Computing in the Field of Complex Algebraic Numbers

ADAM WOJCIECH STRZEBOŃSKI

Wolfram Research Inc., 100 Trade Centre Drive, Champaign, IL 61820, U.S.A.
and Department of Mathematics, Jagiellonian University, Kraków, Poland

In this paper we present two methods of computing with complex algebraic numbers. The first uses isolating rectangles to distinguish between the roots of the minimal polynomial, the second method uses validated numeric approximations. We present algorithms for arithmetic and for solving polynomial equations, and compare implementations of both methods in Mathematica.

1. Representation of Complex Algebraic Numbers

Let $K$ denote the field of complex algebraic numbers, i.e. the field of all complex numbers that are algebraic over the rationals. For every complex algebraic number $a$ there is a unique monic polynomial $f$ with rational coefficients (called the minimal polynomial of $a$) such that $f(a) = 0$ and for every polynomial $g$ with rational coefficients if $g(a) = 0$ then $f$ divides $g$. The minimal polynomial is necessarily irreducible.

To specify an algebraic number we give its minimal polynomial, and also we need to tell which root of this polynomial we have in mind. To distinguish between the roots of a given polynomial we find disjoint isolating sets in the complex plane, such that each set contains exactly one root of the polynomial.

The first method described here uses one-point isolating sets for rational roots, open intervals with rational endpoints for real roots, and open rectangles in the complex plane (Cartesian products of open intervals with rational endpoints) for complex roots. For real root isolation we use the method described by Akritas et al. (1994), based on the Descartes' Rule of Signs (see also Akritas and Collins (1976), and Akritas (1980)). Our complex root isolation algorithm is based on Collins and Krandick (1992). An algebraic number is represented by the minimal polynomial and the standard isolating rectangle given by our root isolation procedure—so that the representation of a given algebraic number is unique. (This makes checking equality of two algebraic numbers an easy task, otherwise to check equality we would have to count roots of the minimal polynomial in the intersection of isolating rectangles.)

The second approach is to identify roots of a given polynomial based on their variable-precision floating-point approximations (which corresponds to giving circular isolating
sets). We use the Jenkins–Traub algorithm (Jenkins, 1969, see also Ralston and Rabinowitz, 1978) to find numeric approximations of roots. To calculate the justified number of digits of precision in each approximation we use case \( k = 1 \) of the following proposition.

**Proposition 1.1.** Let \( x \) be a complex number, and let \( x_1, \ldots, x_n \) be all the roots of a polynomial \( f \) numbered so that \( |x - x_1| \leq |x - x_2| \leq \cdots \leq |x - x_n| \). Then

\[
|x - x_1| \leq \left( \frac{n!}{(n-k)!} \left| \frac{f(x)}{f^{(k)}(x)} \right| \right)^{1/k},
\]

for all \( k \) such that \( f^{(k)}(x) \neq 0 \).

**Proof.** We have

\[
\left| \frac{f(x)}{f^{(k)}(x)} \right| \geq \frac{\prod_{j=1}^n |x - x_j|}{\sum_{(i_1, \ldots, i_k)} \prod_{j \notin \{i_1, \ldots, i_k\}} |x - x_j|} \geq \frac{\prod_{j=1}^n |x - x_j|}{\frac{n!}{(n-k)!} \prod_{j=k+1}^n |x - x_j|}
= \frac{(n-k)!}{n!} \prod_{j=1}^k |x - x_j| \geq \frac{(n-k)!}{n!} |x - x_1|^k.
\]

We compute the explicit error bounds for values of \( f(x) \) and \( f'(x) \), so that we get a guaranteed upper bound for the right hand side of (1). For another method of a posteriori validation of complex roots see Aberth (1988).

We will call variable-precision floating-point approximations of roots of \( f \) good approximations if they are all different up to precision (more than the last 7 bits differ). If the constructed approximations are good we have the roots isolated, if not we increase the precision of computations in the numeric method and repeat the whole process. The precision we used in Jenkins–Traub method to compute the approximate value is stored with the algebraic number.

