Fast reversion of power series

Fredrik Johansson

November 2011

Overview

- Fast power series arithmetic
- Fast composition and reversion (Brent and Kung, 1978)
- A new algorithm for reversion
- Implementation results

Operations on power series

Let R be a commutative ring (with 1). We consider arithmetic in the ring of truncated formal power series $P = R[[x]]/\langle x^n \rangle$.

$$a = a(x) = \sum_{k=0}^{n-1} a_k x^k,\ a_k = [x^k]a(x) \in R.$$

Addition	$h(x) = f(x) + g(x) = \sum_{k=0}^{n-1}(f_k + g_k)x^k$
Multiplication	$h(x) = f(x)g(x) = \sum_{k=0}^{n-1}\left(\sum_{i=0}^{k} f_i g_{k-i}\right)x^k$
Reciprocal	If f_0 is a unit, $h(x)$ such that $f(x)h(x) = 1$
Composition	If $g_0 = 0$, $h(x) = f(g(x)) = \sum_{k=0}^{n-1} f_k(g(x))^k$
Reversion	If $f_0 = 0$ and f_1 is a unit, $h(x)$ s.t. $f(h(x)) = x$

Computational complexity

We measure complexity (as a function of n) for an operation in P by counting the number of ring operations in R.

$M(n)$: multiplying two length-n polynomials

Classical	$M(n) = O(n^2)$
Karatsuba	$M(n) = O(n^{\log_2(3)}) = O(n^{1.59})$
Fast Fourier Transform	$M(n) = O(n \log^{1+o(1)} n) = O(n^{1+o(1)} n)$

$MM(n) = O(n^\omega)$: multiplying two $n \times n$ matrices

Classical	$MM(n) = O(n^3)$
Strassen	$MM(n) = O(n^{\log_2(7)}) = O(n^{2.81})$
Coppersmith-Winograd	$MM(n) = O(n^{2.38})$

Newton iteration

Let $g_k \in P$ and $\varphi \in P[t]$ be such that $\varphi'(g_k)$ is invertible and $\varphi(g_k) \equiv 0 \bmod x^m$. If $g_{k+1} \in P$ is a solution of

$$g_{k+1} \equiv g_k - \frac{\varphi(g_k)}{\varphi'(g_k)} \bmod x^{2m},$$

then $\varphi(g_{k+1}) \equiv 0 \bmod x^{2m}$, $g_{k+1} \equiv g_k \bmod x^m$, and $\varphi'(g_{k+1})$ is invertible.

$O(M(n))$ computation of $1/f$: take $\varphi(g) = \frac{1}{g} - f$, giving the iteration $g_{k+1} = 2g_k - fg_k^2$.

Reversion using Newton iteration

Let $f \in P$ with $[x^0]f = 0$ and $[x^1]f$ invertible in R.

Define φ by $\varphi(g(x)) = f(g(x)) - x$. Then $\varphi(f^{-1}(x)) = 0$, so the compositional inverse (reversion) f^{-1} is a root of φ.

Quadratically convergent iteration:

$$g_{k+1} = g_k - \frac{f(g_k) - x}{f'(g_k)}.$$

Up to constant factors, composition $\Leftrightarrow$ reversion (Brent and Kung, 1978).

Fast composition

Can we reduce composition to multiplication?

Composition by elementary functions

Assume that $1, 2, \ldots, n-1$ are invertible in R.

$$f'(x) = \sum_{k=0}^{n-2} (k+1) f_{k+1} x^k \qquad \int f(x) = \sum_{k=1}^{n-1} \left(\frac{1}{k}\right) f_{k-1} x^k$$

$$\log(1+f(x)) = \int \frac{f'(x)}{1+f(x)} \qquad \operatorname{atan}(f(x)) = \int \frac{f'(x)}{1+(f(x))^2}$$

Newton iteration: log $\to$ exp, atan $\to$ tan.
sin, cos, sinh, cosh, asin ... using algebraic transformations.

Cost: $O(M(n)) = O(n^{1+o(1)})$.

Generalization: differential equations (hypergeometric, ...)

