Solving Cyclotomic Polynomials by Radical Expressions

Andreas Weber and Michael Keckeisen*

Abstract: We describe a Maple package that allows the solution of cyclotomic polynomials by radical expressions. We provide a function that is an extension of the Maple *solve* command. The major algorithmic ingredient of the package is an improvement of a method due to Gauss which gives radical expressions for roots of unity. We will give a summary for computations up to degree 100, which could be done within a few hours of cpu time on a standard workstation.

Keywords: solution in radicals, roots of unity, cyclotomic polynomials, Galois theory, Gauss algorithm

Introduction

Niels Henrik Abel and Evariste Galois showed that polynomial equations of degree higher than four cannot be solved by radical expressions in general. As Galois stated in his work, radical solutions exist if and only if the Galois group of the polynomial is solvable.

Since the Galois group of a cyclotomic polynomial is Abelian, its Galois group is solvable, and so its solutions can be expressed by radical expressions.

Carl Friedrich Gauss proved this special case already in 1797 and published it in 1801 in his master-work *Disquisitiones Arithmeticae* [1] as an extension of his work to determine which regular polygons can be constructed by compass and ruler. In this book Gauss describes an algorithm to compute radical expressions for primitive roots of unity.

We implemented a variant of the algorithm of Gauss in Maple, which has an improved time complexity compared to Gauss' original proposition.

The hard part of the proof and of the algorithm is to compute a radical expression for a primitive p-th root of unity, where p is a prime number. The improvement in our algorithm reduces the asymptotic time complexity in this case from

$$\mathrm{O}\left((p^5 + p^3\, m^6)\log(p)\right)$$

to

$$\mathrm{O}\left(p^3\, m^6 \log(p)\right) \quad ,$$

where m is the largest prime factor of $p-1$. An analysis of the algorithm and the statement and proof of the proposition on which the improvement is based can be found in [2]. We also refer to this paper for a description of the underlying ideas of Gauss' algorithm and the statement of the theorems that yield its correctness.

Compared to the Maple-code given in [2], we managed to improve the practical speed of the algorithm to a great extent, reducing the needed amount of memory at the same time. For instance, the computations of radical expressions for the 47th, 59th, or 83rd root of unity failed with the previous implementation, but can now be accomplished within a few hours.

*Arbeitsbereich Symbolisches Rechnen, Fakultät für Informatik, Universität Tübingen, 72076 Tübingen, Germany, E-mail: {weber,keckeise}@informatik.uni-tuebingen.de; WWW: http://www-sr.informatik.uni-tuebingen.de

The Algorithm

First, we will show how the task of computing radical solutions for cyclotomic polynomials can be reduced to the one of calculating radical expressions for roots of unity. Second, we will sketch the algorithm for the latter task, which mainly is the one given in [2].

RADICAL SOLUTIONS FOR CYCLOTOMIC POLYNOMIALS

The n-th cyclotomic polynomial over the rational numbers is of the form

$$p_n(x) = \frac{x^n - 1}{\gcd(x^n - 1, \prod_{j=1}^{n-1}(x^j - 1))}$$

and its degree is

$$\phi(n) = \prod_{i=1}^{s} \left({p_i}^{r_i} - {p_i}^{(r_i-1)}\right) \quad ,$$

where $n = \prod_{i=1}^{s} {p_i}^{r_i}$ and ϕ is the Eulerian ϕ-Function, cf. [3, p. 13 and 169]. From the defining formula of ϕ it can be seen that the inverse image $\phi^{(-1)}$ of ϕ is finite. Thus, to determine whether or not an irreducible factor q of a polynomial is cyclotomic, it suffices to compare the j-th cyclotomic polynomial with q for all j in $\phi^{(-1)}$. Maple provides the function *numtheory[invphi]* to calculate $\phi^{(-1)}$; the computational costs of this function is negligible for the range of arguments which are feasible for computing radical expressions.

Given the n-th cyclotomic polynomial $p_n(x)$, the task of computing $\phi(n)$ radical solutions can be reduced to finding a radical expression for one root of unity, because this root will

be primitive, e. g. a multiplicative generator for the others. Further, it suffices to find a primitive n-th root of unity to compute primitive roots of unity for all prime factors of *n* by applying some relatively simple recursion formulas, see e. g. [4].