The correctness of the above algorithm follows from the convergence of the Jenkins–Traub algorithm and from the following lemma.

**Lemma 1.1.** If \( n > 1 \), \( x_1, \ldots, x_n \) are all the roots of a squarefree polynomial \( f \), and \( y_1, \ldots, y_n \) are complex numbers such that

\[
\forall 1 \leq i \leq n |y_i - x_i| < \frac{1}{3n} \min_{j \neq i} |y_i - x_j|,
\]

then the circles \( B \left( y_i, \frac{f(y_i)}{f'(y_i)} \right) \) are pairwise disjoint.

**Proof.** Without a loss of generality we may assume that \( f \) is monic. Fix \( 1 \leq i \leq n \), and take \( j \neq i \) such that \( |y_i - x_j| = \min_{k \neq i} |y_i - x_k| \). We have

\[
|f'(y_i)| = \prod_{k \neq i} (y_i - x_k) + \sum_{m \neq i} \prod_{k \neq m} (y_i - x_k) \geq \prod_{k \neq i} |y_i - x_k| - |y_i - x_i| \cdot \sum_{m \neq i} \prod_{k \neq m} |y_i - x_k|
\geq (|y_i - x_j| - (n-1)|y_i - x_i|) \prod_{k \neq i,j} |y_i - x_k| > (2n + 1)|y_i - x_i| \prod_{k \neq i,j} |y_i - x_k|.
\]
Hence
\[ n \left| \frac{f(y_i)}{f'(y_i)} \right| < n \frac{|y_i - x_j|}{(2n + 1)} = \frac{|y_i - x_j|}{2} - \frac{|y_i - x_j|}{2(2n + 1)} \leq \frac{|y_i - x_j|}{2} - \frac{|y_i - x_i|}{2}. \]

Therefore, for any \( 1 \leq i, j \leq n, i \neq j, \)
\[ n \left| \frac{f(y_i)}{f'(y_i)} \right| + n \left| \frac{f(y_j)}{f'(y_j)} \right| < \frac{|y_i - x_j|}{2} - \frac{|y_i - x_i|}{2} + \frac{|y_j - x_i|}{2} - \frac{|y_j - x_j|}{2} = \frac{1}{2}. \]

In theory the Jenkins–Traub algorithm, when used with sufficiently high precision of computations, is guaranteed to provide arbitrarily good root approximations. The practical implementations, however, are known to fail in some cases (as pointed out to me by Werner Krandick, and as demonstrated in Gourdon, 1993), therefore, in such cases, it may be a good idea to switch to the exact Collins–Krandick root isolation algorithm.

2. Arithmetic

Minimal polynomials of sum, difference, product, quotient, and rational power of algebraic numbers can be computed using the following theorem (see Loos (1982) for the proof).

**Theorem 2.1.** If \( f \) and \( g \) are the minimal polynomials for algebraic numbers \( a \) and \( b \) (\( b \neq 0 \)), and if \( p/q \) is a rational number then the minimal polynomials of \( a + b \), \( a - b \), \( a \cdot b \), \( a/b \), and \( a^{p/q} \) divide the polynomials given in Table 1.

Our task now is to decide which root of the polynomial given in Table 1 is the result of the arithmetic operation.

2.1. Roots represented by isolating rectangles

In the algorithms for algebraic number arithmetic we will need to perform arithmetic operations on isolating rectangles. It is well known how to perform real interval arithmetic (see Moore, 1979). Now suppose that we have two rectangles in the complex plane: \( R = A + B \cdot I \) and \( S = C + D \cdot I \), where \( A, B, C, \) and \( D \) are real intervals. To add, subtract, multiply, divide (here we assume \( 0 \notin \text{cl}(S) \)) \( R \) and \( S \), or raise \( R \) to a natural power \( n \) we use the following facts:

\[
R \pm S = (A \pm C) + (B \pm D)I \\
R \cdot S \subseteq (AC - BD) + (AD + BC)I \\
R/S \subseteq (AC + BD)/(C^2 + D^2) + (BC - AD)/(C^2 + D^2)I \\
R^n \subseteq \sum_{k=0}^{\lfloor n/2 \rfloor} \binom{n}{k} (-1)^k A^{n-2k} B^{2k} + I \sum_{k=0}^{\lfloor (n-1)/2 \rfloor} \binom{n-1}{2k+1} (-1)^k A^{n-2k-1} B^{2k+1}.
\]

