Article Submitted to Journal of Symbolic Computation

Challenges of Symbolic Computation My Favorite Open Problems*

ERICH KALTOFEN †
DEPARTMENT OF MATHEMATICS
NORTH CAROLINA STATE UNIVERSITY
RALEIGH, N.C. 27695-8205, U.S.A.

With an Additional Open Problem By

ROBERT M. CORLESS AND DAVID J. JEFFREY
DEPARTMENT OF APPLIED MATHEMATICS
UNIVERSITY OF WESTERN ONTARIO
LONDON, ONTARIO N6A 5B7, CANADA

To Bobby F. Caviness
on the occasion of his 60th birthday

Abstract

The success of the symbolic mathematical computation discipline is striking. The theoretical advances have been continuous and significant: Gröbner bases, the Risch integration algorithm, integer lattice basis reduction, hypergeometric summation algorithms, etc. From the beginning in the early 60s, it has been the tradition of our discipline to create software that makes our ideas readily available to scientists, engineers, and educators: SAC-1, Reduce, Macsyma, etc. The commercial viability of our system products is proven by Maple and Mathematica.

Today's user communities of symbolic computation systems are diverse: educators, engineers, stock market analysts, etc. The mathematics and computer science in the design and implementation of our algorithms are sophisticated. The research challenges in symbolic computation at the close of the 20th century are formidable.

I state my favorite eight open problems in symbolic computation. They range

*This material is based on work supported in part by the National Science Foundation under Grant No. CCR-9712267.

†URL: `http://www.kaltofen.net` E-mail: `kaltofen@math.ncsu.edu`

from problems in symbolic/numeric computing, symbolic algorithm synthesis, to system component construction. I have worked on seven of my problems and borrowed one from George Collins. I present background to each of my problems and a clear-cut test that evaluates whether a proposed attack has solved one of my problems. An additional ninth open problem by Rob Corless and David Jeffrey on complex function semantics is given in an appendix.

Introduction

At the *Fifth East Coast Computer Algebra Day*, which was held at the Unites States Naval Academy in Annapolis, Maryland, on April 25, 1998, I gave a one hour lecture of the same title. I repeated this lecture at the *IMACS Conference on Applications of Computer Algebra*, which was held in Prague, Czech Republic, on August 9–11, 1998. In this companion paper I have written up my favorite open problems of symbolic computation and provided a more in-depth discussion with references to the literature. The selection of open problems is my personal one and is not intended to be comprehensive of the field. I am leaving out major areas of investigation, among them differential and difference equations, types of domains in symbolic programming languages, computational group theory, or mathematics on the Internet. In an appendix to this paper, R. Corless and D. Jeffrey state an additional open problem, which was presented by Corless in his lecture at the same Fifth East Coast Computer Algebra Day in April 1998.

A Brief History of Symbolic Mathematical Computation

It is dangerous to stereotype historical development into periods. The following highlights during the decades of symbolic computation, as I perceive them, should simply be taken as a guideline.

- 1960s: pioneering years: polynomial arithmetic, integration
- 1970s: Macsyma and Reduce; abstract domains: Scratchpad/II
- 1980s: polynomial-time methods: factorization; Maple; user interfaces: Mathematica
- 1990s: teaching of calculus; math on the web
- 2000s: merging of symbolic, numeric, geometric, combinatoric, and logic paradigm (?)

In the past 40 years or so the discipline of symbolic computation has made major contributions to science. Collins (1960) pioneers the process of automatic garbage collection by reference counts. New efficient multivariate polynomial greatest common divisor algorithms (see (Brown, 1971) and the references given there) are crucial for the implementation of symbolic algebra. Risch (1969) shows that the problem of finding integrals of mathematical functions in closed form is decidable. Randomization is used by Berlekamp (1970) to efficiently factor polynomials modulo large

prime numbers before the now-famous randomized primality tests. Generic programming is invented in the first half of the 1970s as a means to reuse the code of algebraic algorithms over abstract domains, such as Gaussian elimination (see Section 7). Gosper in 1978 invents an ingenious algorithm for indefinite hypergeometric summation (see (Petkovšek et al., 1996)). Lovász's lattice reduction algorithm, a far-reaching generalization of the Euclidean algorithm, appears first as a substep for polynomial factorization (Lenstra et al., 1982). Interpolation algorithms for sparse multivariate polynomials, some of which are based on error-correcting coding, revise a numerical computation subject that is over 100 years old (see (Grigoriev and Lakshman Y. N., 1995) and the references given there). Today, the mathematical markup for Internet documents exposes several new issues, such as the structuring of compound objects for display and selection.† Last, but not least, we must mention the breakthrough algorithms for computing a Gröbner basis, which are discussed further in Section 5, and for solving a sparse linear system over abstract fields, which are discussed in more detail in Section 3.

Our community of world-wide researchers is relatively small, between 150–300 active full-time researchers. I am sure, however, that we will continue to contribute in a significant way to science, and I hope that one or the other of the following nine problems will attract attention.

1. Symbolic/Numeric Computation

A surface that is defined implicitly by all real roots (x,y,z) of a trivariate polynomial is displayed in Figure 1. The picture indicates that there are two components, an ellipsoid and a hyperboloid, which the factorization of the polynomial over the complex numbers $\mathbb{C}$ verifies.

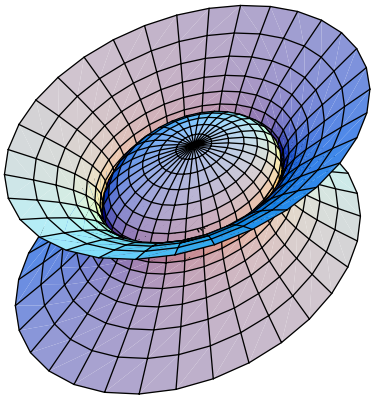
Now we take the two factors, approximate $\sqrt{2}$ numerically by 1.41422 in one factor and 1.41421 in the other, and multiply the product out rounded to three decimal places. We get the numerical trivariate polynomial equation

$$81x^4+16y^4-648.001z^4+72x^2y^2+.002x^2z^2+.001y^2z^2 \\ -648x^2-288y^2-.007z^2+1296=0.$$

Due to continuity the numerical perturbations do not change the picture of Figure 1 by much. The two now deformed components are still present. However, the polynomial has become absolutely irreducible over $\mathbb{C}$. My first open problem concerns the factorization of nearby polynomials over the complex numbers.

Open Problem 1
Given is a polynomial $f(x,y)\in\mathbb{Q}[x,y]$ and $\varepsilon\in\mathbb{Q}$. Decide in polynomial time in the degree and coefficient size if there is a factorizable $\hat{f}(x,y)\in\mathbb{C}[x,y]$ with $\|f-\hat{f}\|\le\varepsilon$, for a reasonable coefficient vector norm $\|\cdot\|$.

†Oral communication by R. S. Sutor.



$$81x^4+16y^4-648z^4+72x^2y^2-648x^2-288y^2+1296=$$
$$(9x^2+4y^2+18\sqrt{2}z^2-36)(9x^2+4y^2-18\sqrt{2}z^2-36)=0$$

Figure 1: Surface represented by trivariate polynomial

This problem was first posed in my survey article (Kaltofen, 1992). Efficient algorithms for performing the factorization of a multivariate polynomial over the complex numbers exactly are described and cited in (Kaltofen, 1995). Galligo and Watt (1997) present heuristics for computing complex numerical factors. Since then, I have learned of several related problems and their solution. They are described next. The constrained root problem described below solves Problem 1 if one looks for the nearest polynomial with a complex factor of degree no more than a given constant bound (Hitz et al., 1999).

Sensitivity analysis: approximate consistent linear system

Suppose the linear system $Ax=b$, where A is an $m\times n$ matrix over a field and b is a vector in an inner product space, is unsolvable. A classical problem is to find $\hat{b}$ "nearest to" b that makes it solvable.

If nearness is measured in terms of the norm induced by the inner product, say if one wishes to minimize the Euclidean distance, $\min_{\hat{x}}||A\hat{x}-b||_2$, a solution is obtained by the method of least squares. Another important case is when the *component-wise* distance is minimized:

$$\min_{\hat{x}}\left(\max_{1\le i\le m}\left|b_i-\sum_{j=1}^{n}a_{i,j}\hat{x}_j\right|\right)$$

By introducing a new variable y we can derive the minimum by solving the linear

program due to Chebyshev.

$$\begin{array}{ll} \text{minimize: } y & \\ \text{linear constraints:} & y \geq b_i - \sum_{j=1}^{n} a_{i,j}\hat{x}_j \quad (1 \leq i \leq m) \\ & y \geq -b_i + \sum_{j=1}^{n} a_{i,j}\hat{x}_j \quad (1 \leq i \leq m) \end{array}$$

Component-wise minimization can account for round-off errors in the entries of b.

Sensitivity analysis: nearest singular matrix

There is no particular reason why one should not consider changes in A for finding solvable systems that are nearby. In fact, there exists a theory of so-called *total* least square methods (Golub and Van Loan, 1996). Related to it is the problem of finding the numeric rank of a non-singular matrix. Both problems are numerically attacked by computing the singular value decomposition of the matrix. Unfortunately, the results are not always satisfactory, and the following may explain why this is:

Consider the following mathematical question. Given are $2n^2$ rational numbers $\underline{a}_{i,j}, \overline{a}_{i,j}$. Let A be the *interval* matrix

$$A = \{ \begin{bmatrix} a_{1,1} & \dots & a_{n,n} \\ \vdots & & \vdots \\ a_{n,1} & \dots & a_{n,n} \end{bmatrix} \mid \underline{a}_{i,j} \leq a_{i,j} \leq \overline{a}_{i,j} \text{ for all } 1 \leq i,j \leq n \}.$$

Does A contain a singular matrix?

This problem is shown to be *NP-complete* (Poljak and Rohn, 1993), i.e., it is computationally as difficult as computing the shortest traveling salesperson route in a complete graph. We mention Poljak's and Rohn's breakthrough reduction because it establishes that floating point roundoff errors may not always be easy to undo. When the distance is measured by a *matrix norm*, the problem of finding the nearest singular matrix can be solved efficiently by a result of Eckart & Young (1936) for Euclidean norms and of Gastinel (see (Kahan, 1966)) for arbitrary matrix norms.

Sensitivity analysis: approximate greatest common divisor

Suppose $f = x^m + a_{m-1}x^{m-1} + \cdots + a_0$ and $g = x^n + b_{n-1}x^{n-1} + \cdots + b_0$ have no common divisor. A problem in the same spirit as above is to efficiently compute $\hat{f}, \hat{g}$ "nearest to" f, g that have a common root.

