Computer Algebra: General Principles

For article on related subject see SYMBOL MANIPULATION.

Computer algebra is a branch of scientific computation. There are several characteristic features that distinguish computer algebra from numerical analysis, the other principal branch of scientific computation. (1) Computer algebra involves computation in algebraic structures such as finitely presented groups, polynomial rings, rational function fields, algebraic and transcendental extensions of the rational numbers, or differential and difference fields. (2) Computer algebra manipulates formulas. Whereas in numerical computation the input and output of algorithms are basically (integer or floating point) numbers, the input and output of computer algebra algorithms are generally formulas. So, typically, instead of computing

$$\int_0^{1/2} \frac{x}{x^2-1} dx = -0.1438... ,$$

an integration algorithm in computer algebra yields

$$\int \frac{x}{x^2-1} dx = \frac{\ln \mid x^2-1 \mid}{2}.$$

(3) Computations in computer algebra are carried through exactly, i.e. no approximations are applied at any step. So, typically, the solutions of a system of algebraic equations such as

$$x^4 + 2x^2y^2 + 3x^2y + y^4 - y^3 = 0$$
$$x^2 + y^2 - 1 = 0$$

are presented as $(0,1), (\pm\sqrt{3/4}, -1/2)$ instead of $(0,1), (\pm 0.86602..., -0.5)$. Because of the exact nature of the computations in computer algebra, decision procedures can be derived from such algorithms that decide, for example, the solvability of systems of algebraic equations, the solvability of integration problems in a specified class of formulas, or the validity of geometric formulas.

Applications of computer algebra

(1) The piano movers problem. Many problems in robotics can be modelled by the piano movers problem: finding a path that will take a given body B from a given initial position to a desired final position. The additional constraint is that along the path the body should not hit any obstacles such as walls or other bodies. A simple example in the plane is shown in Figure 1. The initial and final positions of the body B are drawn in full, whereas a possible intermediate position is drawn in dotted lines. J.T. Schwartz and M. Sharir have shown how to reduce this problem to a certain problem about semialgebraic sets that can be solved by Collins' cylindrical algebraic decomposition (cad) method.

Semialgebraic sets are subsets of a real m–dimensional space $\mathbb{R}^m$ that can be cut out by polynomial equations and inequations. I.e. start with simple sets of the form

$$\{(x_1, \ldots, x_m) \mid p(x_1, \ldots, x_m) = 0\} \quad \text{or} \quad \{(x_1, \ldots, x_m) \mid q(x_1, \ldots, x_m) > 0\},$$

where p, q are polynomials with real coefficients, and allow the construction of more complicated sets by means of intersection, union, and difference. Any subset of $\mathbb{R}^m$ that can be defined in this way is called a semialgebraic set.

Fig. 1 Fig. 2

Consider a two–dimensional problem as in Figure 1. Starting from some fixed position of the body B (say P at the origin, where P is the point at which the parts of B are joined together) in $\mathbb{R}^2$, obtain an arbitrary position of B by applying a rotation T_1 to part B_2, a rotation T_2 to B, and afterwards a translation T_3 to B. (See Figure 2.) Since T_1, T_2 can be described by 2×2–matrices and T_3 by a vector of length 2, any such position of B can be specified by 10 coefficients, i.e. a point in $\mathbb{R}^{10}$. Some of these possible positions are illegal, since the body B would intersect or lie outside of the boundaries. If the legal positions $L(\subset \mathbb{R}^{10})$ can be described by polynomial equations and inequations, then L is a semialgebraic set.

The piano movers problem is now reduced to the question of whether two points P_1, P_2 in L can be joined by a path in L, i.e. whether P_1 and P_2 lie in the same connected component of L. This question can be decided by Collins' cad method, which makes heavy use of computer algebra algorithms. In particular, the cad method uses algorithms for greatest common divisors of polynomials, factorization of polynomials into square–free factors, resultant computations, and isolation of real roots of polynomials.

(2) Algorithmic methods in geometry. Often a geometric statement can be described by polynomial equations over some ground field K, such as the real or complex numbers. Consider, for instance, the statement *"The altitude pedal of the hypothenuse of a right-angled triangle and the midpoints of the three sides of the triangle lie on a circle"* (see Figure 3).