We will also need an algorithm for making an isolating rectangle of an algebraic number \( a \) smaller. We use a rectangle bisection method described by Collins and Krandick (1992), and the following hybrid method.
Table 1.

<table>
<thead>
<tr>
<th>Algebraic number</th>
<th>Polynomial divisible by its minimal polynomial</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a + b )</td>
<td>( \text{Resultant}_y(f(x - y), g(y)) )</td>
</tr>
<tr>
<td>( a - b )</td>
<td>( \text{Resultant}_y(f(x + y), g(y)) )</td>
</tr>
<tr>
<td>( a \cdot b )</td>
<td>( \text{Resultant}_y(y^\text{degree}(f(x/y)), g(y)) )</td>
</tr>
<tr>
<td>( a/b )</td>
<td>( \text{Resultant}_y(f(xy), g(y)) )</td>
</tr>
<tr>
<td>( a^{p/q} )</td>
<td>( \text{Resultant}_y(f(y), x^q - y^p) )</td>
</tr>
</tbody>
</table>

(i) Try to compute a numeric approximation of \( a \) by supplying the middle point of the isolating rectangle \( R \) as a starting point of the second stage of the Jenkins–Traub algorithm.

(ii) If the numeric algorithm does not converge go to (iv).

(iii) Use inequality (1) to find a rectangle \( R_1 \) containing at least one root of the minimal polynomial of \( a \). If \( R_1 \subset R \) put \( R := R_1 \) and go to (v), otherwise go to (iv).

(iv) Bisect \( R \) several times.

(v) If the new \( R \) is sufficiently small return \( R \), otherwise go to (i) with increased precision of computations.

The computing time of the rectangle bisection method grows much faster with the required precision than the computing time of the hybrid method. The rectangle bisection method is, however, more effective when we need to make large rectangles only a few bisections smaller. The comparison of computing times of both methods is given in the last section of this paper. For a different hybrid symbolic-numeric rectangle refinement method see Collins and Krandick (1996).

Now we have all subalgorithms needed to describe the arithmetic algorithm. Suppose we are given two algebraic numbers \( a \) and \( b \) to add, subtract, multiply, or divide, or one algebraic number to raise to a natural power. Let \( c \) denote the resulting algebraic number. We know the minimal polynomials and isolating rectangles for \( a \) and \( b \), and from Table 1 we can find a polynomial \( f \) divisible by the minimal polynomial of \( c \). We need to find a standard representation of \( c \), i.e. the factor of \( f \) which is the minimal polynomial of \( c \), and the standard isolating rectangle of \( c \). We have two ways of achieving this goal.

The first method is as follows.

(i) Isolate the roots of each of the factors of \( f \), get a list of pairs (polynomial, isolating rectangle).

(ii) Perform the arithmetic operation on the isolating rectangles of \( a \) and \( b \), get a rectangle \( R \) containing \( c \).

(iii) Remove from list those elements for which isolating rectangle does not intersect \( R \).

(iv) If more than one element is left make the isolating rectangles of \( a \), \( b \), and all elements left on the list smaller (remembering the original isolating rectangles for elements of the list, because these are the standard isolating rectangles) and go to (ii). Else the only element on the list contains the minimal polynomial and standard isolating rectangle of \( c \).
The second method is as follows.

(i) Perform the arithmetic operation on the isolating rectangles of \(a\) and \(b\), get a rectangle \(R\) containing \(c\).