Karmarkar and Lakshman (1996) describe an algorithm that solves this problem in polynomial time when the Euclidean distance between the combined coefficient vectors,

$$\sqrt{|a_{m-1} - \hat{a}_{m-1}|^2 + \cdots + |a_0 - \hat{a}_0|^2 + |b_{n-1} - \hat{b}_{n-1}|^2 + \cdots + |b_0 - \hat{b}_0|^2}$$

is minimized. The nearest GCD problem can be formulated in matrix form: Compute the nearest singular *Sylvester matrix* to the Sylvester matrix

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

The Sylvester matrix is a block *Toeplitz matrix* of a special form. We note that the more general problem, namely, efficiently computing the nearest singular Toeplitz matrix to a given Toeplitz matrix, remains open.

There is a substantial body of work on variations on the problem formulation, which is cited in passing (Schönhage, 1985, Corless et al., 1995, Emiris et al., 1997).

Sensitivity analysis: Kharitonov Theorem

An amazing result was obtained by V. L. Kharitonov in 1979 (see (Minnichelli et al., 1989)), which concerns the stability of roots of polynomials when the coefficients are individually perturbed. We state his theorem. Given are $2n$ rational numbers $\underline{a}_i, \overline{a}_i$. Let P be the *interval* polynomial

$$P = \{x^n + a_{n-1}x^{n-1} + \cdots + a_0 \mid \underline{a}_i \le a_i \le \overline{a}_i \text{ for all } 0 \le i < n\}.$$

Then every polynomial in P is *Hurwitz* (all roots have negative real parts), if and only if the four "corner" polynomials

$$g_k(x) + h_l(x) \in P, \quad \text{where } k = 1,2 \text{ and } l = 1,2,$$

with

$$\begin{aligned}
g_1(x) &= \underline{a}_0 + \overline{a}_2 x^2 + \underline{a}_4 x^4 + \cdots, \quad h_1(x) = \underline{a}_1 x + \overline{a}_3 x^3 + \underline{a}_5 x^5 + \cdots, \\
g_2(x) &= \overline{a}_0 + \underline{a}_2 x^2 + \overline{a}_4 x^4 + \cdots, \quad h_2(x) = \overline{a}_1 x + \underline{a}_3 x^3 + \overline{a}_5 x^5 + \cdots
\end{aligned}$$

are Hurwitz.

The corner polynomials are easily tested for the Hurwitz condition, for example by a variant of Sturm sequences, and the condition constitutes the stability criterion for the corresponding differential equations (see (Gantmacher, 1960, Ch. XV)). There now exist many generalizations of Kharitonov's theorem.

Sensitivity analysis: constrained root problem

Kharitonov's theorem inspires new problem formulations in root stability. Given is a real or complex polynomial

$$f(z) = a_n z^n + a_{n-1} z^{n-1} + \cdots + a_1 z + a_0$$

and a root $\alpha \in \mathbb{C}$ that may be given explicitly or that may be constrained to a certain subset of $\mathbb{C}$. Compute $\hat{f}$ "nearest to" f such that $\hat{f}(\alpha) = 0$.

We (Hitz and Kaltofen, 1998) can solve this problem efficiently, i.e., in polynomial time for the usual coefficient fields, and for

1. a parametric α (root stability) and Euclidean distance
2. explicit roots $\alpha_1, \alpha_2, \ldots$ and coefficient-wise distance (infinity norm)
3. with linear coefficient constraints, e.g., $a_n = 1$.

A theorem provable by our methods is the following (Hitz et al., 1999). Given is the real polynomial

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0, \quad a_i \in \mathbb{R},$$

with no real root. Then one can compute in polynomial time in n and the size of the coefficients a_i the following quantity (Hitz et al., 1999).

$$\min_{\substack{\hat{a}_0,\ldots,\hat{a}_n \text{ such that} \\ \exists \alpha \in \mathbb{R}\colon\ \hat{a}_n\alpha^n + \hat{a}_{n-1}\alpha^{n-1} + \cdots + \hat{a}_0 = 0}} \left(\max_{0 \le i \le n} |a_i - \hat{a}_i| \right) = \left| \frac{f(\alpha)}{\sum_{i=0}^{n} |\alpha^i|} \right|$$

A related result appears in (Zhi and Wu, 1998).

2. Quantifier Elimination (QE)

What makes the approximation theoretic minimax problems of Section 1 decidable in the first place? The famous process of quantifier elimination in the theory of real closed fields, first published by Tarski in 1948, delineates a substantial mathematical theory where all theorems are decidable and that has no Goedel-like undecidabilities. We illustrate the principle of QE on the simple example of minimizing a quadratic function.

$$\text{for } a > 0\colon \min_x (ax^2 + bx + c) \iff \forall y\colon\ a > 0 \text{ and } ay^2 + by + c \ge ax^2 + bx + c$$
$$\iff a > 0 \text{ and } x = -\frac{b}{2a}$$

The minimizing problem can be rewritten as a quantified expression. The principle of QE allows the removal of the quantified variables—our equivalent third form. All values of the variables that satisfy any quantifier-free expression form a so-called *semi-algebraic* set. By QE those specializations of the free variables of a quantified

expression that yield theorems, i.e., true formulas, form a semi-algebraic set. The algorithmic aspects of QE have been studied extensively by G. Collins, D. Grigoriev, J. Renegar, H. Hong, and others (see (Caviness and Johnson, 1998)). In general, the process is computationally hard, although several of the examples in Section 1 have efficient algorithms. Collins has suggested another minimax problem as a challenge benchmark problem for QE software with the intent to demonstrate the abilities of the different algorithms and their implementations.

Open Problem 2 (Zolotarev's problem by Collins 1992)
Eliminate the quantifiers and solve for $n \geq 6$ on a computer:

$$\textit{for } r > 0\textit{:} \quad \min_{B=b_0+\cdots+b_{n-2}x^{n-2}} \left(\max_{-1\leq x\leq 1} |x^n + rx^{n-1} - B(x)| \right)$$

The best approximation of a polynomial of degree n by a polynomial of lower degree on the interval $\{x \mid -1 \leq x \leq 1\}$ is accomplished via Chebyshev polynomials. Zolotarev seeks best approximations by polynomials whose degree is at least 2 less. Note that mathematical expressions for explicit solutions are known (see (Achiezer, 1956, Addendum E)) and we give the one for $r \leq 1$ and $n = 3$, again in the form of a formula in Tarski's theory.

$$\forall c_0, c_1 \forall x \exists y\colon\ 0 < r \leq 1 \text{ and } -1 \leq x \leq 1 \text{ and } -1 \leq y \leq 1 \text{ and}$$
$$(y^3 + ry^2 - c_1 y - c_0)^2 \geq \Big(x^3 + rx^2 - \underbrace{(\frac{3}{4} + \frac{r}{2} - \frac{r^2}{4})}_{b_1} x - \underbrace{(\frac{r}{4} + \frac{r^2}{6} - \frac{r^3}{108})}_{b_0} \Big)^2$$

3. Linear Algebra With Implicitly Represented Matrices

Several numerical methods for solving systems of linear equations with sparse coefficient matrices are based on the use of not too many matrix-times-vector products, which in the sparse case can be performed quickly. Examples for such methods are the conjugate gradient, the Lanczos and the Krylov algorithms. These methods are sometimes referred to as matrix-free, because no coefficient matrix needs to be explicitly constructed. In 1990 we introduced the notions of black box polynomials, rational functions, and matrices (Kaltofen and Trager, 1990). The black box model of a matrix requires as the representation of a matrix a function that performs the matrix-times-vector product.

$\mathbb{K}$ an arbitrary, e.g., finite field

In the symbolic computation context we consider abstract coefficient fields, such as finite fields. The main objective is to perform all linear algebra operations, e.g., $A^{-1}b$ (Wiedemann, 1986) with

$$\begin{array}{rl} O(n) & \text{black box calls and} \\ n^2(\log n)^{O(1)} & \text{arithmetic operations in } \mathbb{K} \text{ and} \\ O(n) & \text{intermediate storage for field elements} \end{array} \tag{1}$$

We note that a black box matrix algorithm applies not only to sparse matrices but also to structured matrices that have a fast matrix-times-vector function. Fast matrix-times-vector products are often a consequence of the linearization of a given problem. Examples are:

1. Petr's Q matrix in Berlekamp's algorithm for factoring polynomials over finite fields (Knuth, 1997), (Kaltofen and Shoup, 1998).
2. resultant matrices for non-linear algebraic equations (see Section 5 below).
3. the linear systems in Kaltofen's (1995) algorithm for factoring multivariate polynomials over algebraically closed fields.

Linear algebra algorithms for black box matrices are an active subject of research. In Table 1 we list several of them. Giesbrecht's (1997) method for finding integral solutions to sparse linear systems is based on computing rational solutions whose denominators are relatively prime. For instance, if one obtains two rational solutions with common denominator 2 and 3, respectively, one can easily construct an integer solution.

$$\begin{aligned} &A(\frac{1}{2}x^{[1]}) = b, x^{[1]} \in \mathbb{Z}^n, \quad A(\frac{1}{3}x^{[2]}) = b, x^{[2]} \in \mathbb{Z}^n; \\ &\gcd(2,3) = 1 = 2\cdot 2 - 1\cdot 3, \quad A(2x^{[1]} - x^{[2]}) = 4b - 3b = b. \end{aligned}$$

Giesbrecht proceeds by proving that for a small algebraic extension of $\mathbb{Z}$ relative primeness occurs with high probability. Recently, the need for small algebraic extension has been removed (Mulders and Storjohann, 1999).

Lambert (1996), Teitelbaum (1998), Eberly & Kaltofen (1997)	relationship of Wiedemann and Lanczos approach
Villard (1997a), (1997b)	analysis of *block* Wiedemann algorithm
Giesbrecht (1997), Mulders & Storjohann (1999)	computation of integral solutions
Giesbrecht, Lobo & Saunders (1998)	certificates for inconsistency

Table 1: Flurry of recent results

Since Wiedemann's (1986) breakthrough paper, the following problem remains unresolved.

Open Problem 3
Within the resource limitations (1) stated above,[‡] *compute the characteristic polynomial of a black box matrix over an abstract field. Randomization is allowed (of course!), as is a "Monte Carlo" solution.*

Characteristic polynomials are needed, for example, for resultant computations (Canny et al., 1989, Canny, 1990, Emiris and Pan, 1997). We conclude the section by remarks on Monte Carlo vs. Las Vegas randomized algorithms.