RADICAL EXPRESSIONS FOR PRIMITIVE p-TH ROOTS OF UNITY

The hard part of the algorithm is to find a radical expression for a primitive p-th root of unity, where p is a prime number. This part involves the computation of *Gaussian periods*, cf. [1, § 343]. If ζ_p denotes a primitive p-th root of unity the *period* (f, k) for two integers f and k is defined by

$$(f,k) = \sum_{m=0}^{f-1} {\zeta_p}^{(k\, g^{(e\, m)})} \quad ,$$

where $e = \frac{p-1}{f}$ and g is a primitive p-th root. In Maple a primitive p-th root can be computed by the function *primroot* in the package *numtheory*.

The Gaussian periods are generators of intermediate fields of the number field $\mathrm{Q}(\zeta_p)$, which is a number field of degree $p-1$ over the rationals. Using the modern terminology of intermediate fields the algorithm of Gauss works roughly as follows. Starting with the rationals, compute a radical expression for a generator of an intermediate field of prime degree, i.e. an appropriate Gaussian period. After this task has been accomplished, compute a radical expression for a generator of another intermediate field (i.e. another Gaussian period) whose *relative degree* over the other intermediate field is of prime order. Continue this task until a radical expression for a generator of $\mathrm{Q}(\zeta_p)$, i.e. $\zeta_p = (1, g^0)$ has been computed.

The difficult part of Gauss' algorithm is the lifting of radical expressions from one intermediate field to another one. We will not give the details of this lifting here but we refer to [2].

The algorithm described above works for several sequences of intermediate fields to be constructed. All permutations on the list of prime factors of $p-1$ give a possible sequence in the construction of the intermediate fields. Different permutations give different results in general, and also the computational costs are greatly affected by the choice of an appropriate sequence of intermediate fields. Our default strategy in the choice of the intermediate fields is as follows: first take the largest prime factor, then the second largest and so on, until we just get the final field $\mathrm{Q}(\zeta_p)$ as a relative extension of degree 2 of the last intermediate field. Note that we have to count multiplicities as separate factors. This strategy seems to be preferable over others, because bigger relative extensions — which cause more computations — occur in smaller intermediate fields. However, as a tool for experimental mathematics we provide another strategy, too. The second strategy is simply to reverse the ordering; this can be done by setting *SwapPrimeOrder := true*. As had to be expected in all our experiments the obtained computation times were far worse than with our default setting and the computed radical expressions were also larger; some examples are given below in Table 2.

How to Use the Library

The library is included in the file 'radsolvelib'.

```
> read 'radsolvelib':
```

All functionality is available via the function *radsolve*.

The command *radsolve* serves as an extension to the Maple *solve* command. It returns radical solutions for all cyclotomic factors of a polynomial with integer coefficients and calls *solve* for the others.

- **Calling Sequence:** *radsolve(poly)*
- **Parameters:** *poly* — an univariate polynomial with integer coefficients
- The command *radsolve(poly)* returns the solutions of the univariate polynomial *poly* as radical expressions for all the irreducible factors of *poly* that are either cyclotomic or of degree lower than four.
- In general, explicit solutions in terms of radicals for polynomial equations of degree greater than four do exist if and only if the Galois group of the polynomial is solvable. This is the case for cyclotomic polynomials, since their Galois group is cyclic. *radsolve(poly)* computes radical expressions for such cases and calls *solve* for the other factors of poly. In cases, where explicit solutions can't be computed, implicit solutions are given in terms of *RootOfs*.
- Note that to obtain explicit solutions for the general quartic polynomial equation you have to set the global variable *_EnvExplicit* to true, as explained in the topic help to *solve.*
- The output from *radsolve* is a sequence of solutions.
- The behaving of *radsolve(poly)* can be changed by using the following global variables (default values in brackets):
- *_AllSolutions* (false): If *_AllSolutions* is set to false, *radsolve(poly)* returns only one solution (no matter which exponent belongs to the factor) for every irreducible factor of *poly*. If it is set to true, all solutions are given.
- *_UseRootsymbols* (false): if *_UseRootsymbols* is set to true, *RootOfs* are used to represent primitive roots of unity of prime factors of $n-1$, where n is the degree of

the cyclotomic polynomial, although these roots could be expressed in radicals explicitly. This saves time and space. If *_UseRootsymbols* is set to false, all RootOfs are substituted by their radical expressions.