Once the geometric figure is placed into a coordinate system, it can be described by polynomial equations. For instance, the fact that E is the midpoint of the side AC is expressed by the equation $2y_3 - y_1 = 0$; the fact that the line segments EM and FM are of equal length is expressed by the equation $(y_7 - y_3)^2 + y_8^2 - (y_7 - y_4)^2 - (y_8 - y_5)^2 = 0$; and so on. In this way the system $h_1 = \ldots = h_m = 0$ of polynomial equations in the indeterminates $y_1, \ldots, y_n$ determines the geometric figure. Call these polynomials the *hypothesis polynomials*. The equation $(y_7 - y_3)^2 + y_8^2 - (y_7 - y_9)^2 - (y_8 - y_{10})^2 = 0$ then states that the line segments HM and EM are also of equal length. Call this polynomial

the *conclusion polynomial*.

The problem of proving the geometric statement is now reduced to the problem of proving that every common solution of the hypothesis polynomials, i.e. every valid geometric configuration, also solves the conclusion polynomial, i.e. the statement is valid for the configuration. Various computer algebra methods can be used for proving such geometry statement. Wu Wen-tsun has given a method using characteristic sets of polynomials, Kutzler&Stifter and Kapur have used Gröbner bases. The underlying computer algebra algorithms for these methods are mainly the solution of systems of polynomial equations, various decision algorithms in the theory of polynomial ideals, and algorithms for computing in algebraic extensions of the field of rational numbers.

Fig. 3

(3) Modelling in science and engineering. In science and engineering, it is common to express a problem in terms of integrals or differential equations with boundary conditions. Numerical integration leads to approximations of the values of the solution functions. But, as R.W. Hammings has written, "the purpose of computing is insight, not numbers." So instead of computing tables of values it would be much more gratifying to derive formulas for the solution functions. Computer algebra algorithms can do just that for certain classes of integration and differential equation problems.

Consider, for example, the system of differential equations

$$-6\frac{dq}{dx}(x) + \frac{d^2p}{dx^2}(x) - 6\sin(x) = 0,$$
$$6\frac{d^2q}{dx^2}(x) + a^2\frac{dp}{dx}(x) - 6\cos(x) = 0$$

subject to the boundary conditions $p(0) = 0, q(0) = 1, p'(0) = 0, q'(0) = 0$. Given this information as input, any of the major computer algebra systems will derive the formal solution

$$p(x) = -\frac{12\sin(ax)}{a(a^2-1)} - \frac{6\cos(ax)}{a^2} + \frac{12\sin(x)}{a^2-1} + \frac{6}{a^2},$$
$$q(x) = \frac{\sin(ax)}{a} - \frac{2\cos(ax)}{a^2-1} + \frac{(a^2+1)\cos(x)}{a^2-1}$$

for $a \notin \{-1, 0, 1\}$.

Some algorithms in computer algebra

Since computer algebra algorithms must yield exact results, these algorithms use integers and rational numbers as coefficients of algebraic expressions because these numbers can be represented exactly in the computer. Coefficients may also be algebraic.

Addition and subtraction of integers are quite straightforward and these operations can be performed in time linear in the length of the numbers. The classical algorithm for multiplication of integers x and y proceeds by multiplying every digit of x by every digit of y and adding the results after appropriate shifts. This clearly takes time quadratic in the length of the inputs. A faster multiplication algorithm due to A. Karatsuba and Yu. Ofman is usually called the Karatsuba algorithm. The basic idea is to cut the two inputs x, y of length $\leq n$ into pieces of length $\leq n/2$ such that

$$x = a \cdot \beta^{n/2} + b, \quad y = c \cdot \beta^{n/2} + d,$$

where β is the basis of the number system. A usual divide–and–conquer approach would reduce the multiplication of two integers of length n to four multiplications of integers of length $n/2$ and some subsequent shifts and additions. The complexity of this algorithm would still be quadratic in n. However, from

$$x \cdot y = ac\beta^n + ((a+b)(c+d) - ac - bd)\beta^{n/2} + bd$$

we see that one of the four multiplications can be replaced by additions and shifts, which take only linear time. If this reduction of the problem is applied recursively, we get a multiplication algorithm with a time complexity proportional to $n^{\log_2 3}$. This is still not the best we can hope for. In fact, the fastest known algorithm is due to Schönhage and Strassen and its complexity is proportional to $n(\log n)(\log\log n)$. However, the overhead of this algorithm is enormous, and it pays off only if the numbers are incredibly large.