Classes of randomized algorithms

The use of random bits, so-called coin flips, has turned out to be a powerful algorithm design tool. Coin flips are employed for both speed, i.e., for locating the solution by a random walk rather than by deterministic search, and for output correctness, where unverifiable guesses about the solution are made. The latter comes from numerical integration: the frequency, with which a random point in space is below a function, is used to approximate the integral of the function. The following notions have become popular:

Monte Carlo	$\equiv$	always fast, probably correct
Las Vegas	$\equiv$	always correct, probably fast
BPP	$\equiv$	probably correct, probably fast

Here BPP stands for bounded probabilistic polynomial-time and is a notion from complexity theory (Boppana and Hirschfeld, 1989). The complexity class R contains all problems solvable in Las Vegas polynomial time. The term Las Vegas was coined by L. Babai (1979). Clearly, as Gene Cooperman has pointed out to me, a BPP method can be converted to a Monte Carlo method by returning garbage when the random walk consumes too much time. It is unknown if BPP = R, or if R = P, the class of problems solvable deterministically in polynomial-time. Many theorists conjecture that since randomized algorithms perform very well when implemented with pseudo-random number generators (Knuth, 1997, Chapter 3), coin flips are inessential for polynomial-time algorithmic solutions of problems. The practicality of such de-randomization is even further remote.

Some of my colleagues have expressed to me that Las Vegas randomization is fine, but Monte Carlo or BPP algorithms are suspicious as one cannot verify the guessed solution. I believe that such arguments are questionable. A Monte Carlo algorithm can deliver an answer that is correct with a probability that is as high as the user demands, e.g., higher than $1-1/2^{100}$. Of course, the probability of correctness can only be guaranteed if one uses a truly random process for the coin flips, say a quantum-physical effect. However, deterministic algorithms can also fail due to momentary hardware faults and programming bugs. Would the reader really trust a long number-theoretic proof together with a 10,000 line program more than the answer given by a single-page Monte Carlo primality testing program (Knuth, 1997, Section 4.5.4)?

[‡]Joachim von zur Gathen has suggested to relax the $O(n)$ requirements for both the number of black box calls and for intermediate auxiliary storage to $n(\log n)^{O(1)}$.

We have observed a further pitfall when trying to use Las Vegas algorithms. The following scheme is often employed, which makes a Monte Carlo algorithm Las Vegas by verifying its answer by alternative considerations.

repeat
    pick random numbers
    compute candidate answer
until a solution passes a test for it

The scheme has the flaw that a programming bug leads to an infinite loop, which is indistinguishable from bad luck in the coin flips! I have been told by many of my colleagues that such looping has also happened to them. Eventually, we give up in the belief that the coin-flips keep on being unlucky and begin searching for the bug. In one case, the problem turned out to be an invalid input.

A refree points out that even sequential schemes with unbounded iteration have this flaw. For instance, in Brown's (1971) modular GCD algorithm the Chinese remaindering is stopped if the division check by the GCD candidate succeeds. Because of our experience (Kaltofen and Monagan, 1999) we now advocate to avoid such unbounded loops whenever possible, even when adding a small extra cost.[§]

4. Lattice Basis Reduction

The lattice basis reduction algorithm by A. K. Lenstra, H. W. Lenstra, Jr. and L. Lovász (1982) is said to be the major algorithmic breakthrough of symbolic computation in the 1980s (Odlyzko, 1996). The first application of the method was to factoring polynomials over the rational numbers. Since 1982 the algorithm had its impact on a variety of problems (see, e.g., (Borwein and Lisoněk, 1997)). Recently, the method was used to derive the following pretty formula for π.

$$\pi = \sum_{i=0}^{\infty} \frac{1}{16^i} \left(\frac{4}{8i+1} - \frac{2}{8i+4} - \frac{1}{8i+5} - \frac{1}{8i+6} \right) \tag{2}$$

Following D. Bailey, P. Borwein, and S. Plouffe (1997), one proceeds as follows. Each term under the sum (2) can be expressed as an integral multiple of a rational integral.

$$\int_0^1 \frac{y^{k-1}}{1-\frac{y^8}{16}} dy = \int_0^1 \sum_{i=0}^{\infty} y^{k-1} \left(\frac{y^8}{16} \right)^i dy = \sum_{i=0}^{\infty} \frac{1}{16^i} \int_0^1 y^{8i+k-1} dy = \sum_{i=0}^{\infty} \frac{1}{16^i (8i+k)}$$

The question is which integer linear combination of these integrals yields π. Lattice basis reduction is used to find the multipliers. First, we approximate π and the integrals, multiplied by 10^{25}, via a Maple V procedure. Note that the global variable `Digits` in Maple holds the number of decimal mantissa digits with which Maple performs its floating point arithmetic.

[§]A similar programming philosophy has also been expressed by Kurt Mehlhorn for building the LEDA library.

```
> latt := proc(digits)
> local k, j, v, saved_Digits, ltt;
> saved_Digits := Digits; Digits := digits;
> for k from 1 to 8 do
>   v[k] := [];
>   for j from 1 to 10 do v[k] := [op(v[k]), 0]; od;
>   v[k][k] := 1;
>   v[k][10] := trunc(10^digits *
>                   evalf(Int(y^(k-1)/(1-y^8/16),
>                         y=0..1, digits), digits));
> od;
> v[9] := [0,0,0,0,0,0,0,0,1,
>            trunc(evalf(Pi*10^digits,digits+1))];
> ltt := [];
> for k from 1 to 9 do ltt:=[op(ltt),evalm(v[k])];od;
> Digits := saved_Digits;
> RETURN(ltt);
> end:
> L := latt(25);
```

$$
\begin{aligned}
L := [&[1, 0, 0, 0, 0, 0, 0, 0, 0, 100718447641467622864476 00],\\
&[0, 1, 0, 0, 0, 0, 0, 0, 0, 506476876667430480955939 4],\\
&[0, 0, 1, 0, 0, 0, 0, 0, 0, 339230245245199072515585 3],\\
&[0, 0, 0, 1, 0, 0, 0, 0, 0, 255412811882995341602757 0],\\
&[0, 0, 0, 0, 1, 0, 0, 0, 0, 205002557636423533944150 3],\\
&[0, 0, 0, 0, 0, 1, 0, 0, 0, 171317070666497458966732 8],\\
&[0, 0, 0, 0, 0, 0, 1, 0, 0, 147201934672635027195598 1],\\
&[0, 0, 0, 0, 0, 0, 0, 1, 0, 129077042275142343345847 8],\\
&[0, 0, 0, 0, 0, 0, 0, 0, 1, 314159265358979323846264 34]]
\end{aligned}
$$

If π is an integer linear combination of the integrals, the above lattice vectors must sum to a short vector, which can be determined by lattice basis reduction (Håstad et al., 1989) or the PSLQ algorithm (Ferguson and Bailey, 1996).

```
> readlib(lattice):
> lattice(L);
```

$$
\begin{aligned}
[&[-4,0,0,2,1,1,0,0,1,5],[0,-8,-4,-4,0,0,1,0,2,5],\\
&[-61,582,697,-1253,453,-1003,-347,-396,10,559],\\
&[-333,966,324,-1656,-56,784,1131,-351,-27,255],\\
&[429,714,-1591,778,-517,-1215,598,362,-87,398],\\
&[-1046,-259,-295,-260,1286,393,851,800,252,-1120],\\
&[494,906,-380,-1389,1120,1845,-1454,-926,-218,400],\\
&[1001,-1099,422,1766,1405,-376,905,-1277,-394,-30],\\
&[-1144,491,-637,-736,-1261,-680,-1062,-1257,637,-360]]
\end{aligned}
$$

The first short vector corresponds to (2). The second vector is another linearly in-

dependent solution:

$$2\pi = \sum_{i=0}^{\infty} \frac{1}{16^i} \left(\frac{8}{8i+2} + \frac{4}{8i+3} + \frac{4}{8i+4} - \frac{1}{8i+7} \right) \tag{3}$$

One may use Maple V.4 directly to complete the proof of (3) by carrying out the corresponding integral symbolically.

```
> g := (8*y + 4*y^2 + 4*y^3 - y^6)/(1-y^8/16);
```

$$g := \frac{8y + 4y^2 + 4y^3 - y^6}{1 - \frac{1}{16}y^8}$$

```
> int(g, y=0..1);
```

$$2\pi$$

Formulas like (2) and (3) can be used to compute the binary digits of π at very high position without keeping track of the intermediate expansion, the so-called spigot algorithm. At this moment, no formula that allows the computation of decimal digits in this space and time efficient manner is known.

The GGH public key cryptosystem

Recently, the properties of reduced lattice bases have been employed to design public key cryptosystems. Here is a nutshell description of the one by Goldreich *et al*. (1997b).

Public key A lattice basis B (rows B_i are basis vectors).

Private key A *reduced* basis C for lattice spanned by the rows of B.

Clear text Represent the message as a vector x with *small* integer entries.

Encoded message $y = x + \sum_i r_i B_i$ where $\sum_i r_i B_i$ is a random vector in the lattice.

Decryption is based on one of Babai's (1986) algorithms for nearest lattice point: Write $y = \sum_i s_i C_i$ with $s_i \in \mathbb{Q}$. Then $\sum_i \text{nearest-integer}(s_i) C_i$ is a near lattice point, probably $\sum_i r_i B_i$.

In the paper by Goldreich *et al*. (1997b) more details are given on how to choose B and C and how to sample random lattice points so that the decryption method does not produce an incorrect near lattice point. Nonetheless, Phong Nguyen has been able to break this scheme in about 3 days on a 140MHz Ultrasparc using NTL 1.7's Schnorr-Euchner variant of the reduction algorithm for lattices of dimension 200. An alternative cryptosystem is described in (Ajtai and Dwork, 1997, Goldreich et al., 1997a) and proven secure provided that computing a short non-zero lattice vector is hard.

Open Problem 4

Devise a public key cryptosystem that is based on diophantine linear algebra but that is safe from lattice basis reduction.

5. Gröbner Bases

The classical tool for solving a system of non-linear algebraic equations is the u-resultant. Consider the following simple example, due to D. Lazard (1981).

$$
\begin{aligned}
f_1 &= x^2 + xy + 2x \quad + \; y - 1 = 0 \qquad (x,y) = (1,-1),(-3,1),(0,1)\\
f_2 &= x^2 \quad + 3x - y^2 + 2y - 1 = 0\\
f_3 &= \quad ux \quad + vy + w
\end{aligned}
$$

By the theory of Macaulay (1916) the u-resultant can be expressed as the determinant of a matrix whose rows represent the polynomials multiplied by certain terms and whose columns are labeled by certain terms whose coefficients are the entries.

$$
\begin{array}{r|cccccccccc|l}
 & x^3 & x^2y & x^2 & xy^2 & xy & x & y^3 & y^2 & y & 1 & \\
xf_1 & 1 & 1 & 2 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & \\
yf_1 & 0 & 1 & 0 & 1 & 2 & 0 & 0 & 1 & -1 & 0 & \\
f_1 & 0 & 0 & 1 & 0 & 1 & 2 & 0 & 0 & 1 & -1 & \\
xf_2 & 1 & 0 & 3 & -1 & 2 & -1 & 0 & 0 & 0 & 0 & \\
yf_2 & 0 & 1 & 0 & 0 & 3 & 0 & -1 & 2 & -1 & 0 & = \\
f_2 & 0 & 0 & 1 & 0 & 0 & 3 & 0 & -1 & 2 & -1 & \\
xyf_3 & 0 & u & 0 & v & w & 0 & 0 & 0 & 0 & 0 & \\
xf_3 & 0 & 0 & u & 0 & v & w & 0 & 0 & 0 & 0 & \\
yf_3 & 0 & 0 & 0 & 0 & u & 0 & 0 & v & w & 0 & \\
f_3 & 0 & 0 & 0 & 0 & 0 & u & 0 & 0 & v & w &
\end{array}
\begin{array}{l}
(u-v+w)\\
\cdot(-3u+v+w)\\
\cdot(v+w)\\
\cdot(u-v)\\
\\
\text{(u-resultant)}
\end{array}
$$

The u-resultant has linear factors provided the system has a finite solution. Its coefficients are the coordinates of the zeros of the system, including solutions at infinity.