- *_SwapPrimeOrder* (false): If *_SwapPrimeOrder* is set to true, then an alternative order of building intermediate generators of field extensions of order $\frac{p-1}{m}$, where p is the degree of the cyclotomic polynomial and m divides $p-1$, is chosen. This results in different radical expressions and in more computing time. Mainly for experimentation.

- To obtain information about the computing process, set *infolevel[radsolve]* to an appropriate level. Level 0 gives just the solutions, level 1 and level 2 are not used, level 3 gives a general outline, level 4 is nice to use for larger examples and level 5 is designed for those who are really interested in the algorithm.

EXAMPLES

The Maple *solve* command does not express the solutions of a cyclotomic factor of a polynomial of degree higher than 4 in radicals, but uses *sin* and *cos* functions instead.

```
> solve(x^7-1)[4];
```

$$-\cos(\frac{1}{7}\,\pi) + I\sin(\frac{1}{7}\,\pi)$$

In the following, we set *_AllSolutions* to *false*. Thus we get only one solution for any irreducible factor of the polynomial.

```
> restart:
> read 'radsolvelib':
> _AllSolutions:=false:
> lsols:=radsolve(x^7-1);
```

$$\begin{aligned} lsols := 1, &-\frac{1}{2}(-\frac{7}{3} + \frac{1}{3}\,\%2^{1/3}\,(-\frac{1}{2} + \frac{1}{2}\,\sqrt{-3})^4 \\ &+ \frac{1}{3}(6 - 2\,\%1 - \sqrt{-3})\,(-\frac{1}{2} + \frac{1}{2}\,\sqrt{-3})^2 \\ &/\%2^{1/3})^{1/2} - \frac{1}{6} + \frac{1}{6}\,\%2^{1/3} \\ &+ \frac{1}{6}\,\frac{6 - 2\,\%1 - \sqrt{-3}}{\%2^{1/3}} \\ \%1 &:= (-\frac{1}{2} + \frac{1}{2}\,\sqrt{-3})^2 \\ \%2 &:= 9\,\%1 + 8 - 6\,\sqrt{-3} \end{aligned}$$

Using the *radnormal* command we can verify that the radical expression, which was returned by *radsolve* as a solution for the non-trivial factor, is indeed a 7th root of unity.

```
> radnormal(lsols[2]^7-1);
```

$$0$$

To verify the results, one can use *radnormal* for smaller degrees only. One has to use numeric evaluation for larger degrees. The following method works for very large expressions.

```
> restart:
> read 'radsolvelib':
> _AllSolutions:=false:
> z:=radsolve(numtheory[cyclotomic](41,x)):
> Digits:=40:
> readlib(optimize):
> numeval:=`optimize/makeproc`(
>            map(evalf,[optimize(z)])):
> fnormal(numeval()^41-1,35);
```

$$0$$

Practical Limitations of the Algorithm

Compared to [2] the implementation of the main algorithm has been optimized. For results in Table 1 we applied *radsolve* on all cyclotomic polynomials of (prime) degree up to 101 on a Sun Ultrasparc I workstation. It can be seen that the computing time depends on the size of the largest prime factor of $p-1$ to a great extent, as had to be expected from the theoretical complexity analysis given in [2].

Memory usage was up to 20MB for *p=83*, less than 5MB for the rest. Table 1 was generated semi-automatically via the following script.

```
> restart:
> read 'radsolvelib':

> _UseRootsymbols:=true:
> _AllSolutions:=false:
> _SwapPrimeOrder:=false:
> readlib(cost):
> additions:='f':
> multiplications:='f':
> divisions:='f':
> functions:='r':
> assignments:='a':
> readlib(optimize):

> for i from 7 to 101 do
> if numtheory[isprime](i) then
>
> t:=round(time(
>      radsolve(
>        numtheory[cyclotomic](
>          i,x)))):
> z:= radsolve(numtheory[cyclotomic](i,x));
> printf(`\\n%5d %15A %20f %30A%30A`,
>        i,numtheory[ifactor](i-1),
>        t,cost(z),
>        cost(optimize(z)));
> fi:
> od:
```

It is interesting to see the results you obtain by setting *SwapPrimeOrder := true*. You get different radical expressions and it takes a lot more time and memory. Some examples are given in Table 2.

