 397–407. ACM Press, 1991.

[Pri91] Doug Priest. Algorithms for arbitrary precision floating point arithmetic. In *Proc. 10th Symp. on Computer Arithmetic*, 1991. edited by K. Kornerup and D. Matula.

[PS91] P. Pedersen and B. Sturmfels. Product formulas for sparse resultants. Manuscript, 1991.

[Ren89] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals, parts I, II and III. Technical Report 852,855,856, Cornell University, Operations Research Dept., 1989.

[Ren92] J. Renegar. On the computational complexity of the first-order theory of the reals, parts I, II, III. *Journal of Symbolic Computation*, 13(3):255–352, 1992.

[Ric92] Dan Richardson. Computing the topology of a bounded non-algebraic curve in the plane. *Journal of Symbolic Computation*, 14(6), 1992.

[RR89] M. Raghavan and B. Roth. Kinematic analysis of the 6r manipulator of general geometry. In *International Symposium on Robotics Research*, pages 314–320, Tokyo, 1989.

[Stu92] B. Sturmfels. Sparse systems of polynomial equations. Lecture Notes, NSF Regional Geometry Institute, Amherst College, Amherst, Mass., July 1992.

[SV89] M.W. Spong and M. Vidyasagar. *Robot Dynamics and Control.* John Wiley and Sons, 1989.

[SZ93] B. Sturmfels and A. Zelevinsky. Multigraded resultants of sylvester type. *Journal of Algebra*, 1993. to appear.

[TM85] L.W. Tsai and A.P. Morgan. Solving the kinematics of the most general six and five-degree-of-freedom manipulators by continuation methods. *Transactions of the ASME, Journal of Mechanisms, Transmissions and Automation in Design*, 107:189–200, 1985.

[VC92] J. Verschelde and R. Cools. Symbolic homotopy construction. manuscript, 1992.

7 Appendix: Mixed Subdivisions

Let Q denote the Minkowski Sum of all the Q_i:

$$Q = Q_1 + Q_2 + \cdots + Q_{n+1} \subset \mathbb{R}^n$$

Define an $(n+1)$-argument vector sum $\oplus : (\mathbb{R}^n)^{(n+1)} \to \mathbb{R}^n$ as $(p_1, \ldots, p_{n+1}) \mapsto p_1 + \cdots + p_{n+1}$, where $p_i \in \mathbb{R}^n$. Q may be thought of as the image of $Q_1 \times \cdots \times Q_{n+1}$ under $\oplus$. This is clearly a many-to-one mapping, but it is desirable to define a unique inverse by regularization. That is, for each $q \in Q$ we choose a unique $(p_1, \ldots, p_{n+1})$ in $\oplus^{-1}(q) \cap Q_1 \times \cdots \times Q_{n+1}$. To achieve this, the method outlined in [Stu92] is employed.

Choose $n+1$ generic linear forms $l_1, \ldots, l_{n+1} \in \mathbf{Z}[x_1, \ldots, x_n]$. Then the regularized inverse under $\oplus$ of of $q \in Q$ is the point $p = (p_1, \ldots, p_{n+1}) \in Q_1 \times \cdots \times Q_{n+1}$ minimizes

$$l(p) = \sum_{i=1}^{n+1} l_i(p_i)$$

There is also a geometric interpretation of the inverse. Define, for $1 \leq i \leq n+1$, *lifted* Newton polytopes

$$\hat{Q}_i \triangleq \{(p_i, l_i(p_i)) : p_i \in Q_i\} \subset \mathbb{R}^{n+1}.$$

Let the Minkowski Sum of the lifted Newton polytopes be

$$\hat{Q} = \hat{Q}_1 + \cdots + \hat{Q}_{n+1} \subset \mathbb{R}^{n+1}.$$

The $(n+1)$-st coordinate of a point in $\hat{Q}_i$ is to be interpreted as the cost of using that particular point in the vector sum, that is, as the value of $l(p)$. To minimize $l(p)$, we simply choose the lowest point in $\hat{Q}$ which lies over q.

Let $\pi : \mathbb{R}^{n+1} \to \mathbb{R}^n$ denote projection on the first n coordinates, and $h : \mathbb{R}^{n+1} \to \mathbb{R}$ denote projection on the $(n+1)$st. Let $s : \mathbb{R}^n \to \mathbb{R}^{n+1}$ map points in Q to points on the lower envelope of $\hat{Q}$ above them:

$$s(q) \;=\; \hat{q} \in \pi^{-1}(q) \cap \hat{Q} \text{ such that } h(\hat{q}) \text{ is minimized}$$

The lower envelope of $\hat{Q}$ is then $s(Q)$. For generic choices of l_i's every point $\hat{q}$ on the lower envelope can be *uniquely* expressed as a sum of points $\hat{q}_1 + \cdots + \hat{q}_{n+1}$ with $\hat{q}_i \in \hat{Q}_i$. See [Stu92] or [Bet92] for an explanation.

Let $\hat{\Delta}$ denote the natural (coarsest) polyhedral subdivision of the lower envelope of $\hat{Q}$. Each facet (n-dimensional face) of $\hat{\Delta}$ is a Minkowski sum $\hat{F}_1 + \cdots + \hat{F}_{n+1}$ with $\hat{F}_i$ a face of $\hat{Q}_i$, and because lower envelope points have unique expressions as sums,

$$\sum_{i=1}^{n+1} \dim(\hat{F}_i) = n$$

The image of $\hat{\Delta}$ under π induces a polyhedral subdivision Δ of Q.

Definition 7.1 *The subdivision Δ is a* mixed subdivision *of the Minkowski sum Q.*

The facets of Δ are of the form $F_1 + \cdots + F_{n+1}$ with the same dimension property as $\hat{\Delta}$, a corollary of which is the following:

Observation For every facet $F = F_1 + \cdots + F_{n+1}$ in Δ, F_i a face of Q_i, at least one of the F_i is zero-dimensional, i.e. a vertex.

Definition 7.2 A *mixed facet* of the induced subdivision is a facet which is a sum $F_1 + \cdots + F_{n+1}$ where *exactly one* F_i is a vertex. Thus the remaining F_j for $j \neq i$ are edges.