Buchberger's algorithm

The concept of a Gröbner basis (Buchberger, 1965, 1970, 1985, Becker and Weispfenning, 1993, Cox et al., 1996) has revolutionized commutative algebra. Buchberger's algorithm provides an alternative for solving non-linear algebraic systems. Within the algorithm, some S-polynomial constructions and reductions can be interpreted as row-reduction in Macaulay's matrices, like the one given for the u-resultant above. Faugère (1998) has been able to make this correspondence more precise. In particular, his method uses sparse so-called symbolic LU matrix decomposition for efficiently performing these row reductions. His implementation may be the first major marriage of symbolic and numeric methods. The symbolic sparse LU factorization is purely combinatorial and treats the coefficient arithmetic abstractly. In Faugère's implementation the rational coefficients still become exceedingly large, and some computations can only be done modulo a prime number. Many other numeric sparse solvers benefit from iterative approximation of the solution, and my next problem suggests to do the same for Gröbner basis computation.

Open Problem 5
Compute Gröbner bases approximately by iterative methods for solving systems, such as Gauss-Seidel, conjugate gradient, Newton,...
A solution plugs into numerical software and computes some bases faster than by exact arithmetic; the structure of the bases may be determined, e.g., by modular arithmetic.

We note that related aspects of numerical error analysis are discussed in (Shirayanagi, 1996, Stetter, 1996).

6. Transposed Matrix Products

The following phenomenon has been observed in a variety of settings. One has an efficient algorithm for computing a linear map, but one actually needs the transposed map. We shall begin with an example from field theory.

We first describe the overall approach. Let $\sigma \in \mathbb{K}(\alpha,\beta)$ where

$$g = y^2 - 2 \in \mathbb{K}[y], \qquad f = x^2 - y - 1 \in \mathbb{K}[x,y],$$
$$\beta = (y \bmod g) \in \mathbb{K}[y]/(g), \quad \alpha = (x \bmod (f,g)) \in \mathbb{K}[x,y]/(f,g),$$

with $\mathbb{K}[y]/(g) \subset \mathbb{K}[x,y]/(f,g)$. Note that $f(\alpha,\beta) = 0$ and $g(\beta) = 0$. For example, $\sigma = \sqrt{1+\sqrt{2}} - \sqrt{2} = \alpha - \beta$. In field theoretic terms, σ is an algebraic element in a tower of fields $\mathbb{K} \subset \mathbb{K}(\beta) \subset \mathbb{K}(\alpha,\beta)$. The tower is represented by a triangular set of minimum polynomials for the extension elements α and β: $g \in \mathbb{K}[y]$ irreducible over $\mathbb{K}$, $f \in \mathbb{K}(\beta)[x]$ irreducible over $\mathbb{K}(\beta)$.

The computational task is to compute the minimum polynomial $h(\sigma) = 0$:

$$h(x) = x^m - c_{m-1}x^{m-1} - \cdots - c_0 \in \mathbb{K}[x], \quad m \le \deg(f)\cdot\deg(g)$$

The coefficient vectors $\overrightarrow{\sigma^i}$ of $\sigma^i \bmod (f(x,y), g(y))$ satisfy the linear recurrence on vectors

$$\forall\, j \ge 0\colon\ \overrightarrow{\sigma^{m+j}} = c_{m-1}\overrightarrow{\sigma^{m-1+j}} + \cdots + c_0\overrightarrow{\sigma^j}$$

Any non-trivial linear projection map $L(\overrightarrow{\sigma^i})$ preserves the linear recursion because h is irreducible. An algorithm can proceed by computing the field elements $a_i = L(\overrightarrow{\sigma^i})$ for $0 \le i \le 2m-1$, i.e., using a linear map into $\mathbb{K}$, and from them the linear recurrence h, the latter in $m^{1+o(1)}$ field operations in $\mathbb{K}$ by the Berlekamp/Massey algorithm (Massey, 1969, Brent et al., 1980). We now inspect the task of computing the linear projections of the powers of σ in more detail.

Power Projections = Transposed Modular Polynomial Composition

The operator L is represented by a vector of field elements $\begin{bmatrix} u_0 & u_1 & \cdots & u_{n-1} \end{bmatrix}$, where $n = \deg(f)\deg(g)$. The power projections can be expressed by the following

vector times matrix product:

$$\left[L(\overrightarrow{\sigma^0}) \quad L(\overrightarrow{\sigma^1}) \quad L(\overrightarrow{\sigma^2}) \quad \ldots\right] = \left[u_0 \quad u_1 \quad \ldots \quad u_{n-1}\right] \cdot \underbrace{\left[\overrightarrow{\sigma^0} \;\middle|\; \overrightarrow{\sigma^1} \;\middle|\; \overrightarrow{\sigma^2} \;\middle|\; \ldots\right]}_{A}$$

The key observation is that the transposed linear map is modular polynomial composition:

$$w(z) = w_0 + w_1 z + w_2 z^2 + \cdots \longmapsto w(\sigma) \bmod (f(x,y), g(y))$$

$$\overrightarrow{w(\sigma)} = \underbrace{\left[\overrightarrow{\sigma^0} \;\middle|\; \overrightarrow{\sigma^1} \;\middle|\; \overrightarrow{\sigma^2} \;\middle|\; \ldots\right]}_{A} \cdot \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \end{bmatrix}$$

We note that modular polynomial composition in the univariate case has been studied extensively, as its complexity is closely related to factoring univariate polynomials over finite fields efficiently (von zur Gathen and Shoup, 1992, Kaltofen and Shoup, 1997, 1998, Bernstein, 1998, von zur Gathen and Gerhard, 1999).

The use of power projections for finding minimum polynomials of algebraic numbers and its relation to modular polynomial composition is discussed in (Shoup, 1994, Section 3). The special cases

$$\sigma = \alpha \pm \beta, \sigma = \alpha \cdot \beta, \sigma = \alpha/\beta \text{ with } f, g \in \mathbb{K}[x],$$

i.e., arithmetic operations on roots of polynomials, can be handled alternatively by resultant computations (Loos, 1982). These require the factorization of a polynomial of degree $\deg(f) \cdot \deg(g)$ over $\mathbb{K}$, while our approach requires the factorization of f over $\mathbb{K}(\beta)$ or of g over $\mathbb{K}(\alpha)$, which is performed similarly by either a factorization of a resultant over $\mathbb{K}$ (Trager, 1976, Encarnación, 1997), or by completely different and in some cases more efficient algorithms (Weinberger and Rothschild, 1976, Lenstra, 1987).

Nonetheless, we still have to compute the power projections. An algorithmic theorem, which we call *transposition principle*, now states that any linear algorithm that computes a matrix times vector product can be transformed into one that computes the transposed matrix times vector product. One simply reverses the flow of the linear circuit that represents the algorithm.[¶] Such reversal of flow preserves the number of arithmetic operations, but not the space needed to store intermediate results. Therefore, a fast modular polynomial composition algorithm yields one for power projections, at least in theory. Shoup (1995) proceeds differently and designs, for the case of a single algebraic extension, a baby step/giant step algorithm based on modular polynomial multiplication. Again, he needs the transposed map for which he synthesizes the following algorithm (see (Shoup, 1995) for more details).

[¶]The history of the principle is manifold. Bürgisser *et al*. (1997) trace it to circuit analysis (see (Antoniou, 1979, §4.7)). Fiduccia derives the principle in his Ph.D. thesis (1973) (see also (Fiduccia, 1972)) and Kaminski *et al*. (1988) publish a paper on it.

Transposed Modular Polynomial Multiplication in NTL

1. $T_1 \leftarrow \mathrm{FFT}^{-1}(\mathrm{RED}_k(g))$
2. $T_2 \leftarrow T_1 \cdot S_2$
3. $v \leftarrow -\mathrm{CRT}_{0\ldots n-2}(\mathrm{FFT}(T_2))$
4. $T_2 \leftarrow \mathrm{FFT}^{-1}(\mathrm{RED}_{k+1}(x^{n-1} \cdot v))$
5. $T_2 \leftarrow T_2 \cdot S_3$
6. $T_1 \leftarrow T_1 \cdot S_4$
7. Replace T_1 by the 2^{k+1}-point residue table whose j-th column $(0 \le j < 2^{k+1})$ is 0 if j is odd, and is column number $j/2$ of T_1 if j is even.
8. $T_2 \leftarrow T_2 + T_1$
9. $u \leftarrow \mathrm{CRT}_{0\ldots n-1}(\mathrm{FFT}(T_2))$

The algorithm has no interpretation of its own.[‖]

> "we offer no other proof of correctness other than the validity of this transformation technique (and the fact that it does indeed work in practice)" (Shoup, 1995, Section 7.5)

It is, however, as time and space efficient as modular polynomial multiplication, and one is left with the question if the transposition principle has this property in general. Here is then our sixth open problem.

Open Problem 6
With inputs $A \in \mathbb{K}^{m\times n}$ and $y \in \mathbb{K}^n$ you are given an algorithm for $A \cdot y$ that uses $T(m,n)$ arithmetic field operations and $S(m,n)$ auxiliary space. Show how to construct an algorithm for $A^T \cdot z$ where $z \in \mathbb{K}^m$ that uses $O(T(m,n))$ time and simultaneously $O(S(m,n))$ space. Your construction must be applicable to practical problems.

The transposition principle is a special case of *automatic differentiation* (Kaltofen and Lakshman Yagati, 1988):

$$\text{For } f(x_1,\ldots,x_n) = b^T(A \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}) \text{ we have } \begin{bmatrix} \partial_{x_1} f \\ \vdots \\ \partial_{x_n} f \end{bmatrix} = A^T b.$$

Therefore, the so-called *reverse mode* of automatic differentiation for computing the gradient vector of a function applies. Griewank (1992) shows how to solve the above problem with a factor of $O(\log(mn))$ penalty in both time and space.

