A high-performance algorithm for calculating cyclotomic polynomials.

Andrew Arnold
Department of Mathematics
Simon Fraser University
Burnaby, B.C. Canada.
ada26@sfu.ca.

Michael Monagan
Department of Mathematics
Simon Fraser University
Burnaby, B.C. Canada.
mmonagan@cecm.sfu.ca.

ABSTRACT

The n_{th} cyclotomic polynomial, $\Phi_n(z)$, is the monic polynomial whose $\phi(n)$ distinct roots are the n_{th} primitive roots of unity. $\Phi_n(z)$ can be computed efficiently as a quotient of terms of the form $(1-z^d)$ by way of a method the authors call the Sparse Power Series algorithm. We improve on this algorithm in three steps, ultimately deriving a fast, recursive algorithm to calculate $\Phi_n(z)$. The new algorithm, which we have implemented in C, allows us to compute $\Phi_n(z)$ for $n > 10^9$ in less than one minute.

1. INTRODUCTION

The n_{th} cyclotomic polynomial, $\Phi_n(z)$, is the minimal polynomial over $\mathbb{Q}$ of the n_{th} primitive roots of unity.

$$\Phi_n(z) = \prod_{\substack{j=1 \\ \gcd(j,n)=1}}^{n} \left(z - e^{\frac{2\pi i}{n} j}\right). \tag{1.1}$$

We let the *index* of $\Phi_n(z)$ be n and the *order* of $\Phi_n(z)$ be the number of distinct odd prime divisors of n. The n_{th} inverse cyclotomic polynomial, $\Psi_n(z)$, is the polynomial whose roots are the n_{th} non-primitve roots of unity:

$$\Psi_n(z) = \prod_{\substack{j=1 \\ \gcd(j,n)>1}}^{n} \left(z - e^{\frac{2\pi i}{n} j}\right) = \frac{z^n - 1}{\Phi_n(z)}. \tag{1.2}$$

We denote by $A(n)$ the *height* of $\Phi_n(z)$, that is, the largest coefficient in magnitude of $\Phi_n(z)$. It is well known that for $n < 105$, $A(n) = 1$ but for $n = 105$, $A(n) = 2$. The smallest n with $A(n) > 2$ is $n = 385$ where $A(n) = 3$. Although the heights appear to grow very slowly, Paul Erdős proved in [2] that $A(n)$ is not bounded above by any polynomial in n, that is, for any constant $c > 0$, there exist n such that $A(n) > n^c$. A natural question to ask is, what is the first n for which $A(n) > n$, that is the the height exceeds the index?

In earlier work [1], we developed two asymptotically fast algorithms to compute $\Phi_n(z)$. The first algorithm, which we call the FFT algorithm, uses the Fast Fourier Transform to perform a sequence of polynomial exact divisions in $\mathbb{Z}[z]$ modulo a prime q. Using this algorithm we found the smallest n such that $A(n) > n$, namely for $n = 1,181,895$, the height $A(n) = 14,102,773$. Here $n = 3 \cdot 5 \cdot 11 \cdot 13 \cdot 19 \cdot 29$. To find $\Phi_n(z)$ with larger height, we tried simply multiplying this n by additional primes. In this way we found an n with $A(n) > n^2$ and several $n > 10^9$ with $A(n) > n^4$, the latter requiring the use of a supercomputer with a lot of RAM.

The second algorithm, which we call the Sparse Power Series (SPS) algorithm, does a sequence of sparse series multiplications and divisions in $O(2^k \phi(n))$ integer arithmetic operations. Although not asymptotically faster than the FFT algorithm, it turns out that because the SPS algorithm only needs integer additions and subtractions, it is considerably faster (more than 20 times - see section 4) than the FFT algorithm. Using the SPS algorithm we found the smallest n with $A(n) > n^2$, $A(n) > n^3$ and $A(n) > n^4$, namely $n = 43,730,115$, $n = 416,690,995$, and $1,880,394,945$, respectively, as well as other new results. One of the difficulties when $n > 10^9$ is space. For such n, even storing $\Phi_n(z)$ requires many gigabytes of memory. The SPS algorithm has a substantial space advantage over the FFT algorithm. It has now been implemented in the Sage and Maple 13 computer algebra systems.