Polynomial arithmetic with coefficients in a field, like the rational numbers, presents no problem. These polynomials form a Euclidean domain, so we can carry out division with quotient and remainder. Often, however, we need to work with polynomials whose coefficients lie in an integral domain like the integers. Addition, subtraction and multiplication are again obvious, but division with quotient and remainder is not possible. Fortunately, we can replace division by a similar process, called pseudo–division. If $a(x) = a_m x^m + \cdots + a_1 x + a_0$ and $b(x) = b_n x^n + \cdots + b_1 x + b_0$, with $m \geq n$, then there exists a unique pair of quotient $q(x)$ and remainder $r(x)$ such that $b_n^{m-n+1} \cdot a(x) = q(x)b(x) + r(x)$ where r is either the zero polynomial or the degree of r is less than the degree of b.

Good algorithms are needed for computing the greatest common divisor (gcd) of polynomials. If we are working with polynomials over a field, we can use Euclid's algorithm, which takes two polynomials $f_1(x), f_2(x)$ and computes a chain of remainders $f_3(x), \ldots, f_k(x), f_{k+1}(x) = 0$, such that f_i is the remainder of dividing f_{i-2} by f_{i-1}. Then $f_k(x)$ is the desired greatest common divisor. For polynomials over the integers we can replace division by pseudo–division, and the Euclidean algorithm still works. The problem, however, is that although the inputs and the final result might be quite small, the intermediate polynomials can have huge coefficients. This problem becomes even more pronounced if we deal with multivariate polynomials. As an example, consider the computation of the greatest common divisor of two bivariate polynomials

$$f(x,y) = y^6 + xy^5 + x^3y - xy + x^4 - x^2, \quad g(x,y) = xy^5 - 2y^5 + x^2y^4 - 2xy^4 + xy^2 + x^2y$$

with integral coefficients. Consider y to be the main variable, so that the coefficients of powers of y are polynomials in x. Euclid's algorithm yields the polynomial remainder sequence

$$\begin{aligned}
r_0 =& f,\\
r_1 =& g,\\
r_2 =& (2x - x^2)y^3 + (2x^2 - x^3)y^2 + (x^5 - 4x^4 + 3x^3 + 4x^2 - 4x)y +\\
& x^6 - 4x^5 + 3x^4 + 4x^3 - 4x^2,\\
r_3 =& (-x^7 + 6x^6 - 12x^5 + 8x^4)y^2 + (-x^{13} + 12x^{12} - 58x^{11} + 136x^{10} - 121x^9 - 117x^8 +\\
& 362x^7 - 236x^6 - 104x^5 + 192x^4 - 64x^3)y - x^{14} + 12x^{13} - 58x^{12} + 136x^{11} -\\
& 121x^{10} - 116x^9 + 356x^8 - 224x^7 - 112x^6 + 192x^5 - 64x^4,\\
r_4 =& (-x^{28} + 26x^{27} - 308x^{26} + 2184x^{25} - 10198x^{24} + 32188x^{23} - 65932x^{22} + 68536x^{21} +\\
& 42431x^{20} - 274533x^{19} + 411512x^{18} - 149025x^{17} - 431200x^{16} + 729296x^{15} -\\
& 337472x^{14} - 318304x^{13} + 523264x^{12} - 225280x^{11} - 78848x^{10} + 126720x^9 -\\
& 53248x^8 + 8192x^7)y - x^{29} + 26x^{28} - 308x^{27} + 2184x^{26} - 10198x^{25} + 32188x^{24} -\\
& 65932x^{23} + 68536x^{22} + 42431x^{21} - 274533x^{20} + 411512x^{19} - 149025x^{18} -\\
& 431200x^{17} + 729296x^{16} - 337472x^{15} - 318304x^{14} + 523264x^{13} - 225280x^{12} -\\
& 78848x^{11} + 126720x^{10} - 53248x^9 + 8192x^8.
\end{aligned}$$

The greatest common divisor of f and g is obtained by eliminating common factors $p(x)$ in r_4. The final result is $y+x$. Although the inputs and the output are small, the intermediate expressions get very big. The biggest polynomial in this computation happens to occur in the pseudo–division of r_3 by r_4. The intermediate polynomial has degree 70 in x.