We conclude by noting that the Lanczos algorithms and certificates of inconsistency of Table 1 all depend on efficient transposed matrix times vector products.

[‖] In the meantime, Shoup (1999) has been able to derive an explicit fast algorithm for the transposed modular polynomial multiplication problem. However, his algorithm is still by a constant factor slower than the the one based on the transposition principle.

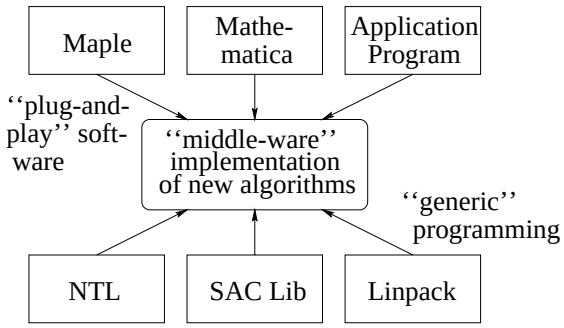


Figure 2: Component interaction in symbolic computation

7. Plug-And-Play Components

The notion of plug-and-play components comes from the interaction between computer hardware and the resident operating system. If new hardware is installed, the operating system can probe the device and determine and configure its characteristics without human help. The device must respond to the probes in an agreed fashion and the operating system must know of the possible different device characteristics. The question arises if a plug-and-play approach can be realized for purpose of arranging software components in a customized fashion.

An example for the need of the plug-and-play methodology is given in Figure 2. A new algorithm, say the solution to Problem 1, is to be made available to researchers outside the discipline of symbolic computation. We assume that the users work on commercial platforms such as Maple, Mathematica, or an Internet browser/Java engine. Clearly, it is desirable to implement the brand-new algorithm in such a way that it is callable from these common platforms. R. Loos (1974) recognizes the need as vertical integration.

A second issue concerns the usage of existing libraries, say for polynomial arithmetic or arbitrary precision floating point arithmetic. Computer algebra has pioneered what is now called *generic programming*, where the underlying implementations are hidden and multiple packages can be used, even at the same time. Generic programming was introduced simply because it became unwieldy to write a Gaussian elimination procedure, for example, for each of the different coefficient fields that arose. Now generic programming has become a means of technology transfer. When a new sparse symbolic LU factorization algorithm is released in Linpack, that algorithm should be instantaneously called from all programs that rely on sparse LU factorization.

The notion of *problem solving environments* has been coined. We offer plug-and-play and generic programming software design as a definition. With it an end-user can easily custom-make symbolic software tools.

Example: FOXBOX (Díaz and Kaltofen, 1998)

One goal of our implementation of the algorithms for factoring polynomials in so-called black box representation (Kaltofen and Trager, 1990) was a plug-and-play interface to our FOXBOX system. Our Maple server makes the procedures in FOXBOX accessible to a Maple session. These are a Maple user's commands.

```
# Call FoxBox server from Maple
> SymToeQ  := BlackBoxSymToe( BBNET_Q, 4, -1, 1.0 ):
> SymToeZP := BlackBoxSymToe( BBNET_ZP, 4, -1, 1.0 ):
> FactorsQ := BlackBoxFactors( BBNET_Q, SymToeQ, Mod, 1.0, Seed ):
> FactorsZP := BlackBoxHomomorphicMap( BBNET_FACS, FactorsQ, SymToeZP ):
```

The server program is a generic one. Here we give a code fragment of the version that uses SACLIB 1.1 rational number and polynomial arithmetic. The C++ constructor calls get compiled from the FOXBOX template library.

```
// construct factors of a symmetric Toeplitz determinant in C++
typedef BlackBoxSymToeDet< SaclibQ, SaclibQX > BBSymToeDetQ;
typedef BlackBoxFactors< SaclibQ, SaclibQX, BBSymToeDetQ > BBFactorsQ;

BBSymToeDetQ SymToeDetQ( N );
BBFactorsQ   FactorsQ( SymToeDetQ, Probab, Seed, &MPCard );
```

Software Design Issues

We think it useful to distinguish the plug-and-play interface from the generic programming interface, although the two are somewhat similar. We have identified several issues with both methodologies.

Plug-and-play

1. The software components create a standard *serialized representation* for exchanging their data. For mathematical objects several standards have been proposed: MP (Bachmann et al., 1997), OpenMath (Dalmas et al., 1997), and MathML (Ion and Miner, 1998). All standards use static representation of the objects.
2. In FOXBOX we could from the beginning transfer procedures for making or evaluating polynomials and matrices. Our procedures were written in standard programming languages, e.g., in C++, as security from illegal operations was not an issue. Following the Java paradigm, it will be useful to transfer *byte code* for constructing objects in place of the parse trees for their canonical representation (cf. (Norman and Fitch, 1996)). The matrix $\left[1/(i^2+j^2)\right]_{1\le i,j\le 100}$ surely should be communicated via a procedure for constructing it. A standard for the mathematical byte code solves the task.**

**The OpenMath "programming content dictionary" proposed by Gaston Gonnet offers a possible solution.

3. Problem solving environments (PSEs) (Lakshman Y. N. et al., 1998) are the end product of the assembly of the different software components. *Visual programming environments* for instantiation of generic and assembly of plug-and-play components can become part of the user interface to symbolic computation systems, and assist the end-user in making her/his PSE, who thus unwittingly turns into a very high level programmer.
4. A language that allows *overloaded operators* permits the wrapping of existing code with new definitions for the operators. MITMatlab (Husbands et al., 1998) is an example where parallelism is introduced in this manner without having to modify sequential library functions.

Generic Programming

1. The definition of *common object interfaces* have been studied extensively in the context of computer algebra (Musser, 1975, Abdali et al., 1986, Jenks and Sutor, 1992, Monagan, 1993, Watt et al., 1994). The Standard Template Library of C++ (Musser and Saini, 1996) is an example from main-stream programming. STL has container objects with a standardized format for manipulating them.[††] Such standards can be created for mathematical objects such as polynomials, matrices, or combinatorial graphs. Libraries then are accessed through so-called wrapper classes that convert the standard calling interface to the internal organization. No efficiency is lost, as the wrapper functions can be inlined on compilation. In FOXBOX the wrapper classes provide streams of objects. For instance, `K::random_generator(500)` is a generic function object of the field class that provides random field elements that are uniformly sampled from a set of 500 elements. Random field elements are needed for randomized algebraic algorithms (cf. Section 3). Specification of the object interfaces can be expressed in CORBA, the Common Object Request Broker Architecture (Iglio and Attardi, 1998).
2. Several libraries, e.g., SACLIB, perform *garbage collection*, which complicates the generic object interface. Explicit *storage management* remains a sticky issue even in STL, as different C++ compilers implement different memory models.
3. *Exceptions*, like division by zero or failure due to an unlucky selection of random elements, must be handled in a generic fashion across the components.
4. Object interfaces sometimes become a barrier that disallows interaction between the generic algorithm and the used generic arithmetic. For instance, a ring of polynomials whose arithmetic is implemented by FFT-based methods, such as in NTL (Shoup, 1998), has an efficient specialized evaluation/interpolation scheme. The generic algorithms can use a *shortcut* into the specialized procedures when employing a homomorphic imaging strategy. The wrapper classes can facilitate these shortcuts.

[††]STL does not supply a standard for serializing its containers for transport across systems.

5. *Parallel distribution* of a symbolic computation over many different computers is important as our computations tend to be large. Internet browsers can present a familiar platform for managing the network of computers one utilizes. The algorithms must be programmed with high-level parallelization that is built-in from the beginning.

Open Problem 7
Devise a plug-and-play and generic programming methodology for symbolic mathematical computation that is widely adopted by the experts in algorithm design, the commercial symbolic software producers, and the outsider users.

From our FOXBOX experience I observed that designing a system that simultaneously plugs into several others is difficult. H. Hong, a co-author of SACLIB, noted that the reverse is also true, namely, that designing a system that someone else can plug-in is difficult. However, our symbolic software is becoming very complex and a solution to Problem 7 is crucial.

8. Killer Applications

The commercial success of a computer product is, according to Stephen Jobs, dependent on a so-called "killer" application that everyone wants but only the one product has. In Table 2 I attempt to list the killer app's for several products, including symbolic software.

Product	"Killer application"
Macintosh	Document preparation
Personal computer	Spreadsheets
Supercomputers	Weather forecasting
Mainframe computers	Social security system
Symbolic software	Calculus teaching

Table 2: Successful applications

Certainly, the students learning calculus with the help of a computer algebra system constitute the most numerous users of our software. At North Carolina State University alone they number about 8000 per year. Such proliferation of use makes a discipline important to society, but it also influences the direction of its future development. I am told that much of the MathML Internet standard is geared towards mathematics education. My last problem, admittedly non-scientific, addresses this situation.

Open Problem 8
Besides math education, find another so-called "killer" application for symbolic computation.
The problem is solved when the new application makes the software written for it a commercial success.

Summary, Acknowledgement and Note Added in March 2000

These are my open problems.

1. Nearby multivariate polynomials that factor over $\mathbb{C}$
2. Zolotarev's problem on a computer
3. Characteristic polynomial of a black box matrix
4. Lattice basis reduction-safe GGH-like cryptosystems
5. Gröbner bases via iterative numerical methods
6. Space and time efficient transposition principle
7. Plug-and-play and generic programming methodology for symbolic computation
8. Another "killer" application besides education

I would like to thank Laurent Bernardin, Bob Caviness, George Nakos, and Peter Turner for giving me the forum to present them and Victor Shoup for his comments on Problem 6. I appreciate the suggestions for improvement made by George Collins and by P. Borwein, P. Lisoněk and M. Monagan during my visit at Simon Fraser University in July 1998. Rob Corless carefully read my manuscript and made several helpful suggestions. All three referees of the paper corrected several errors and made many valuable suggestions, for which I am grateful.

The above problems were posed in April 1998. To my knowledge, none has been resolved as of March 2000. However, there has been significant progress on several of them, which I would like to mention. Problem 1 has been tackled by several authors using numerical techniques. While the resulting algorithms do not resolve the problem in its entirety, the solutions are nonetheless useful in many cases, similarly to complex root finding procedures that do not converge for all inputs. Good progress can be reported on problem 3. Gilles Villard has found an algorithm that is within a factor of $n^{1/2+o(1)}$ of the required complexity. The algorithm does not rely on fast matrix multiplication algorithms and therefore is more practical than those methods, which are still asymptotically faster. Important research is being conducted on problem 7, but I shall only provide some references (Wang, 1999, Le and Howlett, 1999, Bernardin et al., 1999). The solution of problem 8 may lie in the past. The Nobel Prize in physics was awarded in 1999 to Gerardus 't Hooft and Martinus J. G. Veltman, and as the citation of the Nobel Foundation reads [`www.nobel.se/announcement-99/physics99.html`] "At the end of the 1960s ... Veltman had developed the Schoonschip computer program which, using symbols, performed algebraic simplifications of the complicated expressions that all quantum field theories result in when quantitative calculations are performed. ... With the help of Veltman's computer program 't Hooft's partial results were now verified and together they worked out a calculation method in detail."