(ii) Count the number of roots of subsequent factors of \(f\) in \(R\), using the method described by Collins and Krandick (1992).

(iii) If more than one root was found stop counting, make the isolating rectangles of \(a\) and \(b\) smaller and go to (i). If there is only one root of one factor in \(R\), then this factor \(g\) is the minimal polynomial of \(c\).

(iv) To find the standard isolating rectangle of \(c\) isolate the roots of \(g\) and check which one of the isolating rectangles contains a root of \(g\) in its intersection with \(R\).

Before dividing two rectangles we make sure that zero is not in the closure of the second rectangle, making it smaller if needed.

A considerable speed-up can be obtained by use of real-root isolation instead of complex-root isolation when we know that \(c\) is real (for example if \(a\) and \(b\) are real, or we are computing Re, Im, or Abs of an algebraic number).

Our experience shows that the second method is faster than the first when there is more than one factor of \(f\) or the result is real and the operation is not exponentiation.

A different method is used for computing the \(n\)th root of an algebraic number \(a \neq 0\) (since we do not have an algorithm for computing the \(n\)th root of a rectangle). As the branch cut line for \(n\)th root function we take the negative real half-line, so that the image is the sector \(A := \{ z : -\pi/n < \text{Arg}(z) \leq \pi/n \}\). Suppose \(a\) is not a negative real number and let \(p\) denote the minimal polynomial of \(a\), \(k\) the degree of \(p\), and \(c\) the \(n\)th root of \(a\), and let \(f(x) := p(x^n)\). Note that if \(m\) is the number of negative real roots of \(p\) then the number of roots of \(f\) in \(\text{int}(A)\) is \(k - m\). We can compute \(c\) using the following algorithm.

(i) Compute \(k - m\).

(ii) Isolate the roots of each of the factors of \(f\), get a list of pairs \((\text{polynomial}, \text{isolating rectangle})\). Let \(\text{list}_1\) be the empty list.

(iii) For each element of \(\text{list}\) if \(\text{isolating rectangle}\) is contained in \(A\) remove the element from \(\text{list}\) and put it on \(\text{list}_1\), if \(\text{isolating rectangle}\) is disjoint from \(A\) remove the element from \(\text{list}\).

(iv) If the length of \(\text{list}_1\) is less than \(k - m\) make isolating rectangles of elements of \(\text{list}\) smaller (remembering the original standard values) and go to (iii).

(v) Remove from \(\text{list}_1\) all elements for which the \(n\)th power of the isolating rectangle does not intersect the isolating rectangle of \(a\).

(vi) If more than one element is left make the isolating rectangles of all elements remaining on \(\text{list}_1\) smaller and go to (v). Otherwise the only element on \(\text{list}_1\) contains the minimal polynomial and standard isolating rectangle of \(c\).

If \(a\) is a negative real number, then to compute the \(n\)th root of \(a\) we multiply the \(n\)th root of \(-a\) by the \(n\)th root of \(-1\) (which for this purpose can be represented by a squarefree polynomial \(x^n + 1\) and a suitable isolating rectangle).

2.2. Roots represented by variable-precision numeric approximations

To perform arithmetic on algebraic numbers we may need to increase the precision of good (as defined in Section 1) approximations. Let \(s\) be a good approximation of an
algebraic number \( a \), and let \( \text{prec} \) be the precision of computations in the Jenkins–Traub algorithm used to compute \( s \). We use the following algorithm:

(i) Try to compute a numeric approximation of \( a \) by supplying the current approximation \( s \) as a starting point of the second stage of the Jenkins–Traub algorithm, and using a precision of computations higher than \( \text{prec} \).
(ii) If the numeric algorithm does not converge go to (iv).
(iii) Use inequality (1) to find a justified precision of the new approximation. If the new approximation is equal to \( s \) within precision and if the precision of the new approximation is high enough return the new approximation. If the precision of the new approximation is not high enough increase the precision of computations and go to (i). If the new approximation is not equal to \( s \) go to (iv).
(iv) Compute approximation of all the roots of the minimal polynomial using a precision of computations higher than \( \text{prec} \), and find justified precisions of all approximations.
(v) Choose the approximations equal to \( s \) within precision. If only one number \( s_1 \) is chosen, then if \( s_1 \) has more digits of precision than \( s \) put \( s := s_1 \).
(vi) If precision of the new \( s \) is high enough return \( s \), otherwise increase the precision of computations and go to (i).