In this paper we present a fast recursive algorithm to calculate $\Phi_n(z)$ and $\Psi_n(z)$. It improves on the sparse power series (**SPS**) algorithm by approximately another factor of 10 (see section 4). To give one specific benchmark; Yoichi in [4, 5] found $A(n)$ for n the product of the first 7 odd primes but was unable to determine $A(n)$ for n the product of the first 8 primes. We used the FFT algorithm to find $A(n)$ for n the product of the first 9 odd primes in approx. 12 hours. The SPS algorithm takes 7 minutes and our new algorithm takes 50 seconds. A challenge problem given to us by Noe [6] is to compute $\Phi_n(z)$ for $n = 99,660,932,085$ which we expect will have a huge height. The main difficulty now is space; for the mentioned unsolved problem, storing $\Phi_n(z)$ to 64-bit precision requires over 150 GB of space.

Our paper is organized as follows. Section 2 presents identities involving $z^n - 1$, $\Phi_n(z)$ and $\Psi_n(z)$ used in the algorithms, and the basic algorithm used for the FFT approach. Section 3 details the Sparse Power Series algorithm for computing $\Phi_n(z)$ and introduces a similar algorithm for computing $\Psi_n(z)$, then develops improvements in three steps. The third step makes the algorithm recursive. Section 4 presents some timings comparing the FFT algorithm, the original SPS algorithm, and the three improvements.

2. USEFUL IDENTITIES OF CYCLOTOMIC POLYNOMIALS

Before describing the algorithms, we establish some basic identities of cyclotomic polynomials. First, as the roots of $\Phi_n(z)$ and $\Psi_n(z)$ consist of all n_{th} roots of unity, we have that

$$\Phi_n(z)\Psi_n(z) = \prod_{j=0}^{n-1}(z - e^{\frac{2\pi j}{n}i}) = z^n - 1. \tag{2.1}$$

Every n_{th} root of unity is a d_{th} primitive root of unity for some unique $d|n$, $d > 0$. Conversely, if $d|n$, every d_{th} primitive root of unity is trivially an n_{th} root of unity. As such,

$$\prod_{d|n}\Phi_d(z) = z^n - 1. \tag{2.2}$$

Applying the Möbius inversion formula to (2.2), we have

$$\Phi_n(z) = \prod_{d|n,d>0}(z^d - 1)^{\mu(\frac{n}{d})}, \tag{2.3}$$

where μ is the Möbius function. For non squarefree m, $\mu(m) = 0$. As the number of positive squarefree divisors of $n > 1$ is even, it is oftentimes convenient to write, for $n > 1$,

$$\Phi_n(z) = \prod_{d|n}(1 - z^d)^{\mu(\frac{n}{d})}. \tag{2.4}$$

From (2.1) and (2.3) we obtain a similar identity for $\Psi_n(z)$:

$$\Psi_n(z) = \prod_{d|n,0<d<n}(z^d - 1)^{-\mu(\frac{n}{d})}, \tag{2.5}$$

Since the product in (2.5) contains every positive squarefree divisor of n except for n, it has an odd number of terms for $n > 1$. That is, for $n > 1$,

$$\Psi_n(z) = -\prod_{d|n,0<d<n}(1 - z^d)^{-\mu(\frac{n}{d})}. \tag{2.6}$$

From (2.1) and (2.2), we have that

$$\Psi_n(z) = \prod_{d|n,0<d<n}\Phi_d(z). \tag{2.7}$$

Indeed, every n_{th} nonprimitive root of unity is a d_{th} primitive root of unity for some $d|n$, $d < n$.