Composition of general power series

Horner's rule

$$f(g(x)) = f_0 + g(f_1 + g(f_2 + \cdots + g(f_{n-2} + f_{n-1}x)\cdots))$$

Complexity: $O(nM(n))$

Brent-Kung 2.1: baby-step, giant-step version of Horner's rule
Complexity: $O(n^{1/2}M(n) + n^{1/2}MM(n^{1/2}))$

Brent-Kung 2.2: divide-and-conquer Taylor expansion
Complexity: $O((n \log n)^{1/2}M(n))$

Brent-Kung algorithm 2.1

Example with $n = 9$, $m = \lceil \sqrt{n} \rceil = 3$
Computing $f(g(x))$ where $f(x) = f_0 + f_1 x + \ldots + f_8 x^8$

$$(f_0 + f_1 g + f_2 g^2) + (f_3 + f_4 g + f_5 g^2) g^3 + (f_6 + f_7 g + f_8 g^2) g^6$$

$(m \times m) \times (m \times m^2)$ matrix multiplication:

$$\begin{pmatrix} f_0 & f_1 & f_2 \\ f_3 & f_4 & f_5 \\ f_6 & f_7 & f_8 \end{pmatrix} \times \begin{pmatrix} 1 \\ g \\ g^2 \end{pmatrix}$$

Final combination using "block" Horner's rule.

Complexity of composition algorithms

Horner's rule: $O(nM(n))$

BK 2.1: $O(n^{1/2}M(n) + n^{1/2}MM(n^{1/2}))$

BK 2.2: $O((n \log n)^{1/2}M(n)) = O(n^{3/2+o(1)})$

$M(n)$	$MM(n)$	Horner	BK 2.1	BK 2.2
n^2	n^3	n^3	$n^{2.5}$	$n^{2.5} \log^{0.5} n$
$n^{1.59}$	n^3	$n^{2.59}$	$n^{2.09}$	$n^{2.09} \log^{0.5} n$
$n \log n$	n^3	$n^2 \log n$	n^2	$n^{1.5} \log^{1.5} n$
$n \log n$	$n^{2.38}$	$n^2 \log n$	$n^{1.69}$	$n^{1.5} \log^{1.5} n$
$(n \log n)$	(n^2)	$(n^2 \log n)$	$(n^{1.5} \log n)$	$(n^{1.5} \log^{1.5} n)$

BK 2.1 wins for small n, BK 2.2 wins for large n

BK 2.1 wins with hypothetical $O(n^2)$ matrix multiplication

The Lagrange inversion theorem

If $f(x) = x/h(x)$ then the compositional inverse or reversion $f^{-1}(x)$ satisfying $f(f^{-1}(x)) = f^{-1}(f(x)) = x$ exists and its coefficients are given by

$$[x^k]f^{-1}(x) = \frac{1}{k}[x^{k-1}]h(x)^k.$$

$O(nM(n))$ algorithm: compute $x/h(x)$ using Newton iteration for the reciprocal, then compute $h, h^2, h^3, \ldots$ using successive multiplications

Better than classical algorithms ($O(n^3)$)

Equivalent to Newton iteration + Horner's rule

Worse than Newton iteration + BK2.1 or BK2.2

Fast Lagrange inversion

For $0 < k < n$, write $k = im + j$. Using the definition of the Cauchy product, we can extract a single coefficient in $O(n)$ operations:

$$[x^k]h^k = \sum_{r=0}^{k} ([x^r])\, h^{im} \times \left([x^{k-r}]\right) h^j$$

We only need to compute h^j where $j = 1, 2, \ldots, m-1$ and h^{im} where $i = 1, 2, \ldots, n/m$.

Choose $m \approx \sqrt{n} \Rightarrow$ only $O(\sqrt{n})$ polynomial multiplications.

We replace $O(n)$ full polynomial multiplications with $O(n^2)$ coefficient operations.

Fast Lagrange inversion, first version

1: $m \leftarrow \lfloor \sqrt{n} \rfloor$
2: $h \leftarrow x/f \bmod x^{n-1}$
3: **for** $1 \leq i < m$ **do**
4: $\quad h^{i+1} \leftarrow h^i \times h \bmod x^{n-1}$
5: $\quad b_i \leftarrow \frac{1}{i}[x^{i-1}]h^i$
6: **end for**
7: $t \leftarrow h^m$
8: **for** $i = m, 2m, 3m, \ldots, \ell m < n$ **do**
9: $\quad b_i \leftarrow \frac{1}{i}[x^{i-1}]t$
10: $\quad$ **for** $1 \leq j < m$ while $i + j < n$ **do**
11: $\quad\quad b_{i+j} \leftarrow \frac{1}{i+j} \sum_{k=0}^{i+j-1} ([x^k]t) \cdot ([x^{i+j-k-1}]h^j)$
12: $\quad$ **end for**
13: $\quad t \leftarrow t \times h_m \bmod x^{n-1}$
14: **end for**
15: **return** $b_1 + b_2 x + \ldots + b_{n-1} x^{n-1}$