This problem of coefficient growth is ubiquitous in computer algebra, and there are some general approaches for dealing with it. In the special case of polynomial gcd's, we could always make the polynomials <u>primitive</u>, i.e., eliminate common factors not depending on the main variable. This approach keeps intermediate remainders as small as possible, but at a high price: many gcd computations on the coefficients. The subresultant gcd algorithm can determine many of the common factors of the coefficients without ever computing gcd's of coefficients. The remainders stay reasonably small during this algorithm. In fact, in our example the integer coefficients grow only to length 4.

The most efficient algorithm for computing gcd's of multivariate polynomials is the <u>modular algorithm.</u> The basic idea is to apply homomorphisms to the coefficients, compute the gcd's of the evaluated polynomials, and use the <u>Chinese remainder algorithm</u> to reconstruct the actual coefficients in the gcd. If the input polynomials are univariate, we can take homomorphisms H_p, mapping an integer a to $a \bmod p$. If the input polynomials are multivariate, we can take evaluation homomorphisms of the form $H_{x_1=r_1}$ for reducing the number of variables. In our example we get

$$\gcd(H_{x=2}(f), H_{x=2}(g)) = y + 2, \quad \gcd(H_{x=3}(f), H_{x=3}(g)) = y + 3.$$

So the gcd is $y + x$. Never during this algorithm did we have to consider big coefficients.

Decomposing polynomials into irreducible factors is another crucial algorithm in computer algebra. A few decades ago only rather inefficient techniques for polynomial factorization were available. Research in computer algebra has contributed to a deeper understanding of the problem and as a result has created much better algorithms. Let us

first consider univariate polynomials with integer coefficients. Since the problem of coefficient growth appears again, one usually maps the polynomial $f(x)$ to a polynomial $f_{(p)}(x)$ by applying a homomorphism H_p, p a prime. $f_{(p)}$ can now be factored by the Berlekamp algorithm, which involves some linear algebra and computations of gcd's. Conceivably we could factor f modulo various primes $p_1, \ldots, p_k$ and try to reconstruct the factors over the integers by the Chinese remainder algorithm, as we did in the modular gcd algorithm. The problem is that we do not know which factors correspond. So instead, one uses a p–adic approach based on Hensel's lemma which states that a factorization of f modulo a prime p can be lifted to a factorization of f modulo p^k, for any positive integer k. Since we know bounds for the size of the coefficients that can occur in the factors, we can determine a suitable k and thus construct the correct coefficients of the integral factors. There is, however, an additional twist. If $f(x)$ can be decomposed into irreducible factors $f_1(x), f_2(x)$ over the integers, it could well be that, modulo p, these irreducible factors can be split even further. So after we have lifted the factorization modulo p to a factorization modulo p^k for a suitable k, we need to try combinations of factors for determining the factors over the integers. For instance, $x^4 + 1$ is a polynomial that is irreducible over the integers, but factors modulo every prime. Theoretically, this final step is the most costly one and it makes the time complexity of the Berlekamp–Hensel algorithm exponential in the degree of the input. Nevertheless, in practice the algorithm works very well for most examples.

In 1982 Lenstra, Lenstra and Lovász developed an algorithm for factoring univariate polynomials over the integers with a polynomial time complexity. Kaltofen extended this result to multivariate polynomials. The overhead of this algorithm, however, is extremely high.

