MATHEMATICS OF COMPUTATION
Volume 80, Number 273, January 2011, Pages 387–394
S 0025-5718(2010)02392-0
Article electronically published on July 8, 2010

FASTER ALGORITHMS FOR THE SQUARE ROOT AND RECIPROCAL OF POWER SERIES

DAVID HARVEY

Abstract. We give new algorithms for the computation of square roots and reciprocals of power series in $\mathbf{C}[\![x]\!]$. If $M(n)$ denotes the cost of multiplying polynomials of degree n, the square root to order n costs $(1.333\ldots+o(1))M(n)$ and the reciprocal costs $(1.444\ldots+o(1))M(n)$. These improve on the previous best results, $(1.8333\ldots+o(1))M(n)$ and $(1.5+o(1))M(n)$, respectively.

1. Introduction

It has been known for some time that various operations on power series, such as division, reciprocal, square root, and exponentiation, may be performed in a constant multiple of the time required for a polynomial multiplication of the same length. More recently the focus has been on improving the constants. A wealth of historical and bibliographical information tracking the downward progress of these constants may be found in [Ber04] and [Ber08]. In this paper we present results that improve on the best known constants for the *square root* and *reciprocal* operations.

Let $M(n)$ denote the cost of multiplying two polynomials in $\mathbf{C}[x]$ of degree less than n. By 'cost' or 'running time' we always mean number of ring operations in $\mathbf{C}$. We assume FFT-based multiplication throughout, so that $M(n) = O(n \log n)$.

Let $f \in \mathbf{C}[\![x]\!]$, $f = 1 \bmod x$. There are two algorithms for computing $f^{-1} \bmod x^n$ that achieve the previously best known running time bound of $(1.5+o(1))M(n)$. The first is that of Schönhage [Sch00, Theorem 2]. If g is an approximation to f^{-1} of length k, then the second-order Newton iteration $g' = (2g - g^2(f \bmod x^{2k}))$ yields an approximation to f^{-1} of length $2k$. Schönhage observed that it suffices to compute $g^2(f \bmod x^{2k})$ modulo $x^{3k} - 1$, which can be achieved by two forward FFTs and one inverse FFT of length $3k$. Iterating this process, the running time bound follows easily. Bernstein's 'messy' algorithm [Ber04, p. 10] is more complicated. Roughly speaking, he splits the input into blocks of consecutive coefficients and applies a second-order Newton iteration at the level of blocks. This blocking strategy allows transforms of blocks to be reused between iterations.

Our new reciprocal algorithm may be viewed as a simultaneous generalization of Bernstein's reciprocal algorithm and van der Hoeven's division algorithm [vdH09, p. 8]. The main innovation is the use of a third-order Newton iteration, whose additional term is computed essentially free of charge, leading to a running time of $(1.444\ldots+o(1))M(n)$ (Theorem 5). Although this is only a small improvement, it is interesting theoretically because the 'nice' bound $(1.5\ldots+o(1))M(n)$, achieved

Received by the editor November 10, 2009 and, in revised form, November 25, 2009.
2010 *Mathematics Subject Classification.* Primary 68W30.

by two quite different algorithms, had been a plausible candidate for the optimal bound for almost ten years. Furthermore, the methods presented in this paper suggest that $(1.333\ldots + o(1))M(n)$ may be attainable (see the final remark in Section 5).

For the square root, there are again two contenders for the previously best known bound of $(1.8333\ldots + o(1))M(n)$. Bernstein computes the square root and reciprocal square root together, alternately extending approximations of each [Ber04, p. 9]. Hanrot and Zimmermann first compute the reciprocal square root to half the target precision, using a technique similar to Schönhage's, and then apply a different iteration at the last step to obtain the square root [HZ04]. (They claim only $1.91666\ldots$, but there is an error in their analysis; the cost of line 3 of Algorithm 'SquareRoot' is $M(n)/3$, not $M(n)/2$.)

Our new square root algorithm achieves $(1.333\ldots + o(1))M(n)$ (Theorem 3). It is quite different from both of the above algorithms. It operates on blocks of coefficients and may be viewed as a straightforward adaptation of van der Hoeven's division algorithm to the case of extracting square roots.

For simplicity, we only discuss the case of $\mathbf{C}[\![x]\!]$. We make some remarks on the floating-point case in Section 6.