Using matrix multiplication

Take $n = 8$, $m = \lceil \sqrt{8-1} \rceil = 3$. We need the coefficients $b_1, b_2, \ldots, b_7$ of $1, x, \ldots, x^6$ in powers of h. If $h_i^k \equiv [x^i]h^k$:

$$A = \begin{pmatrix} h_2^0 & h_1^0 & h_0^0 & 0 & 0 & 0 & 0 & 0 & 0 \\ h_5^3 & h_4^3 & h_3^3 & h_2^3 & h_1^3 & h_0^3 & 0 & 0 & 0 \\ (h_8^6) & (h_7^6) & h_6^6 & h_5^6 & h_4^6 & h_3^6 & h_2^6 & h_1^6 & h_0^6 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & 0 & h_0^1 & h_1^1 & h_2^1 & h_3^1 & h_4^1 & h_5^1 & h_6^1 \\ 0 & h_0^2 & h_1^2 & h_2^2 & h_3^2 & h_4^2 & h_5^2 & h_6^2 & (h_7^2) \\ h_0^3 & h_1^3 & h_2^3 & h_3^3 & h_4^3 & h_5^3 & h_6^3 & (h_7^3) & (h_8^3) \end{pmatrix}$$

$$AB^T = \begin{pmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & - & - \end{pmatrix}$$

Example: $b_5 = h_4^3 h_0^2 + h_3^3 h_1^2 + h_2^3 h_2^2 + h_1^3 h_3^2 + h_0^3 h_4^2$

Fast Lagrange inversion, matrix version

1: $m \leftarrow \lceil \sqrt{n-1} \rceil$
2: $h \leftarrow x/f \bmod x^{n-1}$
3: {Assemble $m \times m^2$ matrices B and A from $h, h^2, \ldots, h^m$ and $h^m, h^{2m}, h^{3m}, \ldots$}
4: **for** $1 \le i \le m,\ 1 \le j \le m^2$ **do**
5: $\quad B_{i,j} \leftarrow [x^{i+j-m-1}]\ h^i$
6: $\quad A_{i,j} \leftarrow [x^{im-j}]\ h^{(i-1)m}$
7: **end for**
8: $C \leftarrow AB^T$
9: **for** $1 \le i < n$ **do**
10: $\quad b_i \leftarrow C_i/i$ (C_i is the ith entry of C read rowwise)
11: **end for**
12: **return** $b_1 + b_2 x + \ldots + b_{n-1} x^{n-1}$

Comparison with BK 2.1

Let $m = \sqrt{n}$.

Fast Lagrange inversion for reversion:

1. $2m + O(1)$ polynomial multiplications
2. One $(m \times m^2)$ times $(m^2 \times m)$ matrix multiplication
3. $O(n)$ additional operations

"Fast Horner's rule" (BK 2.1) for composition:

1. m polynomial multiplications, each with cost $M(n)$
2. One $(m \times m)$ times $(m \times m^2)$ matrix multiplication
3. m polynomial multiplications and additions

Both $O(n^{1/2}(M(n) + MM(n^{1/2})))$, same constant factor.

Speedup by avoiding Newton iteration

Suppose composition has cost $C(n) \sim cn^r$.