To integrate a rational function $A(x)/B(x)$, where A, B are polynomials with integral coefficients, we could split the polynomial B into linear factors in a suitable algebraic extension field, compute a partial fraction decomposition of the integrand, and integrate all the summands in this decomposition. The summands with linear denominators lead to logarithmic parts in the integral. Computations in the splitting field of a polynomial are very expensive; if n is the degree of the polynomial, the necessary algebraic extension has degree $n!$. So the question arises whether it is really necessary to go to the full splitting field. For instance,

$$\int \frac{x}{x^2-2}dx = \int \frac{1/2}{x-\sqrt{2}}dx + \int \frac{1/2}{x+\sqrt{2}}dx =$$
$$\frac{1}{2}[\log(x-\sqrt{2}) + \log(x+\sqrt{2})] = \frac{1}{2}\log(x^2-2).$$

The example shows that although we had to compute in the splitting field of the denominator, the algebraic extensions actually disappear in the end. A deeper analysis of the problem reveals that instead of factoring the denominator into linear factors it suffices to compute a so–called square–free factorization, i.e. a decomposition of a polynomial f into $f = f_1 \cdot f_2^2 \cdot \ldots \cdot f_r^r$, where the factors f_i are pairwise relatively prime and have no multiple roots (square–free). The square–free factorization can be computed by successive gcd operations. Now if A and B are relatively prime polynomials over the rational numbers, B is square–free, and the degree of A is less than the degree of B, then

$$\int \frac{A(x)}{B(x)}dx = \sum_{i=1}^{n} c_i \log v_i,$$

where the $c_1, \ldots, c_n$ are the distinct roots of the resultant of $A(x) - c \cdot B'(x)$ and $B(x)$ w.r.t. x, and each v_i is the gcd of $A(x) - c_i \cdot B'(x)$ and $B(x)$. In this way we get the smallest field extension necessary for expressing the integral.

The problem of integration becomes more complicated if the class of integrands is extended. A very common class is that of elementary functions. We get this class by starting with the rational functions and successively adding exponentials ($\exp f(x)$), logarithms ($\log f(x)$) or roots of algebraic equations, where the exponents, arguments, or coefficients are previously constructed elementary functions. Not every elementary integrand has an elementary integral, e.g. $\int e^{x^2} dx$ cannot be expressed as an elementary function. However, there is an algorithm, the Risch algorithm, that can decide whether a given integrand can be integrated in terms of elementary functions, and if so the Risch algorithm yields the integral. The case of algebraic functions is the most complicated part of the Risch algorithm. For a thorough introduction to the algebraic integration problem the reader is referred to the paper by M. Bronstein in the *Journal of Symbolic Computation*, Vol.9.

The discrete analog to the integration problem is the problem of summation in finite terms. We are given an expression for a summand a_n and we want to compute a closed expression for the partial sums of the infinite series $\sum_{n=1}^{\infty} a_n$. That is, we want to compute a function $S(m)$, such that

$$\sum_{n=1}^{m} a_n = S(m) - S(0).$$

For instance, we want to compute

$$\sum_{n=1}^{m} n \cdot x^n = \frac{mx^{m+2} - (m+1)x^{m+1} + x}{(x-1)^2}.$$

For the case of hypergeometric functions, Gosper's algorithm solves this problem. There is also a theory of summation similar to the theory of integration in finite terms.

Gröbner bases are an extremely powerful method for deciding many problems in the theory of polynomial ideals. As an example, consider the system of algebraic equations

$$\begin{gathered} 2x^4 + y^4 + 8x^3 - 3x^2y - 2y^3 + 12x^2 - 6xy + y^2 + 8x - 3y + 2 = 0 \\ 8x^3 + 24x^2 - 6xy + 24x - 6y + 8 = 0 \\ 4y^3 - 3x^2 - 6y^2 - 6x + 2y - 3 = 0. \end{gathered} \tag{1}$$

Every root of these equations is also a root of any linear combination of these equations, so in fact we are looking for zeros of the ideal generated by the left hand sides in the ring of polynomials in x and y over $\mathbb{Q}$. The left hand sides form a specific basis of this ideal. The goal is to compute another basis for this same ideal that is better suited for solving the system. Such a basis is a Gröbner basis with respect to a lexicographic ordering of the variables. In our example we get the following Gröbner basis, which we again write as a system of equations.

$$\begin{gathered} y^3 - y^2 = 0 \\ yx + y = 0 \\ 3x^2 + 2y^2 + 6x - 2y + 3 = 0. \end{gathered} \tag{2}$$

The solutions of (1) and (2) are the same, but obviously it is much easier to investigate the solutions of (2). The system contains a polynomial depending only on y, and the zeros

are $y = 0$ and $y = 1$. Substituting these values for y into the other two equations, we get the solutions $(x = -1, y = 0)$ and $(x = -1, y = 1)$ for the system of algebraic equations.