Table 1: Summary of Computations

The following computations times refer to our Maple implementation of the algorithm on a Sun Ultrasparc I workstation.

p	$p-1$	comp. time (in sec.)	size of term (tree rep.)		size of term (dag rep.)	
			rational operations	radical operations	rational operations	radical operations
7	$2 \cdot 3$	1	95	24	28	5
11	$2 \cdot 5$	2	569	78	72	5
13	$2^2 \cdot 3$	2	291	74	52	9
17	2^4	2	55	38	34	12
19	$2 \cdot 3^2$	5	1127	264	91	9
23	$2 \cdot 11$	34	7941	442	323	5
29	$2^2 \cdot 7$	18	5163	498	212	9
31	$2 \cdot 3 \cdot 5$	18	8337	1138	210	10
37	$2^2 \cdot 3^2$	20	3567	822	153	15
41	$2^3 \cdot 5$	26	6937	958	259	19
43	$2 \cdot 3 \cdot 7$	49	28110	2688	380	10
47	$2 \cdot 23$	856	70126	2026	1218	5
53	$2^2 \cdot 13$	151	39309	1878	593	9
59	$2 \cdot 29$	2524	140127	3250	1863	5
61	$2^2 \cdot 3 \cdot 5$	89	27144	3730	317	20
67	$2 \cdot 3 \cdot 11$	206	120397	7052	729	10
71	$2 \cdot 5 \cdot 7$	237	87960	8374	595	10
73	$2^3 \cdot 3^2$	114	17788	4052	305	25
79	$2 \cdot 3 \cdot 13$	362	201333	10002	947	10
83	$2 \cdot 41$	13962	407621	6562	3682	5
89	$2^3 \cdot 11$	342	113155	6608	920	19
97	$2^5 \cdot 3$	216	10015	2566	256	37
101	$2^2 \cdot 5^2$	388	68853	9194	643	19

Table 2: Some results with *SwapPrimeOrder := true*

All other settings were equal to the ones used for Table 1.

p	$p-1$	comp. time (in sec.)	size of term (tree rep.)		size of term (dag rep.)	
			rational operations	radical operations	rational operations	radical operations
23	$2 \cdot 11$	94	10282	649	662	5
43	$2 \cdot 3 \cdot 7$	194	12432	2495	554	10
71	$2 \cdot 5 \cdot 7$	807	90413	13219	980	10

Comparison to Related Work

In the book of Gaal [4] a radical expression for a 7th root of unity is derived by some special reasoning that does not generalize to higher orders. The derived expression is the following:

```
> restart: read `radsolvelib`:
> A:=-1/6+(7/2+21/2*sqrt(-3))^(1/3)/6
> +(7/2-21/2*sqrt(-3))^(1/3)/6
> +(sqrt((-1/3+(7/2+21/2*sqrt(-3))^(1/3)/3
> +(7/2-21/2*sqrt(-3))^(1/3)/3)^2-4))/2;
```

$$
\begin{aligned}
A := & -\frac{1}{6}+\frac{1}{6}\,(\frac{7}{2}+\frac{21}{2}\,I\,\sqrt{3})^{1/3} \\
& +\frac{1}{6}\,(\frac{7}{2}-\frac{21}{2}\,I\,\sqrt{3})^{1/3}+\frac{1}{2}(\\
& (-\frac{1}{3}+\frac{1}{3}\,(\frac{7}{2}+\frac{21}{2}\,I\,\sqrt{3})^{1/3}+\frac{1}{3}\,(\frac{7}{2}-\frac{21}{2}\,I\,\sqrt{3})^{1/3})^2 \\
& -4)^{1/2}
\end{aligned}
$$