Correctness of the above algorithm follows from the correctness of Jenkins–Traub algorithm.

Here is the algorithm we use to add, subtract, multiply, or divide two algebraic numbers \( a \) and \( b \), or to raise an algebraic number to a rational power (\( c \) denotes the resulting algebraic number.)

(i) Find a polynomial divisible by the minimal polynomial of \( c \) in Table 1 and factor it.
(ii) Isolate the roots of each of the factors of \( f \), get a list of pairs (polynomial, good approximation of a root).
(iii) Perform the arithmetic operation on the approximations of \( a \) and \( b \), get a variable-precision floating-point approximation \( r \) of \( c \).
(iv) Remove from list those elements for which approximation of root is not equal to \( r \) (up to precision).
(v) If more than one element is left compute higher precision approximations of \( a \), \( b \), and all elements left on the list and go to (iii). Else the only element on the list contains the minimal polynomial and a good approximation of \( c \).

3. Solving Polynomial Equations

In this section we describe algorithms for solving univariate polynomial equations with complex algebraic number coefficients. Algorithms using Gröbner bases then allow us to solve systems of multivariate polynomial equations with complex algebraic coefficients. (We do not necessarily need to compute Gröbner bases over the field of algebraic numbers. We can substitute a new variable for each algebraic number, add minimal polynomials to the system of equations, compute a Gröbner basis of the new system over the rationals with the new variables being smallest in the ordering, and then construct solutions only over the initial algebraic numbers.)

Suppose we are given a polynomial \( f(x) \) with coefficients being polynomials in complex algebraic numbers \( a_1, \ldots, a_n \). Our task is to find roots of this polynomial.
LEMMA 3.1. Let $F_0(x, z_1, \ldots, z_n)$ be the polynomial obtained by substituting variables $z_1, \ldots, z_n$ for algebraic numbers $a_1, \ldots, a_n$ in $f(x)$, and let $g_i(z_i)$ be the minimal polynomial of $a_i$ for $1 \leq i \leq n$. Define recursively

$$F_i(x, z_{i+1}, \ldots, z_n) := \text{Resultant}_{z_i}(F_{i-1}(x, z_1, \ldots, z_n), g_i(z_i))$$

Then the polynomial $G(x) := F_n(x)$ has rational coefficients and all roots of $f(x)$ are also roots of $G(x)$.

PROOF. By properties of the resultant $F_i(x, z_{i+1}, \ldots, z_n) = 0$ iff $F_{i-1}(x, z_i, \ldots, z_n)$ and $g_i(z_i)$ have common root as polynomials in $z_i$. By inductive reasoning (starting with $F_n$) we conclude that $F_n(x) = 0$ iff there exist roots $b_1, \ldots, b_n$ of polynomials $g_1, \ldots, g_n$ such that $F_0(x, b_1, \ldots, b_n) = 0$. $\Box$

In particular $G(x) \equiv 0$ iff for some roots $b_1, \ldots, b_n$ of polynomials $g_1, \ldots, g_n$ we have $F_0(x, b_1, \ldots, b_n) \equiv 0$. If $G(x) \equiv 0$ we simplify subsequent coefficients of $f(x)$ to single algebraic numbers until we find one which is not zero. This way we either prove that $f(x) \equiv 0$ or we get $f(x) = f_1(x)$ and the leading coefficient of $f_1(x)$ is expressed as a single non-zero algebraic number, so it remains non-zero when we replace it with other roots of its minimal polynomial. Therefore the polynomial $G(x)$ from the lemma computed for $f_1(x)$ is not identically zero.

