Using fast power-series arithmetic in the Kedlaya-Denef-Vercauteren algorithm

Daniel J. Bernstein *

djb@cr.yp.to

The problem. Let $K$ be a field of characteristic 0. Fix a positive integer $g$. We’re given $f, h, Q \in K[x]$ where $f$ is monic, $\deg f = 2g + 1$, and $\deg h \leq g$. How do we compute $P, R \in K[x]$ with $\deg R < 2g$ and $Q - R = (2f' + hh')P + (1/3)(4f + h^2)P'$? One can take $(P, R) = (0, Q)$ if $\deg Q < 2g$, so assume that $\deg Q \geq 2g$.

Tiny example: Define $K = \mathbb{C}$, $g = 2$, $f = x^5 + x^4 + 1$, $h = x$, and $Q = x^7 + 11x^5 + x + 1$. How do we compute $P, R \in \mathbb{C}[x]$ with

$$x^7 + 11x^5 + x + 1 - R = (10x^4 + 8x^3 + x)P + (1/3)(4x^5 + 4x^4 + x^2 + 4)P'$$

and $\deg R < 4$?

Application. Kedlaya introduced an algorithm for computing the zeta function of a genus-$g$ hyperelliptic curve over a finite field of size $p^n$ when $p$ is odd. Kedlaya’s algorithm uses roughly $g^4n^3$ bit operations for fixed $p$.

Denef and Vercauteren adapted Kedlaya’s algorithm to the case $p = 2$. The Kedlaya-Denef-Vercauteren algorithm uses roughly $g^4n^3$ bit operations for a “typical” curve but roughly $g^5n^3$ bit operations for some other curves.

At a meeting in Oberwolfach I asked Kedlaya about the discrepancy between $g^4n^3$ and $g^5n^3$. He explained the problem of computing $P, R$ from $f, h, Q$ and told me that this was one of the bottlenecks in the $p = 2$ case.

A slow solution. Apparently Denef and Vercauteren use the equation $Q - R = (2f' + hh')P + (1/3)(4f + h^2)P'$ to determine the coefficients of $P$ one at a time. The algebraic complexity of this computation over $K$—the number of additions, subtractions, multiplications, and divisions of coefficients in $K$—grows quadratically with $g$ in the typical case $\deg Q = 4g$.

Tiny example: Consider again the problem of finding $P, R \in \mathbb{C}[x]$ with $x^7 + 11x^5 + x + 1 - R = (10x^4 + 8x^3 + x)P + (1/3)(4x^5 + 4x^4 + x^2 + 4)P'$ and $\deg R < 4$. Assume that $P$ will have degree at most 3; write $P$ as $P_3x^3 + P_2x^2 + P_1x + P_0$; write $R$ as $R_3x^3 + R_2x^2 + R_1x + R_0$. The problem is now to find $P_3, P_2, P_1, P_0, R_3, R_2, R_1, R_0$ such that

$$x^7 + 11x^5 + x + 1 - (R_3x^3 + R_2x^2 + R_1x + R_0)$$

$$= (10x^4 + 8x^3 + x)(P_3x^3 + P_2x^2 + P_1x + P_0)$$

$$+ (1/3)(4x^5 + 4x^4 + x^2 + 4)(3P_3x^2 + 2P_2x + P_1).$$

* Permanent ID of this document: 4e30a3e7f413533744a20c9c48e7025f. Date of this document: 2006.10.19.
Extract the coefficients of \(x^7, x^6, x^5, x^4, x^3, x^2, x^1, x^0\) from this equation to form a lower-triangular system of linear equations:

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & P_3 \\
0 & 12/3 & 0 & 0 & 0 & 0 & P_2 \\
11 & 0 & 32/3 & 34/3 & 0 & 0 & 0 & P_1 \\
0 & 1 & 0 & 28/3 & 10 & 0 & 0 & P_0 \\
0 & 2 & 2/3 & 0 & 8 & 1 & 0 & R_3 \\
0 & 4 & 2 & 1/3 & 0 & 0 & 1 & R_2 \\
1 & 0 & 8/3 & 2 & 0 & 0 & 1 & R_1 \\
1 & 0 & 0 & 4/3 & 2 & 0 & 0 & 1 & R_0
\end{pmatrix}
\]

Use substitution to solve this system one variable at a time: use the first equation \(1 = 14P_3\) to determine \(P_3 = 1/14\), then use the second equation \(0 = 12P_3 + (38/3)P_2\) to determine \(P_2 = -9/133\), etc.