A. Appendix: Complex Variables in Computer Algebra (by R. M. Corless and D. J. Jeffrey)

By now, many users and most developers of computer algebra systems (CAS) are familiar with a set of problems known generically for many years as "the square root bug." What was meant by this expression was that CAS would sometimes transform complex-valued expressions incorrectly. The example that gave the bug its name was the transformation of $\sqrt{x^2}$ to x, which is not even valid for all real x, much less for complex x. Squarely at the root of the difficulty is multivaluedness, which does happen with real variables, but is much more common with complex variables.

A concise and elegant description of the problem can be found in Stoutemyer (1991). More discussion can be found in the papers by Aslaksen, by Patton, by Fateman, by Rich & Jeffrey, and by Corless & Jeffrey in Issue 116 (June 1996) of the ACM SIGSAM BULLETIN (Communications in Computer Algebra).

One of the fundamental difficulties in dealing with 'the square root bug' is that because of multivaluedness, some cherished algebraic identities, such as

$$\ln z_1 z_2 = \ln z_1 + \ln z_2 \,, \tag{4}$$

no longer hold—they are not true for all specializations of the variables. People have been willing to try to keep these identities, at almost any cost. For example, in Carathéodory (1964), even Carathéodory was willing to *c*hange the meaning of equality in order to keep equation (4). He changed the interpretation of the symbols in the formula in two ways. First, he interpreted each side as a set. Second, he interpreted the equals sign to mean that the set represented on the left-hand side of the equals sign had a non-empty intersection with the set represented on the right-hand side. In a discussion at ECCAD '98, Dana Scott observed that Carathéodory's notion of equality is not transitive, and hence is not an equivalence relation.

Other approaches, such as Riemann surfaces, are also not satisfactory (chiefly because algebra on them seems so difficult).

One of the main consequences of these mathematical difficulties is that there have been persistent bugs in many computer algebra systems, particularly in integration, where changing interpretations of square roots or the like have resulted in absurdities such as positive integrands giving negative (definite) integrals. It is a firmly held conviction in CAS that the proper setting for the integration problem is the complex plane. One consequence is that it is quite possible to integrate f' to get $f + c$ where c is a complex *p*iecewise constant; indeed the discontinuities in c may be complex.

While we would like to pose the general problem of correct simplification of complex-valued transcendental functions as our "open problem," we feel that this is both too vague and too difficult. On the other hand, "fixing all the bugs in CAS" doesn't have the right tone, either. Instead, we focus on a smaller subproblem in this area.

Recent progress on 'the square root bug' includes the removal by all major computer algebra systems of automatic simplifications that are not always true on specialization over the complex numbers (except possibly on sets of measure zero). Progress has also been made on integration, particularly with the papers Jeffrey (1993), Jeffrey

and Rich (1994), Jeffrey (1994, 1997). The main contribution of these papers is that it is better to return an integral that is *c*ontinuous on a domain of maximum extent, rather than trying to fix up spurious singularities and branch cuts later. See also a discussion of Rioboo's algorithm, such as the one in Bronstein (1997).

However, these papers address only the simplest sorts of integrals. One of the central pillars of computer algebra is the Risch integration algorithm and its extensions (see for example Bronstein (1997)). The algorithm, which is algebraic and not analytic in its essentials, does not always produce integrals continuous on domains of maximum extent. Further, it often forces computation into the complex plane. For a simple example, consider the following Maple session.

```
> infolevel[int] := 5;
```

$$infolevel_{int} := 5$$

We force Maple to skip its inexpensive heuristics and go to the Risch algorithm, normally a last resort.

```
> ah := `int/risch_like`(1/(2+sin(z)),z);

int/risch:   enter Risch integration

int/risch/algebraic1:   RootOfs should be algebraic numbers and
functions

int/risch:   the field extensions are
```

$$[z, e^{(\mathrm{RootOf}(\_Z^2+1)\,z)}]$$

```
int/risch:   Introduce the namings:
```

$$\{\_th_1 = e^{(\mathrm{RootOf}(\_Z^2+1)\,z)}\}$$

```
unknown:   integrand is
```

$$\frac{1}{2-\frac{1}{2}\,\mathrm{RootOf}(\_Z^2+1)\,(\_th_1-\frac{1}{\_th_1})}$$

```
unknown:   integrand expressed as
```

$$2\,\frac{\mathrm{RootOf}(\_Z^2+1)\,\_th_1}{4\,\mathrm{RootOf}(\_Z^2+1)\,\_th_1+\_th_1^{\,2}-1}$$

```
int/risch/ratpart:   integrating
```

$$2\,\frac{\mathrm{RootOf}(\_Z^2+1)\,\_th_1}{4\,\mathrm{RootOf}(\_Z^2+1)\,\_th_1+\_th_1^{\,2}-1}$$

```
int/risch/ratpart:   Hermite reduction yields
```

$$\int 2\,\frac{\mathrm{RootOf}(\_Z^2+1)\,\_th_1}{4\,\mathrm{RootOf}(\_Z^2+1)\,\_th_1+\_th_1^{\,2}-1}\,dz$$

```
int/risch/ratpart:   Rothstein's method - factored resultant is
```

$$3z^2+1$$

```
int/risch/ratpart:   result is
```

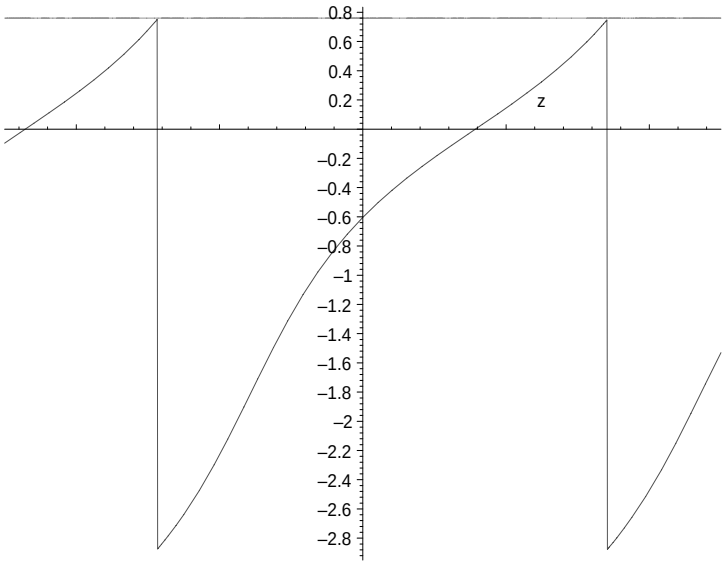
$$\frac{1}{3}I\sqrt{3}\ln(\_th_1+I\sqrt{3}+2\,\mathrm{RootOf}(\_Z^2+1))-\frac{1}{3}I\sqrt{3}\ln(\_th_1-I\sqrt{3}+2\,\mathrm{RootOf}(\_Z^2+1))$$

```
int/risch:   exit Risch integration
```

$$ah:=\frac{1}{3}I\sqrt{3}\ln(e^{(Iz)}+I\sqrt{3}+2I)-\frac{1}{3}I\sqrt{3}\ln(e^{(Iz)}-I\sqrt{3}+2I)$$

We see by the following plot that the Risch algorithm applied to this simple problem does not produce a continuous antiderivative.

```
> plot([evalc(Re(ah)),evalc(Im(ah))],z=-5..5);
```



Open Problem 9

Modify (or replace) the Risch algorithm so as to produce integrals continuous on domains of maximum extent, or else clearly describe the largest class of functions (elementary or other) for which continuous antidifferentiation can be done efficiently.

This problem will be known to be solved when a proof of the algorithm's correctness appears in a good journal, or more preferably when the algorithm is implemented in a major CAS and thus made available for general use and scrutiny.

Since, even for the integration of rational functions over a complex field, we can have branch cuts consisting of any algebraic curve whatsoever, this seems difficult. For rational functions over a real field, Rioboo's algorithm can be used (Bronstein, 1997). Some related issues include the following.

A.1. Domains of Computation

We feel that the default domain of computation should match the user's expectations (and be modifiable at the user's will). If the computer algebra system begins its computations in the real domain, assuming all variables are real and all functions are real-valued, then if at any time during the computation the system decides to move into the complex domain, as it did in our Risch example, then the user should be warned somehow; B. F. Caviness has suggested that the background colour of the screen could change, for example.

Automatic simplifications should only perform transformations valid for all specializations of the variables in the domain in question (possibly using dynamic evaluation, or lazy dynamic evaluation (a.k.a. 'provisos')). The user, of course, should be allowed to perform any manipulation she or he desires.

A.2. Integration of Special Functions

There are several classes of complex-valued special functions which are of great value to the scientist, not all of which are supported equally well by the major computer algebra systems. Examples include the Jacobian elliptic functions and elliptic integrals, and the hypergeometric or even Meijer G functions or still more generally the so-called "H-functions." The Jacobian elliptic functions are very rich in algebraic identities, occur very often in applications (see the beautiful book Lawden (1989)), and being doubly-periodic in the complex plane have multivalued inverses. Therefore, all the difficulties talked about earlier are inherited here as well.

As an example, consider

$$\int \mathrm{cn}(u,k)\,du\,. \tag{5}$$

We will use the substitution $\phi = \mathrm{am}(u)$, where $\mathrm{am}(u)$ is Jacobi's amplitude function and satisfies

$$\begin{aligned} \mathrm{sn}(u,k) &= \sin(\mathrm{am}(u,k)) = \sin(\phi) \\ \mathrm{cn}(u,k) &= \cos(\mathrm{am}(u,k)) = \cos(\phi) \\ \frac{d}{du}\mathrm{am}(u,k) &= \mathrm{dn}(u,k)\,. \end{aligned}$$

This gives that $du = d\phi/\sqrt{1-k^2\sin^2\phi}$, on using the identity

$$k^2\mathrm{sn}^2(u,k) + \mathrm{dn}^2(u,k) = 1\,.$$

Therefore, we may express any integrand rational in sn, cn, and dn as an algebraic integral in $\sin(\phi)$ and $\cos(\phi)$ by a simple change of variables. Our simple example gives

$$\int \mathrm{cn}(u,k)\,du = \int \frac{\cos(\phi)\,d\phi}{\sqrt{1-k^2\sin^2(\phi)}}\,. \tag{6}$$

It is an easy exercise ((Lawden, 1989, p. 40)) to show that this is in fact equal to $\int \mathrm{cn}(u,k)\,du = \sin^{-1}(k\mathrm{sn}(u,k))/k$, up to a constant. More interestingly, Maple gives a nontrivial discontinuous integral for the following continuous integrand (with explicit use of `changevar`, because otherwise Maple does not know how to integrate the Jacobian elliptic functions as yet):

$$\int \frac{du}{2+\mathrm{cn}(u,k)} = \int \frac{d\phi}{(2+\cos\phi)\sqrt{1-k^2\sin^2\phi}}\,. \tag{7}$$

We leave the verification of this as an exercise for the reader.