Our function *radsolve* computes:

```
> B:=radsolve(x^6+x^5+x^4+x^3+x^2+x+1);
```

$$
\begin{aligned}
B := & -\frac{1}{2}\left(-\frac{7}{3}+\frac{1}{3}\,\%2^{1/3}\,(-\frac{1}{2}+\frac{1}{2}\,\sqrt{-3})^4\right. \\
& \left.+\frac{1}{3}\,\frac{(6-\sqrt{-3}-2\,\%1)\,(-\frac{1}{2}+\frac{1}{2}\,\sqrt{-3})^2}{\%2^{1/3}}\right)^{1/2}-\frac{1}{6} \\
& +\frac{1}{6}\,\%2^{1/3}+\frac{1}{6}\,\frac{6-\sqrt{-3}-2\,\%1}{\%2^{1/3}} \\
& \%1 := (-\frac{1}{2}+\frac{1}{2}\,\sqrt{-3})^2 \\
& \%2 := 9\,\%1+8-6\,\sqrt{-3}
\end{aligned}
$$

The *radsolve*-function computed the 7th root of unity that is equal to $e^{(\frac{2\,\pi\,I\,6}{7})}$:

```
> radnormal(A-B^6);
```

$$0$$

Although expression A was obtained by a special method, it is not simpler than the one derived by our general algorithm, as the *cost* function shows:

```
> readlib(cost): readlib(optimize):
> cost(optimize(A));
```

$$
\begin{aligned}
& 8\,\mathit{additions}+16\,\mathit{multiplications} \\
& \quad +12\,\mathit{functions}+6\,\mathit{assignments}
\end{aligned}
$$

```
> cost(optimize(B));
```

$$
\begin{aligned}
& 10\,\mathit{additions}+16\,\mathit{multiplications}+\mathit{divisions} \\
& \quad +6\,\mathit{functions}+7\,\mathit{assignments}
\end{aligned}
$$

Except for an implementation by ourselves of a much more inefficient method for the same task described in [5], we do not know of implementations of other general methods by which radical expressions for higher roots of unity can be computed. Without being aware of an implementation, we know of an algorithm developed by B. Trager, which computes radical expressions for a p-th root of unity [6]. This algorithm is entirely different from the one of Gauss. The major computational task consists of inverting a matrix of size $\mathrm{O}(p)$ over $\mathrm{Q}(\zeta_q)$, where q is a divisor of $p-1$. Thus if $p-1$ is smooth, i.e. if $p-1$ contains only small prime factors, the asymptotic time complexity of our improvement of the algorithm of Gauss is much better. But in special cases, such that $\frac{p-1}{2}$ is prime, the algorithm of Trager might be an interesting alternative. It would be interesting to have a careful implementation of the hard cases, such as $p = 47,\ 59,\ 83$ etc.

Acknowledgment

A. Weber was partially supported by *Deutsche Forschungsgemeinschaft* under grants We 1945/1-1 and Ku 966/6-1. We are grateful to M. Monagan and an anonymous referee for many detailed comments and suggestions.

References

[1] Carl Friedrich Gauss. *Disquisitiones Arithmeticae*. G. Fleischer Jun., Göttingen, 1801. In Latin. Reprinted in [7]. German translation: [8]; English translation: [9].

[2] Andreas Weber. Computing radical expressions for roots of unity. *SIGSAM Bulletin*, 30(117):11–20, September 1996.

[3] W. Narkiewicz. *Elementary and Analytic Theory of Numbers*. Springer-Verlag, 1990.

[4] Lisl Gaal. *Classical Galois Theory*. Chelsea Publishing Company, New York, fourth edition, 1988.

[5] Harold M. Edwards. *Galois Theory*, volume 101 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1984.

[6] Richard Zippel. Computer algebra. Unpublished Lecture Notes, 1994.

[7] Carl Friedrich Gauss. *Werke*, volume I. Georg Olms Verlag, Hildesheim, New York, 1981.

[8] Carl Friedrich Gauss. *Untersuchungen über höhere Arithmetik*. Chelsea Publishing Company, second edition, 1981.

[9] Carl Friedrich Gauss. *Disquisitiones Arithmeticae — English Edition*. Springer-Verlag, 1986.