**A faster solution.** The following solution produces the same output but is much more efficient than the one-at-a-time solution when \(g\) and \(\deg Q - 2g\) are large. This solution relies on standard FFT-based subroutines for fast powerseries multiplication, division, and square root. The higher-level aspects of the solution are also standard, so I’d be embarrassed to receive any credit for the solution; my interests here are purely expository, advertising yet another reason that novices should learn how to use fast multiplication. Anyway, here’s the solution:

- Compute \((4f + h^2)^{1/2} = (2)x^{g+1/2} + (\cdots)x^{g-1/2} + (\cdots)x^{g-3/2} + \cdots\) to high precision in the field \(K((1/\sqrt{x}))\).
- Multiply by \(3Q\), producing \(3Q(4f + h^2)^{1/2}\) to high precision in \(K((1/\sqrt{x}))\).
- Integrate with respect to \(x\), producing \(\int 3Q(4f + h^2)^{1/2}dx\) to high precision in \(K((1/\sqrt{x}))\).
- Divide by \((4f + h^2)^{3/2} = (8)x^{3g+3/2} + (\cdots)x^{3g+1/2} + (\cdots)x^{3g-1/2} + \cdots\), producing \((4f + h^2)^{-3/2}\int 3Q(4f + h^2)^{1/2}dx\) to high precision in \(K((1/\sqrt{x}))\).
- Round to a polynomial \(P \in K[x]\).
- Compute \(R = Q - (2f' + hh')P - (1/3)(4f + h^2)P'\) in \(K[x]\).

Why does this work? Answer: Write \(\epsilon = P-(4f + h^2)^{-3/2}\int 3Q(4f + h^2)^{1/2}dx\). Multiply by \((4f + h^2)^{3/2}\), differentiate, and divide by \(3(4f + h^2)^{1/2}\) to see that \(R = (2f' + hh')\epsilon + (1/3)(4f + h^2)\epsilon'. By construction \(\epsilon = (\cdots)x^{-1} + (\cdots)x^{-2} + \cdots\) so \(R = (2(2g+1)x^{2g} + \cdots)((\cdots)x^{-1} + \cdots) + (1/3)(4x^{2g+1} + \cdots)((\cdots)x^{-2} + \cdots) = (\cdots)x^{2g-1} + \cdots\); i.e., \(\deg R < 2g\) as desired.

I omitted one important detail above: What does “high precision” mean? Answer: We compute the first \(\deg Q - 2g + 1\) coefficients of each series; this is enough information to determine \(P \in K[x]\). This means that we compute

- the coefficients of \(x^{g+1/2}, x^{g-1/2}, \ldots, x^{3g - \deg Q + 1/2}\) in \((4f + h^2)^{1/2}\);
- the coefficients of \(x^{\deg Q + g + 1/2}, x^{\deg Q + g - 1/2}, \ldots, x^{3g + 1/2}\) in \(3Q(4f + h^2)^{1/2}\);
- the coefficients of \(x^{\deg Q + 2g + 3/2}, \ldots, x^{3g + 3/2}\) in \(\int 3Q(4f + h^2)^{1/2}dx\); and
- the coefficients of \(x^{\deg Q - 2g}, \ldots, x^0\) in \((4f + h^2)^{-3/2}\int 3Q(4f + h^2)^{1/2}dx\).
Rounding to \( P \in K[x] \) means simply copying the coefficients of \( x^{\deg Q-2g}, \ldots, x^0 \).