Integration of all these functions would be very useful; integration valid on domains of maximum extent would be more useful. Although the simplification of expressions containing these functions and a few transcendental constants is impossible in general (even recognizing zero is undecidable),[‡‡] one can still ask packages to do as much as is possible.

A.3. Branches and the Unwinding Number

Many people have tried to automate symbolic computations with multivalued functions; see for example Dingle and Fateman (1994) or Corless and Jeffrey (1996). It now appears, at least for the logarithm and hence for simple elementary functions, that complex analysis *c*an be turned into computer algebra. Once you replace the (false) identity $\ln\exp z = z$ with the true identity

$$\ln e^z = z + 2\pi i K(z)\,, \tag{8}$$

where K is the so-called "unwinding number," then computer algebra systems can manipulate some complex formulæ correctly. The geometric information about the branch cuts is encoded in the arguments to the unwinding number (which makes this approach similar, in fact, to that of Dingle and Fateman (1994)). There are simple theorems one can use to simplify some unwinding numbers, and other algebraic identities that can be implemented, such as

$$\begin{aligned}
K(z) &= \left\lceil \frac{\mathrm{Im}(z)-\pi}{2\pi} \right\rceil \\
\ln z_1 z_2 &= \ln z_1 + \ln z_2 + 2\pi i K(\ln z_1 + \ln z_2) \\
w\ln z &= \ln z^w + 2\pi i K(w\ln z) \\
K(z+2\pi i n) &= K(z) + n \\
K(\ln z) &= 0 \\
z_1^w z_2^w &= (z_1 z_2)^w \exp(2\pi i w K(\ln z_1 + \ln z_2)) \\
z^{vw} &= (z^v)^w \exp(2\pi i w K(v\ln z))\,.
\end{aligned}$$

[‡‡]As a point of clarification, it is useful to emphasize the distinction between the undecidability of algebraic simplification in general and the decidability of integration once the model for the field of extensions is known, as shown by Risch. See (Bronstein, 1997) for a detailed discussion.

A prototype implementation in Maple is under construction, by Gurjeet Litt (a Masters' student at the University of Western Ontario at this time of writing).

The talk from which the material of this appendix was extracted can be found at `http://www.apmaths.uwo.ca/~rmc/papers/symbolic/index.html` under the heading "East Coast Computer Algebra Day 1998."

References

S. K. Abdali, G. W. Cherry, and N. Soiffer. An object-oriented approach to algebra system design. In B. W. Char, editor, *Proc. 1986 Symp. Symbolic Algebraic Comput. Symsac '86*, pages 24–30, New York, N. Y., 1986. ACM.

N. I. Achiezer. *Theory of Approximation*. Frederick Ungar Publ. Co., New York, NY, 1956. Translated by C. J. Hyman.

M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average case equivalence. In *Proc. 29nd Annual ACM Symp. Theory Comput.*, pages 284–293, New York, N.Y., 1997. ACM Press.

A. Antoniou. *Digital Filters: Analysis and Design*. McGraw-Hill Book Co., New York, 1979.

L. Babai. Monte-Carlo algorithms in graph isomorphism testing. Rapport de recherches 79-10, Univ. de Montréal, Dép. de mathématiques et de statistique, 1979.

L. Babai. On Lovász lattice reduction and the nearest lattice point problem. *Combinatorica*, 6:1–13, 1986.

O. Bachmann, H. Schönemann, and S. Gray. A proposal for syntactic data integration for math protocols. In Hitz and Kaltofen (1997), pages 165–175.

D. Bailey, P. Borwein, and S. Plouffe. On the rapid computation of various polylogarithmic constants. *Math. Comput.*, 66(218):903–913, 1997.

T. Becker and V. Weispfenning. *Gröbner bases A Computational Approach to Commutative Algebra*. Springer Verlag, New York, N.Y., 1993.

E. R. Berlekamp. Factoring polynomials over large finite fields. *Math. Comp.*, 24: 713–735, 1970.

L. Bernardin, B. Char, and E. Kaltofen. Symbolic computation in Java: an appraisement. In Dooley (1999), pages 237–244.

Daniel J. Bernstein. Composing power series over a finite ring in essentially linear time. *J. Symbolic Comput.*, 26(3):339–341, September 1998.

R. B. Boppana and R. Hirschfeld. Pseudorandom generators and complexity classes. In S. Micali, editor, *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 1–26. JAI Press Inc., Greenwhich, Connecticut, 1989.

J. M. Borwein and P. Lisoněk. Applications of integer relation algorithms. Preprint 97:104, Centre for Experimental and Constructive Mathematics, Simon Fraser University, `http://mosaic.cecm.sfu.ca/preprints/1997pp.html`, 1997.

R. P. Brent, F. G. Gustavson, and D. Y. Y. Yun. Fast solution of Toeplitz systems of equations and computation of Padé approximants. *J. Algorithms*, 1:259–295, 1980.

Manuel Bronstein. *Symbolic Integration I (Transcendental Functions)*, volume 1 of *Algorithms and Computation in Mathematics*. Springer-Verlag, 1997.

W. S. Brown. On Euclid's algorithm and the computation of polynomial greatest common divisors. *J. ACM*, 18:478–504, 1971.

W. S. Brown and J. F. Traub. On Euclid's algorithm and the theory of subresultants. *J. ACM*, 18:505–514, 1971.

B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. Dissertation, Univ. Innsbruck, Austria, 1965.

B. Buchberger. Ein algorithmisches Kriterium für die Lösbarkeit eines algebraischen Gleichungssystems. *aequationes mathematicae*, 4(3):374–383, 1970.

B. Buchberger. Gröbner bases: An algorithmic method in polynomial ideal theory. In N. K. Bose, editor, *Recent Trends in Multidimensional Systems Theory*, pages 184–232. D. Reidel Publ. Comp., Dordrecht (Holland), 1985.

P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic Complexity Theory*. Springer Verlag, Heidelberg, Germany, 1997.

J. Canny. Generalized characteristic polynomials. *J. Symbolic Comput.*, 9(3):241–250, 1990.

J. Canny, E. Kaltofen, and Lakshman Yagati. Solving systems of non-linear polynomial equations faster. In *Proc. ACM-SIGSAM 1989 Internat. Symp. Symbolic Algebraic Comput. ISSAC '89*, pages 121–128. ACM, 1989.

Constantin Carathéodory. *Theory of Functions of a Complex Variable*, volume 1. Chelsea, New York, 1964. p. 259.

B. F. Caviness and J. R. Johnson, editors. *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Springer Verlag, Berlin/Heidelberg, 1998.

G. E. Collins. A method for overlapping and erasure of lists. *Commun. ACM*, 3(12): 655–657, December 1960.

R. M. Corless, P. M. Gianni, B. M. Trager, and S. M. Watt. The singular value decomposition for polynomial systems. In Levelt (1995), pages 96–103.

Robert M. Corless and David J. Jeffrey. The unwinding number. *SIGSAM Bulletin*, 30(1):28–35, 1996. Issue 115.

D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties and Algorithms*. Springer, New York, second edition, 1996.

S. Dalmas, M. Gaëtano, and S. Watt. An OpenMath 1.0 implementation. In Küchlin (1997), pages 241–248. See also `http://naomi.math.ca/`.

A. Díaz and E. Kaltofen. FOXBOX a system for manipulating symbolic objects in black box representation. In Gloor (1998), pages 30–37.

Adam Dingle and Richard Fateman. Branch cuts in computer algebra. In *ISSAC '94 Proc. Internat. Symp. Symbolic Algebraic Comput.*, pages 250–257, 1994.

S. Dooley, editor. *ISSAC 99 Proc. 1999 Internat. Symp. Symbolic Algebraic Comput.*, New York, N. Y., 1999. ACM Press.

W. Eberly and E. Kaltofen. On randomized Lanczos algorithms. In Küchlin (1997), pages 176–183.

C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, September 1936.

I. Z. Emiris, A. Galligo, and H. Lombardi. Certified approximate univariate GCDs. *J. Pure Applied Algebra*, 117 & 118:229–251, May 1997. Special Issue on Algorithms for Algebra.

I. Z. Emiris and V. Y. Pan. The structure of sparse resultant matrices. In Küchlin (1997), pages 189–196.

Mark J. Encarnación. Factoring polynomials over algebraic number fields via norms. In Küchlin (1997), pages 265–270.

J.-C. Faugère. Gb. Internet document, 1998. See links from `http://posso.lip6.fr/~jcf/`. A paper to be published in the forthcoming MEGA'98 Proceedings is also posted.

H. R. P. Ferguson and D. H. Bailey. Analysis of PSLQ, an integer relation finding algorithm. Technical Report NAS-96-005, NASA Ames Research Center, 1996.

C. M. Fiduccia. On obtaining upper bounds on the complexity of matrix multiplication. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 31–40. Plenum Press, New York, 1972.

C. M. Fiduccia. *On the Algebraic Complexity of Matrix Multiplication*. PhD thesis, Brown Univ., Providence, Rhode Island, Center Comput. Inform. Sci., Div. Engin., June 1973.

A. Galligo and S. Watt. A numerical absolute primality test for bivariate polynomials. In Küchlin (1997), pages 217–224.

F. R. Gantmacher. *The Theory of Matrices*, volume 2. Chelsea Publ. Co., New York, N. Y., 1960.

J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, Cambridge, New York, Melbourne, 1999.

J. von zur Gathen and V. Shoup. Computing Frobenius maps and factoring polynomials. *Comput. Complexity*, 2:187–224, 1992.

M. Giesbrecht. Efficient parallel solutions of sparse systems of linear diophantine equations. In Hitz and Kaltofen (1997), pages 1–10.

M. Giesbrecht, A. Lobo, and B. D. Saunders. Certifying inconsistency of sparse linear systems. In Gloor (1998), pages 113–119.

O. Gloor, editor. *ISSAC 98 Proc. 1998 Internat. Symp. Symbolic Algebraic Comput.*, New York, N. Y., 1998. ACM Press.

O. Goldreich, S. Goldwasser, and S. Halevi. Eliminating decryption errors in Ajtai-Dwork cryptosystems. In Kaliski (1997), pages 103–111.

O. Goldreich, S. Goldwasser, and S. Halevi. Public-key crpytosystems from lattice reduction problems. In Kaliski (1997), pages 112–131.

G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, Maryland, third edition, 1996.