Given $\Phi_1(z) = z - 1$ and $\Psi_1(z) = 1$, we can compute all cyclotomic polynomials using the following lemma:

LEMMA 1. *Let p, q be primes such that $p \nmid n$, and $q|n$. Then*

$$\Phi_{np}(z) = \frac{\Phi_n(z^p)}{\Phi_n(z)}, \tag{2.8a}$$

$$\Phi_{nq}(z) = \Phi_n(z^q), \tag{2.8b}$$

$$\Psi_{np}(z) = \Psi_n(z^p)\Phi_n(z), \text{ and} \tag{2.8c}$$

$$\Psi_{nq}(z) = \Psi_n(z^q). \tag{2.8d}$$

PROOF. By (2.1), $\Phi_{np}(z)\Psi_{np}(z) = \Phi_n(z^p)\Psi_n(z^p) = z^{np} - 1$. Thus if (2.8a) holds, we have

$$\Psi_{np}(z) = \frac{z^{np}-1}{\Phi_{np}(z)} = \frac{(z^{np}-1)}{\Phi_n(z^p)}\Phi_n(z) = \Psi_n(z^p)\Phi_n(z). \tag{2.9}$$

Thus (2.8a) is a sufficient condition for (2.8c). Similarly, (2.8d) follows from (2.8b).

As $\mu(d)$ is nonzero only if d is squarefree, we can rewrite (2.4) as

$$\Phi_n(z) = \prod\nolimits^*(1 - z^{\frac{n}{d}})^{\mu(d)}, \tag{2.10}$$

where the product $\prod^*$ is over all squarefree divisors d of n. Suppose d is a squarefree divisor of np, then either d is a squarefree divisor of n, or d is of the form $d = d'p$, where d' is a squarefree divisor of n. Thus

$$\Phi_{np}(z) = \prod\nolimits^*(1 - z^{\frac{np}{d}})^{\mu(d)} \cdot \prod\nolimits^*(1 - z^{\frac{np}{dp}})^{\mu(dp)} \tag{2.11}$$

The first product is exactly $\Phi_n(z^p)$. Since the divisors of n and p are coprime, $\mu(dp) = -\mu(d)$, and thus

$$\begin{aligned}\Phi_{np}(z) &= \Phi_n(z^p) \cdot \prod\nolimits^*(1 - z^{\frac{np}{dp}})^{-\mu(d)} \\ &= \Phi_n(z^p) \cdot \Phi_n(z)^{-1},\end{aligned} \tag{2.12}$$

proving (2.8a).

To prove (2.8b), we observe that as q divides n, the squarefree divisors of nq are exactly the squarefree divisors of q, and thus

$$\Phi_{nq}(z) = \prod\nolimits^*(1 - z^{\frac{nq}{d}})^{\mu(d)} = \Phi_n(z^q). \tag{2.13}$$

□

Lemma 1 effectively gives us a means to calculate $\Phi_n(z)$. For instance, for $n = 75 = 3 \cdot 5^2$, we have that

$$\begin{aligned}\Phi_3(z) &= \tfrac{\Phi_1(z^3)}{\Phi_1(z)} = \tfrac{z^3-1}{z-1} = z^2 + z + 1, \text{ and} \\ \Phi_{15}(z) &= \tfrac{\Phi_3(z^5)}{\Phi_3(z)} = \tfrac{z^{10}+z^5+1}{z^2+z+1} \\ &= z^8 - z^7 + z^5 - z^4 + z^3 - z + 1, \text{ by (2.8a).} \\ \Phi_{75}(z) &= \Phi_{15}(z^5), \\ &= z^{40} - z^{35} + z^{25} - z^{20} + z^{15} - z^5 + 1, \text{ by (2.8b).}\end{aligned}$$

We describe this approach in algorithm 1 below:

While algorithm 1 is beautifully simple, it is not nearly the fastest way to compute $\Phi_n(z)$, particularly if we use classical polynomial division to calculate the quotient of $\Phi_d(z^{p_i})$ divided by $\Phi_d(z)$. For even though the numerator is sparse, the denominator and quotient are typically dense.

We implemented algorithm 1 using the discrete fast Fourier transform (FFT) to divide $\Phi_m(z^{p_i})$ by $\Phi_m(z)$ fast. This is done modulo suitably chosen primes q_j. The cost per prime q_j is $\mathcal{O}(n \log n)$. For a detailed description of the discrete FFT, we refer the reader to [3]. We compute images of $\Phi_n(z)$ modulo sufficiently many primes and recover the integer coefficients of $\Phi_n(z)$ using Chinese remaindering. In order to apply the FFT modulo q, we need a prime q with $2^k|q - 1$ and $2^k > \phi(n)$, the degree of the resulting polynomial $\Phi_n(z)$. For $n > 10^9$ since there are no such 32 bit primes, we cannot directly apply the FFT on a 64 bit machine using machine integer arithmetic. For $n > 10^9$ we used 42-bit primes with arithmetic modulo q coded using 64-bit machine integer arithmetic.