In the algorithms described below we assume that we already made the necessary reduction and obtained a non-zero $G(x)$.

### 3.1. Roots represented by isolating rectangles

(i) Isolate the roots of each of the factors of $G$, get a list of pairs (polynomial, isolating rectangle).

(ii) Compute the finite sequence of non-constant polynomials defined recursively by $d_1(x) = f(x), d_{i+1}(x) = \text{g.c.d.}(d_i(x), d'_i(x))$.

(iii) Substitute the isolating rectangles for $a_1, \ldots, a_n$ in $\{d_i(x)\}$, get polynomials $\{r_i(x)\}$ with rectangle coefficients.

(iv) Put solutions := empty list, and choicelist := list. For (i) running from 1 to the number of $r_i$s:

- Remove from choicelist those elements for which $r_i$ evaluated at the isolating rectangle does not contain zero.
- Append all elements left on choicelist to solutions.

(v) If the number of elements of solutions is greater than the degree of $f$, put list to be all different elements of solutions, make isolating rectangles of $a_1, \ldots, a_n$ and all elements of list smaller (remembering the original, standard isolating intervals), and go to (iii). Else solutions is the list of minimal polynomials and isolating rectangles for all roots of $f$, multiple roots appearing the correct number of times.

To compute the g.c.d. of univariate polynomials with algebraic number coefficients we use Euclid’s algorithm.

Polynomials are evaluated at rectangles using the Horner scheme. Evaluation of multivariate polynomials is done recursively, one variable at a time.
3.2. Roots represented by variable-precision numeric approximations

(i) Isolate the roots of each of the factors of \( G(x) \), get a list of pairs (polynomial, good approximation of a root).

(ii) Replace \( a_1, \ldots, a_n \) in \( f \) with their numeric approximations, get polynomial \( f_a(x) \).

(iii) Compute the approximate solutions of \( f_a(x) \), and use (1) to find justified precision of each solution. (We may need to use higher derivative cases of Proposition 1.1 if both \( f \) and \( f' \) evaluate to zero up to precision.)

(iv) For every solution of \( f_a(x) \) choose from list those elements for which approximation of root is equal to \( r \) (up to precision).

(v) If more than one element is chosen for any solution increase the precision of approximations of \( a_1, \ldots, a_n \), and go to (i) with higher precision of computations. Else the elements chosen are the minimal polynomials and good approximations of roots of \( f \).

The correctness of the above algorithm follows from the convergence of the Jenkins–Traub algorithm combined with the fact that if \( x \) approaches \( x_1 \) the right-hand side of (1) approaches zero for all \( 1 \leq k \leq n \).

4. Implementation of Algorithms and Computing Times

In this section we give a sample of computing times for algorithms presented in this paper. We compare computing times for both representations of complex algebraic numbers. Computations with algebraic numbers represented by the minimal polynomial and the corresponding isolating rectangles and variable-precision floating point approximations are marked (IR) and (FPA), respectively. Both root isolation algorithms are implemented in the C-kernel of Mathematica, all the other algorithms are implemented in the Mathematica programming language. Computing times are shown in seconds. The timings were obtained on a NeXT workstation with a 25 MHz processor and 32 MB of RAM.

In Table 2 we compare computing times of two methods of refining isolating rectangles, the rectangle bisection method (RBM), described by Collins and Krandick (1992), and the hybrid method (HM), described in Section 2.1. For each degree we used a polynomial with 10-bit pseudo-random coefficients, which had a single root in \((0,1) \times (0,1)\), or in \((0,1) \times (0,1/2)\) for degree 20.