2. Notation and complexity assumptions

The Fourier transform $\mathcal{F}_n(g) \in \mathbf{C}^n$ of a polynomial $g \in \mathbf{C}[x]$ is defined by $(\mathcal{F}_n(g))_j = g(e^{2\pi ij/n})$. If $\deg g < n$, we denote by $T(n)$ the cost of computing $\mathcal{F}_n(g)$ from g, or of computing g from $\mathcal{F}_n(g)$.

If $g_1, g_2 \in \mathbf{C}[x]$ and $\deg g_i < n$, the cyclic convolution $g_1 g_2 \bmod x^n - 1$ may be computed by evaluating $\mathcal{F}_n^{-1}(\mathcal{F}_n(g_1)\mathcal{F}_n(g_2))$, where the Fourier transforms are multiplied componentwise. The running time is $3T(n) + O(n)$. To obtain the ordinary product $g_1 g_2$, one may compute $g_1 g_2 \bmod x^{2n'} - 1$ for any $n' \geq n$, leading to the estimate $M(n) = 3T(2n') + O(n')$. While it is known that $T(n) = O(n \log n)$ for all n, for sufficiently smooth n the implied big-O constant may be smaller than the worst case, and one should choose n' to take advantage of this. We therefore assume that we have available a set $S \subseteq \mathbf{Z}^+$ with the following properties: first, $T(2n) = (1/3 + o(1))M(n)$ for $n \in S$ and, second, the ratio of successive elements of S approaches 1. The choice of S will depend on exactly which FFT algorithms are under consideration. For example, Bernstein describes a particular class of FFT algorithms for which the above properties hold with $S = \{2^k m : m \text{ odd and } k \geq m^2 - 1\}$ [Ber04, p. 5]. Following Bernstein, we call elements of S *ultrasmooth* integers.

If $g, h \in \mathbf{C}[x]$, $\deg g < 2n$, $\deg h < n$, we denote by $g \rtimes_n h$ the middle product of g and h. That is, if $gh = p_0 + p_1 x^n + p_2 x^{2n}$ where $p_i \in \mathbf{C}[x]$, $\deg p_i < n$, then by definition $g \rtimes_n h = p_1$. Note that $gh = (p_0 + p_2) + p_1 x^n \bmod x^{2n} - 1$, so $g \rtimes_n h$ may be computed by evaluating $\mathcal{F}_{2n}^{-1}(\mathcal{F}_{2n}(g)\mathcal{F}_{2n}(h))$ and discarding the first half of the output. See [BLS03] and [HQZ04] for more information about the middle product.

In the algorithms given below, we fix a block size $m \geq 1$, and for $f \in \mathbf{C}[\![x]\!]$, we write $f = f_{[0]} + f_{[1]}X + f_{[2]}X^2 + \cdots$, where $X = x^m$ and $\deg f_{[i]} < m$.

3. Blockwise multiplication of power series

Our main tool is a technique for multiplying power series, described in the proof of Lemma 1 below. Bernstein used a similar idea in [Ber04, p. 10]. We follow

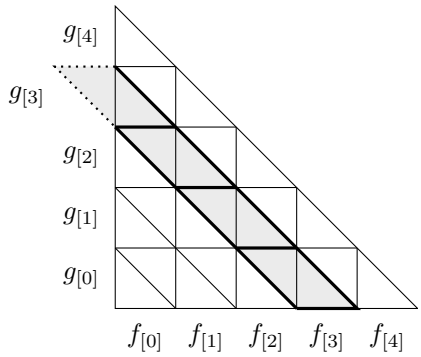

FIGURE 1. Blockwise product of power series. Terms contributing to $(fg)_{[3]}$ are shaded.

van der Hoeven's more recent approach [vdH09], which uses the middle product to obtain a neater algorithm.

Lemma 1. *Let $f, g \in \mathbf{C}[\![x]\!]$ and let $k \geq 0$. Given as input $\mathcal{F}_{2m}(f_{[0]}), \ldots, \mathcal{F}_{2m}(f_{[k]})$ and $\mathcal{F}_{2m}(g_{[0]}), \ldots, \mathcal{F}_{2m}(g_{[k]})$, we may compute $(fg)_{[k]}$ in time $T(2m)+O(m(k+1))$.*

Proof. As shown in Figure 1, we have

$$(fg)_{[k]} = \sum_{i=0}^{k} (f_{[k-i-1]} + f_{[k-i]}X) \rtimes_m g_{[i]},$$

where for convenience we declare that $f_{[-1]} = 0$. Therefore $(fg)_{[k]}$ is obtained as the second half of

$$\mathcal{F}_{2m}^{-1}\left(\sum_{i=0}^{k} \left(\mathcal{F}_{2m}(f_{[k-i-1]}) + \mathcal{F}_{2m}(f_{[k-i]})\mathcal{F}_{2m}(X)\right) \mathcal{F}_{2m}(g_{[i]})\right).$$