Algorithm 1: Computing $\Phi_n(z)$ by repeated polynomial division

Input: $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e^k}$, where $p_1 < \cdots < p_k$ and $e_i > 0$ for $1 \le i \le k$
Output: $\Phi_n(z)$
//Applying (2.8a):
$m \longleftarrow 1$
$\Phi_m(z) \longleftarrow z - 1$
for $i = 1$ **to** k **do**
  $\Phi_{mp_i}(z) \longleftarrow \frac{\Phi_m(z^{p_i})}{\Phi_m(z)}, m \longleftarrow m \cdot p_i$
//m is the largest squarefree divisor of n now
$s \longleftarrow n/m$
//Applying (2.8b):
$\Phi_n(z) \longleftarrow \Phi_m(z^s)$
return $\Phi_n(z)$

3. A HIGH-PERFORMANCE ALGORITHM FOR COMPUTING $\Phi_N(Z)$

For the purposes of this section we only consider $\Phi_n(z)$ and $\Psi_n(z)$ where n is odd and squarefree.

If n is odd, then ω is an n_{th} primitive root of unity if and only if $-\omega$ is a $2n_{th}$ primitive root of unity. Given such, one can verify the first identity of following lemma by showing both sides of the equality have the same roots:

LEMMA 2. *Let $n > 1$ be odd, then*

$$\Phi_{2n}(z) = \Phi_n(-z) \text{ and} \tag{3.1}$$
$$\Psi_{2n}(z) = -\Psi_n(-z)(z^n + 1). \tag{3.2}$$

The identity (3.2) follows from (3.1). As

$$\Phi_{2n}(z)\Psi_{2n}(z) = z^{2n} - 1 = (z^n - 1)(z^n + 1) \text{ and} \tag{3.3}$$
$$\Phi_n(-z)\Psi_n(-z) = (-z)^n - 1 = -(z^n + 1), \tag{3.4}$$

we thus have by (3.1) that

$$\Psi_{2n}(z) = -\Psi_n(-z)(z^n + 1) \text{ for odd } n. \tag{3.5}$$

If n is a nonsquarefree integer with largest odd squarefree divisor $\bar{n}$, lemmas 1 and 2 provides an easy means of obtaining $\Phi_n(z)$ and $\Psi_n(z)$ from $\Phi_{\bar{n}}(z)$ and $\Psi_{\bar{n}}(z)$ respectively.

From (2.4), we have that $\Phi_n(z)$ is a quotient of terms of the form $(1 - z^d)$. Given that the power series expansion of $(1 - z^d)^{-1}$ is $(1 + z^d + z^{2d} + z^{3d} + \dots)$, it becomes equally easy to either multiply or divide by $(1 - z^d)$. As we will refer to the terms $1 - z^d$ of the quotient (2.4) often, we call them the *subterms* of $\Phi_n(z)$. We can rewrite (2.4) as

$$\Phi_n(z) = \prod_{\mu(\frac{n}{d})=1} (1 - z^d) \cdot \prod_{\mu(\frac{n}{d})=-1} (1 + z^d + z^{2d} + \dots). \tag{3.6}$$

$\Phi_n(z)$ can be computed as a truncated power series by this approach, as described in procedure SPS.

Note that while $1 - z^n$ appears in (2.4), we do not multiply by $1 - z^n$ is algorithm SPS, as it does not affect our result. This is because $1 - z^n \equiv 1 \pmod{z^D}$ for $D = \phi(n)/2$.

Given (2.6), we derive an analgous method for $\Psi_n(z)$, described by procedure SPS-Psi.

As $n = p_1 p_2 \dots p_k$ has 2^k positive divisors, the SPS algorithm takes $\mathcal{O}(2^k \cdot \phi(n))$ operations in $\mathbb{Z}$ to calculate $\Phi_n(z)$, and $\mathcal{O}(2^k \cdot (n - \phi(n))$ operations to calculate $\Psi_n(z)$.