Other problems in the theory of polynomial ideals that can be solved by Gröbner bases include the ideal membership problem, the radical membership problem, the primary decomposition of an ideal, or the computation of the dimension of an ideal. Most computer algebra programs contain a Gröbner basis package.

Representation of expressions

Dynamic data structures are necessary for representing the computational objects of computer algebra in the memory of the computer. For instance, during the execution of the Euclidean algorithm, the coefficients in the polynomials expand and shrink again. Since the goal of the computation is an exact result, we cannot just truncate them to the most significant positions.

Most computer algebra programs represent objects as lists. An integer is represented as a list of digits. For more complicated objects, the choice of representation is not that clear. So, for instance, we can represent a bivariate polynomial recursively as a polynomial in a main variable with coefficients in a univariate polynomial ring, or distributively as pairs of coefficients and power products in the variables.

recursive representation: $p(x, y) = (3x^2 - 2x + 1)y^2 + (x^2 - 3x)y + (2x + 1)$

distributive representation: $p(x, y) = 3x^2y^2 - 2xy^2 + x^2y + y^2 - 3xy + 2x + 1$

For both these representations we can use a dense or a sparse list representation. In the dense representation, a polynomial is a list of coefficients, starting from some highest coefficient down to the constant coefficient. So the dense recursive representation of p is

$$(\ (3\ -2\ 1)\ (1\ -3\ 0)\ (2\ 1)\ ).$$

For the dense distributive representation of p we order the power products according to the degree and lexicographically within the same degree. So p is represented as

$$\begin{array}{cccccccccccccc} (& 3 & 0 & 0 & 0 & -2 & 1 & 0 & 1 & -3 & 0 & 0 & 2 & 1 &). \\ & x^2y^2 & x^3y & x^4 & y^3 & xy^2 & x^2y & x^3 & y^2 & xy & x^2 & y & x & 1 & \end{array}$$

If only few power products have a coefficient different from 0, then a dense representation wastes a lot of space. In this case we really want to represent the polynomial sparsely, i.e. by pairs of coefficients and exponents. The sparse recursive representation of p is

$$(\ (((3\,2)\,(-2\,1)\,(1\,0))\ 2)\ (((1\,2)\,(-3\,1))\ 1)\ (((2\,1)\,(1\,0))\ 0)),$$

and the sparse distributive representation of p is

$$((3\,(2\,2))\ (-2\,(1\,2))\ (1\,(2\,1))\ (1\,(0\,2))\ (-3\,(1\,1))\ (2\,(1\,0))\ (1\,(0\,0))).$$

For different algorithms, different representations of the objects are useful or even necessary. The multivariate gcd algorithm works best with polynomials given in recursive representation, whereas the Gröbner basis algorithm needs the input in distributive representation. So in general, a computer algebra program has to provide many different

representations for the various algebraic objects and transformations that convert one form to another.

References

1981. Knuth, D.E. *The Art of Computer Programming, Vol. 2*, 2nd ed., Addison-Wesley, Reading, Massachusetts.

1983. Buchberger, B., Collins, G.E., Loos, R. (eds.) *Computer Algebra — Symbolic and Algebraic Computation*, 2nd ed., Springer-Verlag, Wien–New York.

1988. Davenport, J.H., Siret, Y., Tournier, E. *Computer Algebra — Systems and Algorithms for Algebraic Computation*, Academic Press, London.

1989. Akritas, A.G. *Elements of Computer Algebra*, Wiley, New York.

Journal of Symbolic Computation. Published by Academic Press, London.

F. Winkler