Since $(\mathcal{F}_{2m}(X))_j = (-1)^j$, the above expression may be computed from the $\mathcal{F}_{2m}(f_{[i]})$ and $\mathcal{F}_{2m}(g_{[i]})$ using $O(m(k+1))$ ring operations, followed by a single inverse transform of length $2m$. □

Remark. In Section 5, we will also need to compute expressions of the form $(fg + f'g')_{[k]}$, using the transforms of the blocks of f, f', g, and g' as input. Since the Fourier transform is linear, the same running time bound $T(2m)+O(m(k+1))$ applies (with a larger big-O constant).

4. Square root

If $f = f_0 + f_1x + f_2x^2 + \cdots$ and $g = f^{1/2} = g_0 + g_1x + g_2x^2 + \cdots$, then the coefficients of g may be determined by solving sequentially $g_0^2 = f_0$, $2g_0g_1 = f_1$, $2g_0g_2 = f_2 - g_1^2$, $2g_0g_3 = f_3 - 2g_1g_2$, $2g_0g_4 = f_4 - (2g_1g_3 + g_2^2)$, and so on. Algorithm 1 applies this procedure at the level of blocks, retaining the Fourier transform of each computed block as it proceeds.

Proposition 2. *Algorithm 1 is correct and runs in time $(4r-3)T(2m)+O(r^2m)$.*

Algorithm 1: Square root

Input: $r \in \mathbf{Z}$, $r \geq 1$
$f \in \mathbf{C}[\![x]\!]$, $f = 1 \bmod x$
$g_{[0]} = (f_{[0]})^{1/2} \bmod X$
$h = (g_{[0]})^{-1} \bmod X$

Output: $g = g_{[0]} + \cdots + g_{[r-1]}X^{r-1} = f^{1/2} \bmod X^r$

```
Compute F_{2m}(h)
for 1 ≤ k < r do
    Compute F_{2m}(g_{[k-1]})
    ψ ← ((g_{[0]} + ··· + g_{[k-1]}X^{k-1})^2)_{[k]}
    Compute F_{2m}(f_{[k]} − ψ)
    g_{[k]} ← (1/2)h(f_{[k]} − ψ) mod X
```

Proof. By definition $g_{[0]}$ is correct. In the kth iteration of the loop, assume that $g_{[0]}, \ldots, g_{[k-1]}$ have been computed correctly. Then we have

$$\begin{aligned}(g_{[0]} + \cdots + g_{[k-1]}X^{k-1})^2 &= f_{[0]} + \cdots + f_{[k-1]}X^{k-1} + \psi X^k \quad \bmod X^{k+1},\\ (g_{[0]} + \cdots + g_{[k]}X^{k})^2 &= f_{[0]} + \cdots + f_{[k-1]}X^{k-1} + f_{[k]}X^k \quad \bmod X^{k+1}.\end{aligned}$$

Subtracting these yields $2g_{[0]}g_{[k]} = f_{[k]} - \psi \bmod X$, so $g_{[k]}$ is computed correctly.

Each iteration performs one inverse transform to obtain ψ (Lemma 1), one to obtain $g_{[k]}$, and the two forward transforms explicitly stated. The total number of transforms is therefore $4(r-1)+1 = 4r-3$. The loop also performs $O(km)$ scalar operations in the kth iteration (Lemma 1). □

Theorem 3. *The square root of a power series $f = 1 + f_1 x + \cdots \in \mathbf{C}[\![x]\!]$ may be computed to order n in time $(4/3 + o(1))M(n)$.*

Proof. Let $r \geq 1$, and let m be the smallest ultrasmooth integer larger than n/r. We let r grow slowly with respect to n; specifically, we assume that $r \to \infty$ and $r = o(\log n)$ as $n \to \infty$. Then $m \to \infty$ as $n \to \infty$, so $m = (1 + o(1))n/r$. Zero-pad f up to length rm. Compute $g_{[0]} = (f_{[0]})^{1/2} \bmod x^m$ and $h = (g_{[0]})^{-1} \bmod x^m$ using any $O(M(m))$ algorithm. Compute $f^{1/2} \bmod x^{rm}$, hence $f^{1/2} \bmod x^n$, using Algorithm 1. By Proposition 2 the total cost is