A. Griewank. Achieving logarithmic growth of temporal and spatial complexity in reverse automatic differentiation. *Optimization Methods & Software*, 1:35–54, 1992.

D. Yu. Grigoriev and Lakshman Y. N. Algorithms for computing sparse shifts for multivariate polynomials. In Levelt (1995), pages 96–103.

J. Håstad, B. Just, J. C. Lagarias, and C. P. Schnorr. Polynomial time algorithms for finding integer relations among real numbers. *SIAM J. Comput.*, 18(5):859–881, 1989.

M. Hitz and E. Kaltofen, editors. *Second Internat. Symp. Parallel Symbolic Comput. PASCO '97*, New York, N. Y., 1997. ACM Press.

M. A. Hitz and E. Kaltofen. Efficient algorithms for computing the nearest polynomial with constrained roots. In Gloor (1998), pages 236–243.

M. A. Hitz, E. Kaltofen, and Lakshman Y. N. Efficient algorithms for computing the nearest polynomial with a real root and related problems. In Dooley (1999), pages 205–212.

P. Husbands, C. L. Isbell Jr., and A. Edelman. Interactive supercomputing with MIT-Matlab. See link at `http://www-math.mit.edu/~edelman/`, August 1998.

P. Iglio and G. Attardi. Software components for computer algebra. In Gloor (1998), pages 62–69.

P. Ion and R. Miner. Mathematical markup language. Internet document `http://www.w3.org/TR/WD-math/`, 1998.

David J. Jeffrey. Integration to obtain expressions valid on domains of maximum extent. In Manuel Bronstein, editor, *Proc. 1993 Internat. Symp. Symbolic Algebraic Comput. ISSAC'93*, pages 34–41, New York, N. Y., 1993. ACM Press.

David J. Jeffrey. The importance of being continuous. *Mathematics Magazine*, 67: 294–300, 1994.

David J. Jeffrey. Rectifying transformations for the integration of rational trigonometric functions. *J. Symbolic Comput.*, 24:563–573, 1997.

David J. Jeffrey and A. D. Rich. The evaluation of trigonometric integrals avoiding spurious discontinuities. *ACM Trans. Math. Software*, 20:124–135, 1994.

R. D. Jenks and R. S. Sutor. *axiom The Scientific Computing System*. Springer Verlag, New York, 1992.

W. Kahan. Numerical linear algebra. *Canadian Math. Bull.*, 9:757–801, 1966.

B. S. Kaliski, Jr., editor. *Advances in Cryptology — CRYPTO '97*, volume 1294 of *Lect. Notes Comput. Sci.*, Berlin, 1997. Springer.

E. Kaltofen. Polynomial factorization 1987-1991. In I. Simon, editor, *Proc. LATIN '92*, volume 583 of *Lect. Notes Comput. Sci.*, pages 294–313, Heidelberg, Germany, 1992. Springer Verlag.

E. Kaltofen. Effective Noether irreducibility forms and applications. *J. Comput. System Sci.*, 50(2):274–295, 1995.

E. Kaltofen and Lakshman Yagati. Improved sparse multivariate polynomial interpolation algorithms. In P. Gianni, editor, *Symbolic Algebraic Comput. Internat. Symp. ISSAC '88 Proc.*, volume 358 of *Lect. Notes Comput. Sci.*, pages 467–474, Heidelberg, Germany, 1988. Springer Verlag.

E. Kaltofen and M. Monagan. On the genericity of the modular polynomial GCD algorithm. In Dooley (1999), pages 59–66.

E. Kaltofen and V. Shoup. Fast polynomial factorization over high algebraic extensions of finite fields. In Küchlin (1997), pages 184–188.

E. Kaltofen and V. Shoup. Subquadratic-time factoring of polynomials over finite fields. *Math. Comput.*, 67(223):1179–1197, July 1998.

E. Kaltofen and B. Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *J. Symbolic Comput.*, 9(3):301–320, 1990.

M. Kaminski, D. G. Kirkpatrick, and N. H. Bshouty. Addition requirements for matrix and transposed matrix products. *J. Algorithms*, 9:354–364, 1988.

N. Karmarkar and Lakshman Y. N. Approximate polynomial greatest common divisors and nearest singular polynomials. In Lakshman Y. N. (1996), pages 35–42.

D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison Wesley, Reading, Massachusetts, USA, Third edition, 1997.

W. Küchlin, editor. *ISSAC 97 Proc. 1997 Internat. Symp. Symbolic Algebraic Comput.*, New York, N. Y., 1997. ACM Press.

Lakshman Y. N., editor. *ISSAC 96 Proc. 1996 Internat. Symp. Symbolic Algebraic Comput.*, New York, N. Y., 1996. ACM Press.

Lakshman Y. N., B. Char, and J. Johnson. Software components using symbolic computation for problem solving environments. In Gloor (1998), pages 46–53.

R. Lambert. *Computational Aspects of Discrete Logarithms*. PhD thesis, University of Waterloo, 1996.

Derek F. Lawden. *Elliptic Functions and Applications*, volume 80 of *AMS*. Springer-Verlag, 1989.

D. Lazard. Résolution des systèmes d'équation algébriques. *Theoretical Comput. Sci.*, 15:77–110, 1981.

H. Le and C. Howlett. Client-server communication standards for mathematical computation. In Dooley (1999), pages 299–306.

A. K. Lenstra. Factoring multivariate polynomials over algebraic number fields. *SIAM J. Comp.*, 16:591–598, 1987.

A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

A. H. M. Levelt, editor. *Proc. 1995 Internat. Symp. Symbolic Algebraic Comput. ISSAC'95*, New York, N. Y., 1995. ACM Press.

R. Loos. Computing in algebraic extensions. In B. Buchberger, G. Collins, and R. Loos, editors, *Computer Algebra, 2nd ed.*, pages 173–187. Springer Verlag, Vienna, 1982.

R. G. K. Loos. Toward a formal implementation of computer algebra. *SIGSAM Bulletin*, 8(3):9–16, August 1974. Proc. EUROSAM'74.

F. S. Macaulay. *Algebraic theory of modular systems*, volume 19 of *Cambridge Tracts*. Cambridge Univ., England, 1916.

J. L. Massey. Shift-register synthesis and BCH decoding. *IEEE Trans. Inf. Theory*, IT-15:122–127, 1969.

R. J. Minnichelli, J. J. Anagnost, and C. A. Desoer. An elementary proof of Kharitonov's stability theorem with extensions. *IEEE Trans. Automatic Control*, 34(9):995–998, 1989.

M. B. Monagan. Gauss: A parameterized domain of computation system with support for signature functions. In A. Miola, editor, *Proc. DISCO '93*, volume 722 of *Lect. Notes Comput. Sci.*, pages 81–94, Heidelberg, Germany, 1993. Springer Verlag.

T. Mulders and A. Storjohann. Diophantine linear system solving. In Dooley (1999), pages 181–188.

D. R. Musser. Multivariate polynomial factorization. *J. ACM*, 22(2):291–308, April 1975.

D. R. Musser and A. Saini. *STL Tutorial and Reference Guide C++ Programming with the Standard Template Library*. Addison-Wesley Publ. Comp., Reading, Massachusetts, 1996.

A. Norman and J. Fitch. Interfacing REDUCE to Java. In J. Calmet and C. Limongelli, editors, *Proc. DISCO '96*, volume 1128 of *Lect. Notes Comput. Sci.*, pages 271–276, Heidelberg, Germany, 1996. Springer Verlag.

A. Odlyzko. Computer algebra and its applications: Where are we going? Paper accompanying a lecture at the Joint Annual Meeting of the German GI and the Austrian Computergesellschaft in Klagenfurt, Austria, September 1996.

M. Petkovšek, H. S. Wilf, and D. Zeilberger. *A=B*. A K Peters, Wellesley, Massachusetts, USA, 1996.

S. Poljak and J. Rohn. Checking robust nonsingularity is NP-hard. *Math. Control Signals Systems*, 6:1–9, 1993.

R. H. Risch. The problem of integration in finite terms. *Trans. Amer. Math. Soc.*, 139: 167–189, 1969.

A. Schönhage. Quasi-GCD computations. *J. Complexity*, 1:118–137, 1985.

Kiyoshi Shirayanagi. Floating point Gröbner bases. *Math. and Comput. in Simulation*, 42:509–528, 1996.

V. Shoup. Fast construction of irreducible polynomials over finite fields. *J. Symbolic Comput.*, 17(5):371–391, May 1994.

V. Shoup. A new polynomial factorization algorithm and its implementation. *J. Symbolic Comput.*, 20(4):363–397, 1995.

V. Shoup. NTL: A library for doing number theory (version 3.1b). Link on web document `http://www.cs.wisc.edu/~shoup/`, Univ. Wisconsin, Dept. Comput. Sci, 1998.

V. Shoup. Efficient computation of minimal polynomials in algebraic extensions of finite fields. In Dooley (1999), pages 53–58.

H. J. Stetter. Stabilization of polynomial systems solving with Groebner bases. In Lakshman Y. N. (1996), pages 117–124.

David R. Stoutemyer. Crimes and misdemeanors in the computer algebra trade. *Notices of the AMS*, 38(7):778–785, September 1991.

J. Teitelbaum. Euclid's algorithm and the Lanczos method over finite fields. *Math. Comput.*, 67(224):1665–1678, October 1998.

B. M. Trager. Algebraic factoring and rational function integration. In *Proc. 1976 ACM Symp. Symbolic Algebraic Comp.*, pages 219–228, New York, 1976. ACM.

G. Villard. Further analysis of Coppersmith's block Wiedemann algorithm for the solution of sparse linear systems. In Küchlin (1997), pages 32–39.

G. Villard. A study of Coppersmith's block Wiedemann algorithm using matrix polynomials. Rapport de Recherche 975-I-M, Institut d'Informatique et de Mathématiques Appliquées de Grenoble, `www.imag.fr`, April 1997b.

P. S. Wang. Design and protocol for Internet accessible mathematical computation. In Dooley (1999), pages 291–298.

S. M. Watt, P. A. Broadbery, S. S. Dooley, P. Iglio, S. C. Morrison, J. M. Steinbach, and R. S. Sutor. A first report on the $A^\sharp$ compiler. In *ISSAC '94 Proc. Internat. Symp. Symbolic Algebraic Comput.*, pages 25–31, New York, N. Y., 1994. ACM Press.

P. J. Weinberger and L. P. Rothschild. Factoring polynomials over algebraic number fields. *ACM Trans. Math. Software*, 2:335–350, 1976.

D. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory*, IT-32:54–62, 1986.

Lihong Zhi and Wenda Wu. Nearest singular polynomial. *J. Symbolic Comput.*, 26 (6):667–675, 1998. Special issue on Symbolic Numeric Algebra for Polynomials S. M. Watt and H. J. Stetter, editors.