Procedure SPS(n), computing $\Phi_n(z)$ as a quotient of sparse power series

The Sparse Power Series (SPS) Algorithm

Input: n a squarefree, odd integer
Output: $a(0), \dots, a(\frac{\phi(n)}{2})$, the first half of the coefficients of $\Phi_n(z)$
//we compute terms up to degree D
$D \longleftarrow \frac{\phi(n)}{2}$, $a(0) \longleftarrow 1$
for $1 \le i \le M$ **do** $a(i) \longleftarrow 0$
for $d|n$ *such that* $0 < d < n$ **do**
  if $\mu(\frac{n}{d}) = 1$ **then**
    //multiply by $1 - z^d$
    for $i = D$ **down to** d **by** -1 **do**
    $a(i) \longleftarrow a(i) - a(i - d)$
  else
    //divide by $1 - z^d$
    for $i = d$ **to** D **do** $a(i) \longleftarrow a(i) + a(i - d)$
return $a(0), a(1), \dots a(D)$

Procedure SPS-Psi(n), computing $\Psi_n(z)$ as a quotient of sparse power series

The Sparse Power Series (SPS) Algorithm for $\Psi_n(z)$

Input: n a squarefree, odd integer
Output: $b(0), \dots, b(\lfloor \frac{n-\phi(n)}{2} \rfloor)$, the first half of the coefficients of $\Psi_n(z)$
$D \longleftarrow \lfloor \frac{n-\phi(n)}{2} \rfloor$, $b(0) \longleftarrow 1$
for $1 \le i \le D$ **do** $b(i) \longleftarrow 0$
for $d|n$ *such that* $0 < d < n$ **do**
  if $\mu(\frac{n}{d}) = 1$ **then**
    //multiply by $1 - z^d$
    for $i = D$ **down to** d **by** -1 **do**
    $b(i) \longleftarrow b(i) - b(i - d)$
  else
    //divide by $1 - z^d$
    for $i = d$ **to** D **do** $b(i) \longleftarrow b(i) + b(i - d)$
return $b(0), b(1), \dots, b(D)$

We need only truncate to half the degree of $\Phi_n(z)$ (or $\Psi_n(z)$) because of the following lemma.

LEMMA 3. *Suppose $n > 1$ and set $\sum_{i=0}^{\phi(n)} a_i z^i = \Phi_n(z)$ and $\sum_{i=0}^{n-\phi(n)} b_i z^i = \Psi_n(z)$, then*

$$a_i = a_{\phi(n)-i} \text{ and } b_j = b_{n-\phi(n)-j}. \tag{3.7}$$

That is, the coefficients of $\Phi_n(z)$ are palindromic and the coefficients of $\Psi_n(z)$ are antipalindromic.

PROOF. If ω is a primitive root of unity, then ω^{-1} is too. Thus

$$z^{\phi(n)}\Phi_n(z^{-1}) = \sum_{i=0}^{\phi(n)} a_{\phi(n)-i} z^i \tag{3.8}$$

is a polynomial of degree $\phi(n)$ with leading term $a_0 z^{\phi(n)}$ whose roots are exactly the roots of $\Phi_n(z)$. From the identity

(3.6) we have that the constant term of $\Phi_n(z)$, a_0, is 1. Thus $\Phi_n(z) = z^{\phi}(n)\Phi_n(z^{-1}) = \sum a_{\phi(n)-i}z^i$, and so $a_i = a_{\phi(n)-i}$.

Similarly, if ω is a nonprimitive root of unity, then ω^{-1} is as well. Thus $z^{n-\phi(n)}\Psi_n(z^{-1})$ is a polynomial of degree $\phi(n)$ with leading term $b_0 z^{n-\phi(n)}$ whose roots are the roots of $\Psi_n(z)$. Since

$$\sum_{i=0}^{\phi(n)} a_i z^i \cdot \sum_{j=0}^{n-\phi(n)} b_j z^j = \Phi_n(z)\Psi_n(z) = z^n - 1, \qquad (3.9)$$

it follows from the expansion of $\sum a_i z^i \sum b_j z^j$ that $a_0 b_0 = -1$. That is, $b_0 = -1$, and thus $\Psi_n(z) = -z^{n-\phi(n)}\Psi_n(z) = \sum -b_{n-\phi(n)-j}z^j$. It follows that $b_j = b_{n-\phi(n)-j}z^j$.