$$\begin{aligned}O(M(m)) + (4r-3)T(2m) + O(r^2 m) &= (4r/3 + O(1))M(m) + O(r^2 m)\\ &= (4/3 + O(r^{-1}))M(mr) + O(r^2 m)\\ &= (4/3 + O(r^{-1}))M(n) + O(rn)\\ &= (4/3 + o(1))M(n),\end{aligned}$$

assuming that $M(n) = \Theta(n \log n)$. □

Remark. If $g_{[0]}$ and h are computed using Bernstein's $(2.5 + o(1))M(n)$ algorithm for the simultaneous computation of the square root and reciprocal square root [Ber04, p. 9], then already for $r = 3$ the new algorithm matches the previous bound of $(1.8333\ldots + o(1))M(n)$ and is strictly faster for $r \geq 4$.

Remark. Let $f \in \mathbf{C}[x]$ be a monic polynomial of degree $2n$. The above algorithm may be adapted to compute the square root with remainder, that is, polynomials $g, h \in \mathbf{C}[x]$ with $\deg g = n$, $\deg h < n$, and $f = g^2 + h$, in time $(5/3 + o(1))M(n)$.

For this, write $\tilde{f}(x) = x^{2n} f(1/x)$, $\tilde{g}(x) = x^n g(1/x)$, $\tilde{h}(x) = x^n h(1/x)$. Then $\tilde{f}, \tilde{g}, \tilde{h} \in \mathbf{C}[\![x]\!]$, and we want to solve $\tilde{f}(x) = \tilde{g}(x)^2 + x^n \tilde{h}(x)$. First compute $\tilde{g}(x)$ using the above algorithm; to find $\tilde{h}(x)$, it then suffices to compute $\tilde{g}(x)^2$. Observe that at the end of Algorithm 1, we may compute $((\tilde{g}_{[0]} + \cdots + \tilde{g}_{[r-1]} X^{r-1})^2)_{[j]}$ for $r \le j < 2r$ in time $rT(2m) + O(r^2 m)$ using Lemma 1, since the transforms of the $\tilde{g}_{[j]}$ are all known. This increases the cost from $(4r-3)T(2m) + O(r^2 m)$ to $(5r-3)T(2m) + O(r^2 m)$, leading to the claimed bound in the same way as in the proof of Theorem 3.

5. Reciprocal

Let $f = 1 + f_1 x + \cdots \in \mathbf{C}[\![x]\!]$, and suppose that $g = f^{-1} \bmod x^n$. Then $fg = 1 + \delta x^n \bmod x^{3n}$ for some $\delta \in \mathbf{C}[x]$, $\deg \delta < 2n$. Putting $g' = g(1 - \delta x^n + \delta^2 x^{2n})$, we have $g' = f^{-1} \bmod x^{3n}$. This is the third-order Newton iteration for the reciprocal.

The idea of Algorithm 2 below is to use the above recipe at the level of blocks, with an additional twist. If we write $\delta = \delta_0 + \delta_1 x^n$, where $\deg \delta_i < n$, then $g' = g(1 - \delta_0 x^n + (\delta_0^2 - \delta_1) x^{2n}) \bmod x^{3n}$. The algorithm first computes δ_0, applying Lemma 1 to compute the relevant blocks of fg. Then, instead of computing δ_0^2 and δ_1 separately, it computes the sum $\delta_0^2 - \delta_1$ in one pass, using only one inverse transform per block (see the remark after Lemma 1). This is possible since δ_0 is already completely known and constitutes the main source of savings over Bernstein's algorithm.

Proposition 4. *Algorithm* 2 *is correct and runs in time* $(13s-3)T(2m) + O(s^2 m)$.

Proof. By definition $g_{[0]}$ is correct. Lines 3–7 compute $g_{[1]}, \ldots, g_{[s-1]}$ as follows. In the kth iteration of the loop, assume that $g_{[0]}, \ldots, g_{[k-1]}$ are correct. Then

$$\begin{aligned} (f_{[0]} + \cdots + f_{[k]} X^k)(g_{[0]} + \cdots + g_{[k-1]} X^{k-1}) &= 1 + \psi X^k \quad \bmod X^{k+1}, \\ (f_{[0]} + \cdots + f_{[k]} X^k)(g_{[0]} + \cdots + g_{[k]} X^k) &= 1 \quad \bmod X^{k+1}. \end{aligned}$$

Subtracting yields $f_{[0]} g_{[k]} = -\psi \bmod X$, so $g_{[k]}$ is computed correctly. (This loop is essentially van der Hoeven's division algorithm, applied to compute $1/f \bmod X^s$.)