Tables 3 and 4 contain computation times for arithmetic operations. Degree is the degree of the minimal polynomial of the result. In the Plus, Times, and Divide columns, degrees 9, 12, 16, and 20, respectively, are obtained by adding, multiplying or dividing algebraic numbers with minimal polynomials of degrees 3 and 3, 3 and 4, 4 and 4, and 4 and 5, respectively. In the Power column, degrees 9, 12, 16, and 20, respectively, are obtained by raising algebraic numbers with minimal polynomials of degrees 3, 6, 4, and 4 to powers 2/3, 3/2, 5/4, and 4/5, respectively. Minimal polynomials of the input algebraic numbers are chosen with pseudo-random 10-bit coefficients. All input algebraic numbers in Table 3 are not real, while all input algebraic numbers in Table 4 are real (cases of real and complex results of power of a real number are shown separately).

We see that for a complex input or result (FPA) is 2.5 to 20 times faster than (IR), and the difference increases with the degree. If both the input and the result contain only real numbers (IR) is usually slightly faster than (FPA). The advantage of (IR) here is that we can isolate only the real roots of polynomials involved, while if we use a numeric
Table 2. Refining isolating rectangles

<table>
<thead>
<tr>
<th>Degree</th>
<th>Accuracy goal</th>
<th>(RBM)</th>
<th>(HM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>$10^{-10}$</td>
<td>15.6</td>
<td>1.91</td>
</tr>
<tr>
<td>5</td>
<td>$10^{-20}$</td>
<td>45.1</td>
<td>2.80</td>
</tr>
<tr>
<td>5</td>
<td>$10^{-20}$</td>
<td>266</td>
<td>5.13</td>
</tr>
<tr>
<td>5</td>
<td>$10^{-100}$</td>
<td>1283</td>
<td>12.3</td>
</tr>
<tr>
<td>10</td>
<td>$10^{-10}$</td>
<td>95.3</td>
<td>4.67</td>
</tr>
<tr>
<td>10</td>
<td>$10^{-20}$</td>
<td>375</td>
<td>7.16</td>
</tr>
<tr>
<td>10</td>
<td>$10^{-50}$</td>
<td>3376</td>
<td>18.6</td>
</tr>
<tr>
<td>10</td>
<td>$10^{-100}$</td>
<td>?</td>
<td>49.6</td>
</tr>
<tr>
<td>20</td>
<td>$10^{-10}$</td>
<td>342</td>
<td>15.3</td>
</tr>
<tr>
<td>20</td>
<td>$10^{-20}$</td>
<td>1305</td>
<td>20.3</td>
</tr>
<tr>
<td>20</td>
<td>$10^{-50}$</td>
<td>12043</td>
<td>37.1</td>
</tr>
<tr>
<td>20</td>
<td>$10^{-100}$</td>
<td>?</td>
<td>73.9</td>
</tr>
</tbody>
</table>

Table 3. Complex algebraic number arithmetic

<table>
<thead>
<tr>
<th>Degree</th>
<th>Plus (IR/FPA)</th>
<th>Times (IR/FPA)</th>
<th>Divide (IR/FPA)</th>
<th>Power real (IR/FPA)</th>
<th>Power complex (IR/FPA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>11.0/4.09</td>
<td>9.41/3.23</td>
<td>13.9/4.66</td>
<td>22.4/2.25</td>
<td>13.0/1.59</td>
</tr>
<tr>
<td>12</td>
<td>32.7/4.80</td>
<td>40.6/3.61</td>
<td>26.6/4.90</td>
<td>42.1/3.98</td>
<td>25.6/3.12</td>
</tr>
<tr>
<td>16</td>
<td>50.3/6.79</td>
<td>42.6/6.98</td>
<td>83.5/6.90</td>
<td>87.5/4.36</td>
<td>57.7/4.00</td>
</tr>
<tr>
<td>20</td>
<td>83.9/7.47</td>
<td>232./8.33</td>
<td>107./8.71</td>
<td>106./5.72</td>
<td></td>
</tr>
</tbody>
</table>

method we have to compute approximations of all the roots, because we do not know how many of them are real.