□

Note that lemma 3 does not hold for $\Phi_1(z) = z-1$, which is antipalindromic and $\Psi_n(z) = 1$, which is trivially palindromic. We can generalize lemma 3 to products of cyclotomic polynomials:

LEMMA 4. *Let*

$$f(z) = \Phi_{n_1}(z) \cdot \Phi_{n_2}(z) \cdots \Phi_{n_k}(z) = \sum_{i=0}^{D} c_i z^i \qquad (3.10)$$

be a product of cyclotomic polynomials such that n_j is odd for $1 \le j < k$. Then if D is odd $c_i = -c_{D-i}$ for $0 \le i < D$. If n is even, $c_i = c_{D-i}$ for $0 \le i < D$. In other words, if D is odd $f(z)$ is antipalindromic, and if D is even $f(z)$ is palindromic.

PROOF. Clearly $f(z)$ is monic. As before, we observe that if ω is a root of f, then ω^{-1} is too. Set

$$g(z) = z^D f(z^{-1}) = \sum_{i=0}^{D} c_{D-i}(z). \qquad (3.11)$$

$g(z)$ is a polynomial of degree D with leading coefficient c_0 whose roots are the roots of f. Thus $f(z)$ and $g(z)$ only differ by a constant factor. We need only resolve c_0. To that end, we observe that $\phi(n)$ is even for odd $n > 1$, and $\phi(1) = 1$. Thus $r \equiv D \bmod 2$, where r is the cardinality of

$$\{j : 1 \le j \le k \text{ and } n_j = 1\}. \qquad (3.12)$$

Note that the constant term of f, c_0, is the product of the constant terms of the $\Phi_{n_j}(z)$ in (3.10). Since the constant term of $\Phi_1(z) = z-1$ is -1, and by (2.4), the constant term of $\Phi_n(z)$ is 1 for $n > 1$, we have that $c_0 = (-1)^r = (-1)^D$.

Thus if D is even, $c_0 = 1$, thus $g(z)$ is monic and equal to $f(z)$. It follows from (3.11) that $c_i = c_{D-i}$. If D is odd, $c_0 = -1$ and so $f(z) = -g(z)$. In which case, $c_i = -c_{D-i}$. □

We note that lemma 4 does not hold if we relax the restriction that n_j must be odd in (3.10). Consider the trivial counterexample $\Phi_2(z) = z+1$. By (2.7), we have that lemma 4 applies to $\Psi_n(z)$ for odd n, or any product of the form

$$\Psi_{n_1}(z) \cdot \Psi_{n_2}(z) \cdots \Psi_{n_k}(z), \qquad (3.13)$$

where $n_1, n_2, \ldots, n_k$ are all odd.

3.1 Improving the sparse power series method by further truncating degree

The sparse power series algorithm slows appreciably as we calculate $\Phi_n(z)$ for n with increasingly many factors. The slowdown in computing $\Phi_{np}(z)$ compared to $\Phi_n(z)$ is twofold. By introducing a new prime factor p we double the number of subterms $(1 - z^d)$ in our quotient (2.4). In addition, the degree of $\Phi_{np}(z)$ is $p-1$ times that of $\Phi_n(z)$, and thus the algorithm exhibits poorer locality for $\Phi_{np}(z)$ than $\Phi_n(z)$.