This computation has algebraic complexity essentially linear in \( g \), rather than quadratic in \( g \), in the typical case \( \deg Q = 4g \). More precisely, this computation has algebraic complexity \( O(g \lg g \lg \lg g) \), with the \( \lg \lg g \) disappearing for some choices of \( K \). The complexity here is within a constant factor of the complexity of multiplication, division, and square root; I haven’t analyzed or optimized the choices of \( h \).

Now round to \( P, R \) compute \( \zeta \) and \( \nu \) compute \( h \) and \( g \).

**Consider the cost of computing the zeta function** of a genus-\( g \) hyperelliptic curve \( y^2 + h(x)y = f(x) \) over a field of size \( 2^n \). “Cost” here refers to bit operations.

The Denef-Vercauteren “Theorem 1” reports cost \( \sim O((g^\lambda + g^\nu)g^{4+\epsilon}n^{3+\epsilon}) \).

As a mathematician I feel compelled to point out that the order of quantifiers here is horribly unclear. Do the authors mean “for each \( \epsilon > 0 \) there exists \( n_0 \) such that for each \( n \geq n_0 \) there exist \( g_0, \epsilon \) such that for each \( g \geq g_0 \) the cost is at most \( c(g^\lambda + g^\nu)g^{4+\epsilon}n^{3+\epsilon} \)? Do they mean “for each \( \epsilon > 0 \) there exist \( \epsilon, d_0 \) such that for each \( n, g \) with \( ng \geq d_0 \) the cost is at most \( c(g^\lambda + g^\nu)g^{4+\epsilon}n^{3+\epsilon} \)? There are many other possibilities. How is a reader supposed to apply this “theorem” without redoing the analysis?

Anyway, the Denef-Vercauteren parameters \( \lambda \) and \( \nu \) refer to the size and ramification of the polynomial \( h \) in the curve \( y^2 + h(x)y = f(x) \). Specifically, \( g^\lambda \) is (modulo further \( O \) confusion) shorthand for \( \deg f - 2 \deg h \), and \( g^\nu \) is shorthand for the maximum exponent in the factorization of \( h \).

For a uniform random curve, usually \( \deg h = g \), and usually \( h \) has very few repeated factors, so \( g^\lambda + g^\nu \) is close to \( 1 \). On the other hand, I can imagine users selecting curves where \( g^\lambda \) is much larger. Consider, for example, the Lange-Stevens hyperelliptic-curve addition formulas; one reason that these formulas are
so fast is that they force $h$ to have small degree. Perhaps users are also interested in curves where $g^\nu$ is large.

Evidently there are two different ways that the Denef-Vercauteren cost can grow more quickly than $g^{4+o(1)}n^{3+o(1)}$:

- $g^\lambda = \deg f - 2\deg h$ can grow more quickly than $g^{o(1)}$; e.g., $\deg h$ could be around $g - \sqrt{g}$, or around $g/2$. My impression is that the problem here is exactly the problem I’ve addressed, and that the one-at-a-time solution is the Denef-Vercauteren bottleneck; I speculate that the fast-arithmetic solution eliminates this bottleneck.

- $g^\nu$, the maximum exponent in the factorization of $h$, can grow more quickly than $g^{o(1)}$; for example, $h(x)$ could be $x^{g/2}(x - 1)(x - 2)\cdots(x - g/2)$. My impression is that this is a completely different problem, caused by Denef and Vercauteren working modulo, e.g., $(x(x - 1)\cdots(x - g/2))^{g/2}$. Without looking more closely at the computation—which I’m certainly not planning to do any time soon—I can’t guess whether such a large modulus is really necessary.

Bottom line: I speculate that fast power-series arithmetic expands the set of “$g^4n^3$ curves” to allow small $h$ degrees. I have no idea whether the set can be further expanded to allow large powers in $h$. 