In Table 5 we present computing times for solving equations with algebraic number coefficients. For each equation we give the degree of polynomial $G$ from Lemma 3.1, and the highest degree among the minimal polynomials of roots of the equation. Here are the polynomials we use:

\[
\begin{align*}
    f_1(x) &:= x^5 + \sqrt{2}x + I \\
    f_2(x) &:= x^3 + (\sqrt{2} + \sqrt{3})x^2 + (\sqrt{2} + 7)x - 1 \\
    f_3(x) &:= x^{10} + (\sqrt{7} + 5)x^3 + 3\sqrt{7} - 11 \\
    f_4(x) &:= x^3 + ax + b,
\end{align*}
\]

where $a$ and $b$ are roots of $x^3 - 7x - 5$ with approximate values $-0.7828156787$ and $2.9488283581$, respectively.

Table 4. Real algebraic number arithmetic

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>3.40/4.05</td>
<td>2.69/2.98</td>
<td>4.52/4.55</td>
<td>2.26/1.86</td>
<td>13.0/1.59</td>
</tr>
<tr>
<td>12</td>
<td>3.25/2.75</td>
<td>2.65/3.45</td>
<td>4.56/4.76</td>
<td>3.72/3.17</td>
<td>25.6/3.12</td>
</tr>
<tr>
<td>16</td>
<td>6.62/4.93</td>
<td>6.95/5.42</td>
<td>6.74/6.47</td>
<td>2.99/3.72</td>
<td>57.7/4.00</td>
</tr>
<tr>
<td>20</td>
<td>4.08/7.33</td>
<td>7.02/8.23</td>
<td>6.28/8.76</td>
<td>3.75/5.12</td>
<td>96.5/5.14</td>
</tr>
</tbody>
</table>
Table 5. Equations with algebraic number coefficients

<table>
<thead>
<tr>
<th>Equation</th>
<th>Degree of $G$</th>
<th>Maximum degree of roots</th>
<th>(IR)</th>
<th>(FPA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1(x) = 0$</td>
<td>20</td>
<td>20</td>
<td>172</td>
<td>6.00</td>
</tr>
<tr>
<td>$f_2(x) = 0$</td>
<td>20</td>
<td>20</td>
<td>219</td>
<td>5.94</td>
</tr>
<tr>
<td>$f_3(x) = 0$</td>
<td>27</td>
<td>18</td>
<td>141</td>
<td>6.28</td>
</tr>
<tr>
<td>$f_4(x) = 0$</td>
<td>28</td>
<td>28</td>
<td>121</td>
<td>6.15</td>
</tr>
<tr>
<td>$f_5(x) = 0$</td>
<td>40</td>
<td>4</td>
<td>255</td>
<td>42.3</td>
</tr>
<tr>
<td>$f_6(x) = 0$</td>
<td>75</td>
<td>60</td>
<td>10342</td>
<td>62.0</td>
</tr>
</tbody>
</table>

\[
f_5(x) := x^7 + 4Ix^3 + I\sqrt{3}x - \sqrt{3}(5 + I)
\]
\[
f_6(x) := -10\sqrt{2} - 8\sqrt{3} + (38 + 16\sqrt{6})x - (29\sqrt{2} - 24\sqrt{3})x^2 + (23 + 8\sqrt{6})x^3 - (5\sqrt{2} - 2\sqrt{3})x^4 + x^5 = (x - \sqrt{2})^4(x - \sqrt{2} - \sqrt{3})^2
\]
\[
f_7(x) := (a + 2b)x^3 - abx + a^2 + b^2,
\]

where $a$ and $b$ are roots of $x^5 - x - 1$ with approximate values $1.1673039783$ and $-0.7648844336 + 0.3524715461I$, respectively.

The (FPA) method is here much faster than (IR). The big advantage of the (FPA) method is that we can solve the equation with approximate coefficients and get exact bounds on solutions, and we do not have to check for multiple roots (i.e. compute g.c.d. in multiple algebraic extensions of the rationals).

References


Originally received 22 July 1994

Accepted 2 October 1997