Now we use the symbols δ, δ_0, δ_1 introduced earlier, putting $n = sm$. After lines 8–10 we have $d_{[0]} + \cdots + d_{[s-1]} X^{s-1} = -\delta_0$, and the subsequent loop computes $d_{[s]} + \cdots + d_{[2s-1]} X^{s-1} = \delta_0^2 - \delta_1 \bmod X^s$. Therefore $d_{[0]} + \cdots + d_{[2s-1]} X^{2s-1} = -\delta + \delta^2 X^s \bmod X^{2s}$. The final loop computes the appropriate blocks of $g' = g(1 - \delta X^s + \delta^2 X^{2s}) \bmod X^{3s}$.

Altogether the algorithm performs $1 + 3s + 2(s-1) + s + s$ forward transforms, $2(s-1) + s + s + 2s$ inverse transforms, and $O(s^2 m)$ scalar operations (apply Lemma 1 to each loop). □

Theorem 5. *The reciprocal of a power series* $f \in \mathbf{C}[\![x]\!]$ *may be computed to order* n *in time* $(13/9 + o(1))M(n)$.

Proof. Apply the proof of Theorem 3 to Proposition 4, with $r = 3s$. □

Remark. If $g_{[0]}$ is computed using a $(1.5 + o(1))M(n)$ algorithm, the new algorithm achieves the same bound for $s = 3$ ($r = 9$) and is faster for $s \ge 4$ ($r \ge 12$).

Algorithm 2: Reciprocal

Input: $s \in \mathbf{Z}$, $s \geq 1$
$f \in \mathbf{C}[\![x]\!]$, $f = 1 \bmod x$
$g_{[0]} = (f_{[0]})^{-1} \bmod X$
Output: $g = g_{[0]} + \cdots + g_{[3s-1]}X^{3s-1} = f^{-1} \bmod X^{3s}$

Compute $\mathcal{F}_{2m}(g_{[0]})$
for $0 \leq i < 3s$ **do** compute $\mathcal{F}_{2m}(f_{[i]})$
for $1 \leq k < s$ **do**
  $\psi \leftarrow ((f_{[0]} + \cdots + f_{[k]}X^k)(g_{[0]} + \cdots + g_{[k-1]}X^{k-1}))_{[k]}$
  Compute $\mathcal{F}_{2m}(\psi)$
  $g_{[k]} \leftarrow -g_{[0]}\psi \bmod X$
  Compute $\mathcal{F}_{2m}(g_{[k]})$
for $0 \leq k < s$ **do**
  $d_{[k]} \leftarrow -((f_{[0]} + \cdots + f_{[3s-1]}X^{3s-1})(g_{[0]} + \cdots + g_{[s-1]}X^{s-1}))_{[k+s]}$
  Compute $\mathcal{F}_{2m}(d_{[k]})$
for $s \leq k < 2s$ **do**
  $d_{[k]} \leftarrow ((d_{[0]} + \cdots + d_{[s-1]}X^{s-1})^2)_{[k-s]}$
    $- ((f_{[0]} + \cdots + f_{[3s-1]}X^{3s-1})(g_{[0]} + \cdots + g_{[s-1]}X^{s-1}))_{[k+s]}$
  Compute $\mathcal{F}_{2m}(d_{[k]})$
for $s \leq k < 3s$ **do**
  $g_{[k]} \leftarrow ((d_{[0]} + \cdots + d_{[2s-1]}X^{2s-1})(g_{[0]} + \cdots + g_{[s-1]}X^{s-1}))_{[k-s]}$

Remark. A natural question is whether the key idea of Algorithm 2 can be extended to Newton iterations of arbitrarily high order. That is, if $fg = 1 + \delta x^n$, is it possible to compute $1 - \delta x^n + \delta^2 x^{2n} \cdots \pm \delta^{k-1}x^{(k-1)n} \bmod x^{kn}$ in essentially the same time as $1 + \delta x^n \bmod x^{kn}$ itself, for arbitrary k? Algorithm 2 corresponds to the case $k = 3$. An affirmative answer for arbitrary k would presumably lead to a $(1.333\ldots + o(1))M(n)$ algorithm for the reciprocal.

6. The floating-point case

It seems likely that the algorithms presented in this paper may be adapted to achieve the same constants relative to multiplication for computing square roots or reciprocals of floating-point numbers with n bits. We have not checked the details, but we make a few relevant comments here.

