Computing Polynomial Greatest Common Divisors over Algebraic Number and Function Fields

Mahdi Javadi and Michael Monagan, Simon Fraser University

Abstract

We report on a new GCD algorithm that has been developed by the computer algebra group at Simon Fraser University and which has been implemented and integrated into the development version of Maple. We present some benchmarks which illustrate the improvements and include some comments on the implications of this work.

Introduction

Below are two polynomials. They are typical of what might be encountered by an engineer or scientist in a computation.

$$\begin{aligned} f &= x^2 + sy^2 + xy\sqrt{1-s} - 3\sqrt{2}\,x + s^2, \\ g &= 3x^2 + y^2 + \sqrt{2}\,xy + sy\sqrt{1-s} + s. \end{aligned}$$

One could easily multiply f and g by hand to get their product h. With some care, you could also divide the product h by f to get back g. In a computer algebra system, like Maple, an important operation is to simplify the fraction

$$\frac{f}{g} = \frac{x^2 + sy^2 + xy\sqrt{1-s} - 3\sqrt{2}\,x + s^2}{3x^2 + y^2 + \sqrt{2}\,xy + sy\sqrt{1-s} + s}.$$

Here we need to compute and remove the greatest common factor between the numerator f and denominator g, that is, the GCD of f and g. This is much more difficult to do by hand. The polynomials f and g are both *quadratic* in two *variables* x and y. They involve two *field extensions*, in this case both simple square-roots. The first extension, $\sqrt{2}$, is an *algebraic number* while the second, $\sqrt{1-s}$, is an *algebraic function* in one *parameter* s. In general, we may have polynomials of degree $d > 0$ in $n > 0$ variables with coefficients involving $m \geq 0$ field extensions in $k \geq 0$ parameters. Here is a second example.

$$\begin{gathered} g = 2tx^2 - 3sy^2 - \alpha s^3 xy + \alpha^2 sx + 5\sqrt{2}\,t^3, \\ a = x^2 + y^2 + \sqrt{2}\,\alpha xy + t^3 x + 3x + \alpha^2 s, \quad \text{and} \\ b = sx^2 - ty^2 + sxy + \alpha^2\sqrt{2}y - s^2\sqrt{2}x + 3t^2, \\ \text{where} \quad \alpha^3 + t\alpha^2 + s = 0. \end{gathered}$$

In this example, g, a and b are quadratic ($d = 2$) polynomials in two ($n = 2$) variables x and y in two ($k = 2$) field extensions $\sqrt{2}$ and α. This time α is a root of a cubic polynomial in $m = 2$ parameters s and t.

Suppose we form the two products : $A = g \times a$ and $B = g \times b$ so that g is a common factor of A and B. How difficult is it to compute the greatest common factor of A and B (the answer is g)? It turns out that the Euclidean algorithm, and variations of the Euclidean algorithm which are based on polynomial division in one variable, "blow up" with the degree and number of variables and parameters. The blow up can be very bad, even for small problems, which makes computing with polynomials involving field extensions "hopeless".

A new GCD algorithm for polynomials with algebraic extensions.

In 2004, as part of our MITACS project, Monagan and van Hoeij [2] designed and implemented a first modular GCD algorithm for polynomials with one field extension. Later in 2007 Monagan and Javadi [4] extended this algorithm to treat multiple field extensions. At the same time the algorithm was redesigned to use a sparse interpolation to in order to improve performance for problems with many variables and/or many parameters. The sparse interpolation that was used, a modification of Zippel's algorithm, came from de Kleine, Monagan and Wittkopf in [3].

Here we report on the completed implementation of the general algorithm and its incorporation into Maple. Thus we now have an effective algorithm that works for all cases, polynomials in $n > 0$ variables, $k \geq 0$ field extensions in $m \geq 0$ parameters.

Below we present three benchmarks. The first is for the GCD problem described above where the GCD g and cofactors a and b are quadratic in two variables x and y. In the second benchmark the GCD g and the cofactors a and b are cubic rather than quadratic. In the third benchmark we have added a third polynomial variable z so that the GCD g and the cofactors a and b are quadratic in three variables. The Maple commands for creating the benchmarks are included in the Appendix. We remark that even for these small degrees and numbers of parameters, we soon get to the edge of what was possible using the old GCD algorithm in Maple.

We report the CPU time to compute the GCD (in seconds) and the space used in megabytes (MB) which counts the maximum data used during the test run. Here d is the degree of the gcd g and the cofactors a and b. The benchmarks were run in Maple 12 on an AMD Opteron CPU running at 2.2 GHz. The improvements mean that we can now compute effectively with polynomials in many variables which involve algebraic numbers and functions.

Benchmark	Existing code	New code
1 ($d = 2, n = 2, k = 2, m = 2$)	887.5s (272 MB)	0.180s (6 MB)
2 ($d = 3, n = 2, k = 2, m = 2$)	>10,000s (>942 MB)	0.195s (7 MB)
3 ($d = 2, n = 3, k = 2, m = 2$)	>10,000s (>1872 MB)	0.321s (8 MB)

Table 1: Timings (in CPU seconds) for computing the GCD of A and B

Factoring polynomials with algebraic extensions.