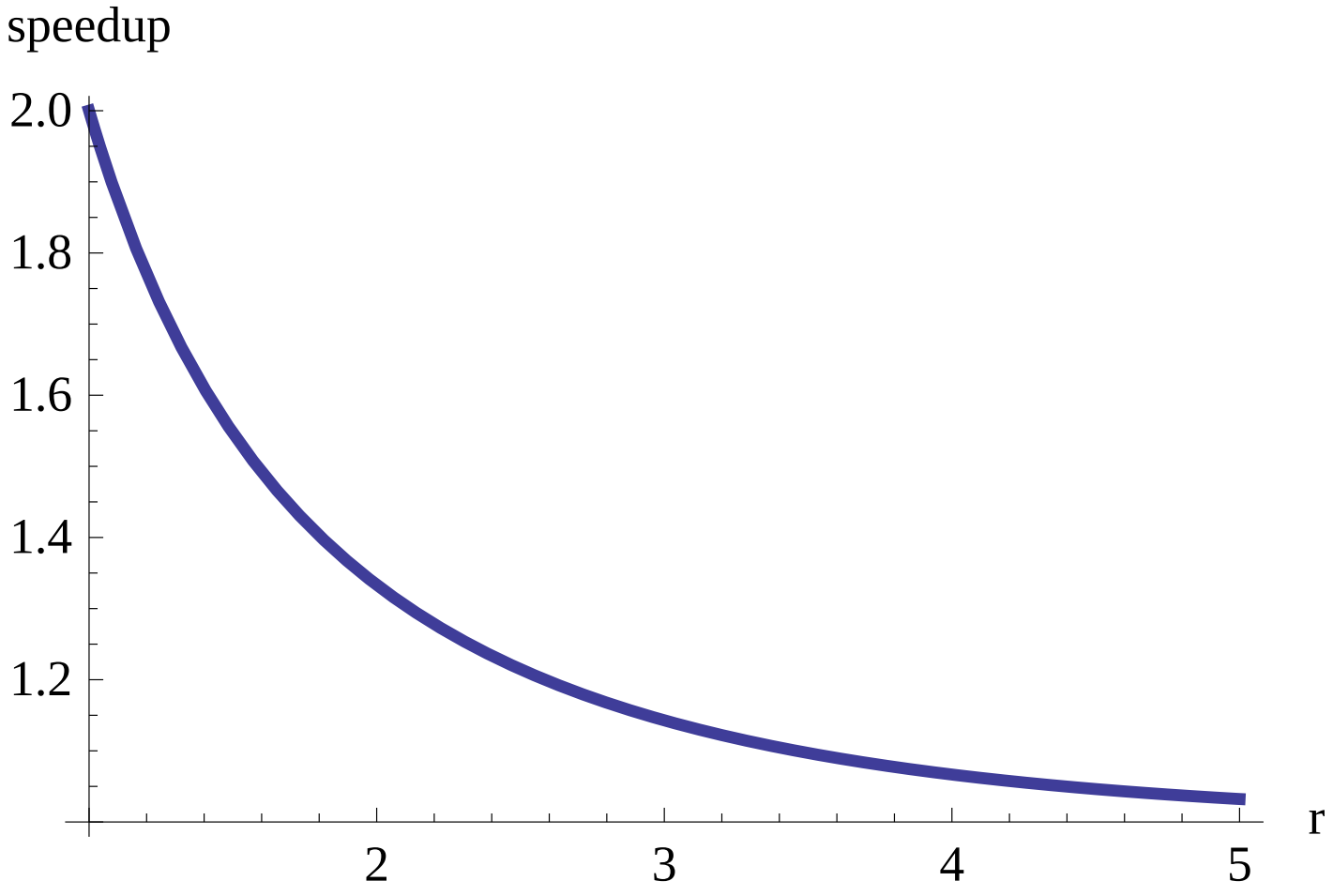
Overhead of reversion using Newton iteration:

$$C(n) + C(n/2) + C(n/4) + \ldots = cn^r \left(\frac{2^r}{2^r - 1} \right)$$

Classical polynomial multiplication: $\frac{4}{31}(8 + \sqrt{2}) \approx 1.214$

FFT polynomial multiplication, classical matrix multiplication: $4/3$

Newton iteration speedup, visually



Faster matrix multiplication

The matrices in BK 2.1 are full, whereas the matrix A in fast Lagrange inversion is half empty.

Can we exploit this structure while using fast matrix multiplication ($\omega < 3$)?

Faster matrix multiplication, first algorithm

Decompose each $m \times m$ block A_i into blocks of size $(m/k) \times (m/k)$ such that only the bottom row in is nonempty. Asymptotic speedup s:

$$s = m^{\omega+1} / \sum_{k=1}^{\infty} \left(\frac{m}{k} - \frac{m}{k+1} \right) k^2 \left(\frac{m}{k} \right)^{\omega}$$

$$s > \left(\sum_{k=0}^{\infty} \frac{2^{k-1}}{2^{k\omega}} \right)^{-1} = 2 - 2^{2-\omega} > 1$$