In procedure SPS, we effectively compute 2^k distinct power series, each a quotient of subterms $1 - z^d$, each truncated to degree $\frac{\phi(n)}{2}$. We can improve the SPS algorithm if we truncate any intermediate power series to the minimal degree necessary, thereby reducing the number of arithmetic operations and leveraging locality where possible. Towards that end, we let $n = mp$, where p is the largest prime divisor of n and $m > 1$. In which case

$$\begin{aligned} \Phi_{mp}(z) &= \frac{\Phi_m(z^p)}{\Phi_m(z)} \text{ by lemma 1,} \\ &= -\Psi_m(z) \cdot \Phi_m(z^p) \cdot \frac{1}{1-z^m}. \end{aligned} \qquad (3.14)$$

By (2.4) and (2.6), we can rewrite equation (3.14) as:

$$\begin{aligned} &\Phi_n(z) = \\ &\left(\prod_{d|m, d<m} (1-z^d)^{-\mu(\frac{m}{d})}\right) \cdot \left(\prod_{d|m} (1-z^{dp})^{\mu(\frac{m}{d})}\right) \cdot \frac{1}{(1-z^m)}, \\ &= \left(\prod_{d|m, d<m} (1-z^d)^{\mu(\frac{n}{d})}\right) \cdot \left(\prod_{d|m} (1-z^{dp})^{\mu(\frac{n}{dp})}\right) \cdot \frac{1}{(1-z^m)}. \end{aligned} \qquad (3.15)$$

Thus to compute $\Phi_n(z)$, we can compute $\Psi_m(z)$, the leftmost product of (3.15) to degree $\frac{m-\phi(m)}{2}$, use the antipalindromic property of $\Psi_m(z)$ to reconstruct its remaining coefficients, and then multiple or divide the remaining subterms $(1 - z^d)$ as we would in algorithm SPS. Algorithm SPS2 describes the method.

This new procedure does less work to handle subterms $1-z^d$ appearing in $\Psi_m(z)$, as we truncate to degree $\frac{m-\phi(m)}{2}$ as opposed to $\frac{\phi(n)}{2}$.

3.2 Calculating $\Phi_n(z)$ by way of a product of inverse cyclotomic polynomials

In algorithm SPS2 we reduce the degree to which we compute the first $2^{k-1}-1$ intermediate power series of procedure SPS. We are able to further bound the degree to which we must compute any intermediate power series of algorithm SPS2. To that end we establish the next identity. Let $n = p_1 p_2 \cdots p_k$, a product of k distinct odd primes. For $1 \le i \le k$, let $m_i = p_1 p_2 \cdots p_{i-1}$ and $e_i = p_{i+1} \cdots p_k$. We set $m_1 = e_k = 1$, and let $e_0 = n$. Note that $n = e_i p_i n_i$ for $1 \le i \le k$. In addition, $e_{i-1} = p_i e_i$ and $m_{i+1} = m_i p_i$. Then by repeated application of lemma 1, we have

Procedure SPS2(n) : The first revision of the **SPS** algorithm

The Improved Sparse Power Series (SPS) Algorithm

Input: $n = mp$, a squarefree, odd integer with greatest prime divisor p
Output: $a(0), \ldots, a(\frac{\phi(n)}{2})$, the first half of the coefficients of $\Phi_n(z)$

```
//Compute first half of Ψ_m(z)
```
$a(0), a(1), \ldots, a(\lfloor \frac{n-\phi(m)}{2} \rfloor) \longleftarrow$ SPS-Psi(m)

```
//Construct other half of Ψ_m(z) using lemma 3
```
$D \longleftarrow \max(m - \phi(m), \frac{\phi(n)}{2})$
for $i = \lceil \frac{m-\phi(m)}{2} \rceil$ **to** D **do** $a(i) \longleftarrow -a(m - \phi(m) - i)$

```
//Multiply by Φ_m(z^p)
```
$D \longleftarrow \frac{\phi(n)}{2}$
$a(m - \phi(m) + 1), a(m - \phi(m) + 2), \ldots, a(D) \longleftarrow 0$
for $d|m$ *such that* $0 < d < m$ **do**
  if $\mu(\frac{n}{d}) = -1$ **then**
    for $i = D$ **down to** dp **by** -1 **do**
    $a(i) \longleftarrow a(i) - a(i - dp)$
  else
    for $i = dp$ **to** D **do** $a(i) \longleftarrow a(i) + a(i - dp)$

```
//Divide by 1 − z^m
```
for $i = m$ **to** D **do** $a(i) \longleftarrow a(i) + a(i - m)$
return $a(0), a(1), \ldots a(D)$