The improvements to the GCD algorithm also improve Maple's algorithm for factoring polynomials having algebraic extensions. The algorithm Maple uses for this is known as the Trager-Kronecker algorithm (see [1, Chapter 8] for a description of this algorithm). It reduces factorization over an algebraic number or function field to polynomial factorization over the the rationals – a problem which is solvable in polynomial time. The final step of this algorithm, and one which is computationally intensive, requires a polynomial GCD computation. This is where the new GCD algorithm results in considerable improvement.

Below we report timings for factoring the three A polynomials in the benchmarks. Here d is the degree of A in the variables. The number in the second column is the number of terms in the polynomial that needs to be factored over the rationals.

Benchmark	#terms	Existing code	With new GCD code
1 $(d = 4, n = 2, k = 2, m = 2)$	27,367	>13,000s (1326 MB)	120.7s (55 MB)
2 $(d = 6, n = 2, k = 2, m = 2)$	37,033	Not attempted	571.5s (91 MB)
3 $(d = 4, n = 3, k = 2, m = 2)$	298,439	Not attempted	5,953.7s (592 MB)

Table 2: Benchmarks for factoring the polynomial A.

In these benchmarks, where our new algorithm is being used, most of the time is spent in factorization over the rationals. Very little (under 1%) of the time is spent doing GCD computation. Obviously, $5,953$ seconds is still a very long time and there is room for improvement here. One possible approach that we are considering is to evaluate all parameters at small integers and also all polynomial variables except one, factor the univariate image, and develop a sparse Hensel lifting to recover the variables and parameters in the result.

Acknowledgment

We gratefully acknowledge funding from MITACS and MapleSoft. Also the help from Jürgen Gerhard, Bryan Kravetz and Allan Wittkopf at our industrial partner, Maplesoft.

References

[1] K.O. Geddes, S.R. Czapor and G. Labahn. *Algorithms for Computer Algebra*, Kluwer Publishing, 507 pgs, 1992.

[2] M. van Hoeij and M. B. Monagan. Algorithms for Polynomial GCD Computation over Algebraic Function Fields. *Proceedings of ISSAC '2004*, ACM Press, pp. 297–304, 2004.

[3] J. de Kleine, M. Monagan and A. Wittkopf. Algorithms for the Non-monic case of the Sparse Modular GCD Algorithm. *Proceedings of ISSAC '2005*, ACM Press, pp. 124–131, 2005.

[4] M. Javadi and M. Monagan. A Sparse Modular GCD Algorithm for Polynomial GCD Computation over Algebraic Function Fields. *Proceedings of ISSAC '07*, ACM Press, pp. 187–194, July 2007.

Appendix

Maple input for the benchmarks.

Benchmark 1

```
> alias( alpha=RootOf(z^3+t*z^2+s,z) );
> alias( beta=RootOf(z^2-2) );
> g := 2*t*x^2-3*s*y^2-alpha*s^3*x*y + alpha^2*s*x + 5*beta*t^3;
> a := x^2+y^2+alpha*beta*x*y+t^3*x+3*x+alpha^2*s;
> b := s*x^2-t*y^2+s*x*y+alpha^2*beta*y-s^2*beta*x+3*t^2;
> A := evala(Expand(g*a)):
> B := evala(Expand(g*b)):
> st := time(): G := evala(Gcd(A,B)); time()-st;
> st := time(): evala(Factor(A)); time()-st;
```

Benchmark 2

```
> alias( alpha=RootOf(z^3+t*z^2+s*t,z) );
> alias( beta=RootOf(z^2-2) );
> g := 2*t*x^3-3*s*y^3-alpha*s^2*t*x*y^2 + alpha^2*s*x + 5*beta*t^3;
> a := x^3+y^3+alpha*beta*x*y^2+t^3*x^2+x+alpha^2*s;
> b := s*x^3-t*y^3+s*x*y+alpha^2*beta*y^2-s*beta*x+3*t^2;
> A := evala(Expand(g*a)):
> B := evala(Expand(g*b)):
> st := time(): G := evala(Gcd(A,B)); time()-st;
> st := time(): evala(Factor(A)); time()-st;
```

Benchmark 3

```
> alias( alpha=RootOf(z^3+t*z^2+s,z) );
> alias( beta=RootOf(z^2-2) );
> g := 2*t*x^2 - 3*s*y^2 - alpha*s^3*x*y + alpha^2*s*x + 5*beta*t^3
>        + s*t*z^2 - 2*z*y*alpha*beta;
> a := x^2 + y^2 + alpha*beta*x*y + t^3*x + 3*x + alpha^2*s
>        + z^2 + z*x*t + 3*s*z*alpha;
> b := s*x^2 - t*y^2 + s*x*y + alpha^2*beta*y - s^2*beta*x + 3*t^2
>        + alpha*z^2*t + 2*z*y*s^3 + z*beta;
> A := evala(Expand(g*a)):
> B := evala(Expand(g*b)):
> st := time(): G := evala(Gcd(A,B)); time()-st;
> st := time(): evala(Factor(A)); time()-st;
```