Faster matrix multiplication, second algorithm

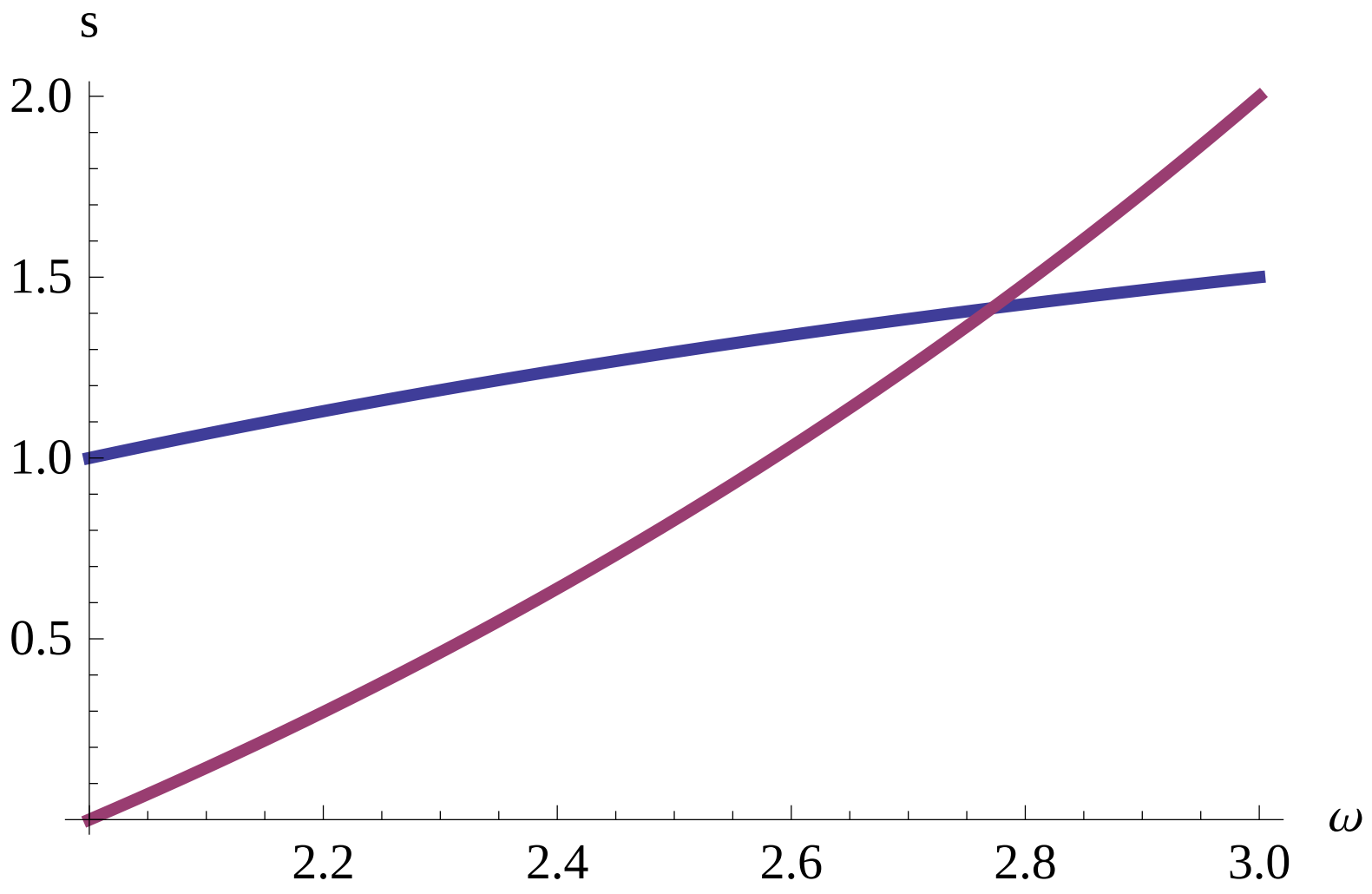
Write $AB^T = (AP)(P^{-1}B^T)$ where P is a permutation matrix that makes each $m \times m$ block in A lower triangular.

Recursive triangular multiplication:
$R(k) = 4R(k/2) + 2(k/2)^\omega + O(k^2)$, $s = 2^{\omega-1} - 2$.

$s > 1$ when $\omega > \log_2 6 \approx 2.585$, and better than the first method when $\omega > 1 + \log_2(2 + \sqrt{2}) \approx 2.771$.

Optimal factor-two speedup with classical multiplication, and 3/2 speedup with Strassen.

Matrix multiplication speedup



Fast rectangular matrix multiplication

$MM(x, y, z)$: $(x \times y)$ by $(y \times z)$ matrix multiplication

Huang and Pan (1998):
$MM(m, m, m^2) = O(n^{1.667}) < mMM(m, m, m) = O(n^{1.688})$

The exponents of $MM(m, m^2, m)$ and $MM(m, m, m^2)$ are the same, but could we lose a constant factor?