Apart from the usual complications arising from carries, it is important to note that the algorithms, as we have presented them, rely on an 'ideal' FFT model and that FFT-based integer multiplication algorithms may depart from this ideal model in various ways. For example, in the Schönhage–Strassen algorithm [SS71], the Fourier coefficients are of size $O(\sqrt{n})$, and a large proportion of the total time is taken up by the recursive multiplications. One would need to include the cost of the FFTs *within* such recursive multiplications in the overall FFT cost. This problem does not occur for integer multiplication algorithms based on a floating-point FFT or number-theoretic transform, where the coefficients are of size $O(\log n)$, although

in these cases one must take care to allow enough bits of precision to accommodate the accumulated sums of transforms in (for example) the main loop of Algorithm 1. In the context of Fürer's recent algorithm [Für07], extreme care must be taken to avoid additional overhead of even $O(\log \log n)$.

Finally we mention that we have not yet been able to adapt the algorithms to FFT models based on the truncated Fourier transform [vdH04, vdH05]. While the division algorithm of [vdH09] can be made to work with the TFT, our Lemma 1 implicitly uses the symmetry of the FFT matrix, the analogue of which does not hold for the TFT.

Acknowledgments

Many thanks to Paul Zimmermann and the referees for their suggestions that greatly improved the presentation of these results.

Postscript

Prior to publication of these results, it was brought to the author's attention that Igor Sergeev found better constants in 2007. He obtains complexity $(5/4 + o(1))M(n)$ for both the reciprocal and square root, in a similar FFT model (Proc. IX International Conf., "Discrete mathematics and its applications", Moscow State University, pp. 123–126, in Russian). However, Sergeev's method requires considering products of *three* Fourier transforms and appears unlikely to achieve the same constants in a synthetic FFT model (Sergeev, personal communication). It is unclear whether it can achieve the same constants in the integer or floating-point setting.

References

[Ber04] Daniel Bernstein, *Removing redundancy in high-precision Newton iteration*, unpublished, available at `http://cr.yp.to/papers.html#fastnewton`, 2004.

[Ber08] ———, *Fast multiplication and its applications*, Algorithmic number theory: Lattices, Number fields, Curves and Cryptography, Math. Sci. Res. Inst. Publ., vol. 44, Cambridge Univ. Press, Cambridge, 2008, pp. 325–384. MR2467550 (2010a:68186)

[BLS03] Alin Bostan, Grégoire Lecerf, and Éric Schost, *Tellegen's principle into practice*, Symbolic and Algebraic Computation (J. R. Sendra, ed.), ACM Press, 2003, Proceedings of ISSAC'03, Philadelphia, August 2003., pp. 37–44.

[Für07] Martin Fürer, *Faster integer multiplication*, STOC'07—Proceedings of the 39th Annual ACM Symposium on Theory of Computing, ACM, New York, 2007, pp. 57–66. MR2402428 (2009e:68124)

[HQZ04] Guillaume Hanrot, Michel Quercia, and Paul Zimmermann, *The middle product algorithm, I*, Appl. Algebra Engrg. Comm. Comput. **14** (2004), no. 6, 415–438. MR2042800 (2005a:65003)

[HZ04] Guillaume Hanrot and Paul Zimmermann, *Newton iteration revisited*, unpublished, available at `http://www.loria.fr/~zimmerma/papers/fastnewton.ps.gz`, 2004.

[Sch00] Arnold Schönhage, *Variations on computing reciprocals of power series*, Inform. Process. Lett. **74** (2000), no. 1-2, 41–46. MR1761197 (2001c:68069)

[SS71] Arnold Schönhage and Volker Strassen, *Schnelle Multiplikation grosser Zahlen*, Computing (Arch. Elektron. Rechnen) **7** (1971), 281–292. MR0292344 (45:1431)

[vdH04] Joris van der Hoeven, *The truncated Fourier transform and applications*, ISSAC 2004, ACM, New York, 2004, pp. 290–296. MR2126956

[vdH05] ———, *Notes on the truncated Fourier transform*, unpublished, available from `http://www.math.u-psud.fr/~vdhoeven/`, 2005.

[vdH09] ———, *Newton's method and FFT trading*, preprint available at `http://hal.archives-ouvertes.fr/hal-00434307/`, 2009.

Courant Institute of Mathematical Sciences, New York University, 251 Mercer St, New York, New York 10012

E-mail address: `dmharvey@cims.nyu.edu`