Conjecture: $MM(m, m^2, m) = (1 + o(1))MM(m, m, m^2)$

Total speedup

Theoretical speedup of fast Lagrange inversion over BK 2.1 by both avoiding Newton iteration and speeding up matrix multiplication.

Dominant operation	Complexity	Newton	Matrix	Total
Polynomial, classical	$O(n^{5/2})$	1.214	1	1.214
Polynomial, Karatsuba	$O(n^{1/2+\log_2 3})$	1.308	1	1.308
Matrix, classical	$O(n^2)$	1.333	2.000	2.666
Matrix, Strassen	$O(n^{(1+\log_2 7)/2})$	1.364	1.500	2.047
Matrix, Cop.-Win.	$O(n^{1.688})$	1.450	1.229	1.782
Matrix, Huang-Pan	$O(n^{1.667})$	1.458	1?	1.458?
(Polynomial, FFT)	$O(n^{3/2}\log^{1+o(1)} n)$	1.546	1	1.546

Implementation

Fast composition and reversion implemented in FLINT (`http://flintlib.org`)

Rings: $\mathbb{Z}$, $\mathbb{Z}/p\mathbb{Z}$, $\mathbb{Q}$

Reversion over $(\mathbb{Z}/p\mathbb{Z})[[x]]$

Input is a a power series with random coefficients, $p = 2^{63} + 29$.

n	Lagrange	BK 2.1	Fast Lagrange
10	12 µs	21 µs	7.0 µs
100	3.8 ms	1.3 ms	0.76 ms
1000	950 ms	100 ms	62 ms
10000	150 s	5.1 s	3.3 s
100000	$\sim$ 6 h	260 s	160 s

Observed speedup: 1.6

Predicted speedup with $O(n^{1+o(1)})$ multiplication and "$O(n^2)$" matrix multiplication: 1.54

Matrix multiplication negligible ($< 10\%$ of running time)

Reversion over $\mathbb{Z}[[x]]$

$$f_1(x) = \sum_{k \geq 1} k! x^k \quad f_2(x) = \frac{x}{\sqrt{1-4x}} \quad f_3(x) = \frac{x + x^2}{1 + x + x^2}$$

n	Lagrange			BK 2.1			Fast Lagrange		
	f_1	f_2	f_3	f_1	f_2	f_3	f_1	f_2	f_3
10	10 µs	10	8.4	16	15	14	6.8	5.7	5.5
50	3.7 ms	1.0	0.46	1.2	0.45	0.40	1.1	0.28	0.14
100	65 ms	10	4.4	12	2.8	2.8	13	1.4	0.87
500	23 s	3.0	1.2	2.5	0.37	0.36	2.3	0.16	0.084
1000	280 s	30	11	22	2.8	2.4	18	2.0	0.54
5000	-	-	-	4100 s	340	230	2400	110	46

Fast Lagrange inversion can lose its advantage (but usually does not get significantly slower than BK 2.1)

It can also be much faster (if $x/f(x)$ is small)

Matrix multiplication is negligible ($< 10\%$ of running time)

Reversion over $\mathbb{Q}[[x]]$

$$f_4(x) = \exp(x) - 1 \quad f_5(x) = x\exp(x) \quad f_6(x) = \frac{3x(1-x^2)}{2(1-x+x^2)^2}$$

n	Lagrange			BK 2.1			Fast Lagrange		
	f_4	f_5	f_6	f_4	f_5	f_6	f_4	f_5	f_6
10	22 µs	20	19	42	44	44	15	14	13
50	5.6 ms	5.3	1.3	2.4	3.5	2.2	1.8	1.6	0.38
100	78 ms	73	12	16	27	14	17	14	2.1
500	30 s	27	3.1	2.1	3.8	1.3	3.3	2.6	0.18
1000	340 s	300	30	17	32	8.8	30	25	1.3

Matrix multiplication is negligible ($< 20\%$ of running time), but 10 times more costly if not implemented carefully (clearing denominators)!

Conclusions

Reversion can be done using an algorithm analogous to BK 2.1, but "directly" (without Newton iteration)

Faster than classical algorithms (no overhead), faster than BK 2.1

Faster than BK 2.2 up to fairly large n (since matrix multiplication is cheap in practice)

But memory consumption can be a problem
(BK 2.2 asymptotically better)