$$\begin{aligned}\Phi_n(z) &= \frac{\Psi_{m_k}(z^{e_k})}{1 - z^{n/p_k}} \Phi_{m_k}(z^{e_{k-1}}) \\ &= \frac{\Psi_{m_k}(z^{e_k})}{(1 - z^{n/p_k})} \frac{\Psi_{m_{k-1}}(z^{e_{k-1}})}{(1 - z^{n/p_{k-1}})} \Phi_{m_{k-1}}(z^{e_{k-2}}) \\ &\cdots \\ &= \frac{\Psi_{m_k}(z^{e_k})}{(1 - z^{n/p_k})} \cdots \frac{\Psi_{m_2}(z^{e_2})}{1 - z^{n/p_2}} \frac{\Psi_{m_1}(z^{e_1})}{1 - z^{n/p_1}} \Phi_1(z^{e_0}) \\ &= \prod_{j=1}^{k} \frac{\Psi_{m_j}(z^{e_j})}{1 - z^{n/p_j}} \cdot (1 - z^n).\end{aligned} \tag{3.16}$$

As $\Psi_{m_1}(z^{e_1}) = \Psi_1(z^{e_1}) = 1$, we have

$$\begin{aligned}\Phi_n(z) &= \prod_{j=1}^{k} \Psi_{m_j}(z^{e_j}) \cdot \prod_{j=1}^{k} (1 - z^{n/p_j})^{-1} \cdot (1 - z^n) \\ &= \prod_{j=2}^{k} \Psi_{m_j}(z^{e_j}) \cdot \prod_{j=1}^{k} (1 - z^{n/p_j})^{-1} \bmod (z^{\phi(n)}).\end{aligned} \tag{3.17}$$

For example, for $n = 105 = 3 \cdot 5 \cdot 7$,

$$\Phi_{105}(z) = \Psi_{15}(z)\Psi_3(z^7) \cdot (1 - z^{15})^{-1}(1 - z^{21})^{-1}(1 - z^{35})^{-1} \cdot (1 - z^{105}) \tag{3.18}$$

As with algorithm SPS2, we first calculate half the terms of $\Psi_{m_k}(z^{e_k}) = \Phi_{p_1 p_2 \cdots p_{k-1}}(z)$, those with degree at most $\lfloor \frac{\phi(m_k)}{2} \rfloor$. We then iteratively compute the product

$$\Psi_{m_k}(z^{e_k}) \cdots \Psi_{m_2}(z^{e_2}). \tag{3.19}$$

When calculating the degree of $\Psi_{m_j}(z^{e_j})$ we truncate to degree at most

$$\left\lfloor \frac{1}{2} \prod_{i=j}^{k} \phi(m_i) e_i \right\rfloor, \tag{3.20}$$

half the degree of the product in (3.19). As our intermediate product grows larger we have to truncate to larger degree. The term $\Psi_{m_i}(z^{e_i})$, comprises $2^{i-1} - 1$ subterms of the form $1 - z^d$. We compute $\Psi_{m_k}(z^{e_k})$ first because that contains nearly half of the $2^k - 1$ subterms we must handle to obtain $\Phi_n(z)$, so it is best that we not have to truncate to a comparatively large degree for these terms.

Note that the degree of the product in (3.19) is, by (3.17),

$$\phi(n) - \sum_{p|n} n/p. \tag{3.21}$$

Thus (3.19) potentially has degree greater than that of $\Phi_n(z)$, provided

$$1/p_1 + 1/p_2 + \cdots + 1/p_k > 1. \tag{3.22}$$

Thus at some point we are forced to truncate to degree $\frac{\phi(n)}{2}$. For $n = p_1 p_2 \cdots p_k$ for which $\Phi_n(z)$ is presently feasible to compute, however, it is seldom the case that (3.22) holds. The smallest odd, squarefree n for which (3.22) holds is $n = 3,234,846,615$, the product of the first nine odd primes. In any case, we still truncate to a lower degree than in procedure SPS (or SPS2) when calculating $\Psi_{m_i}(z^{e_i})$ for

$$k - 8 < i \leq k$$
$$(k - 8 < i < k, \text{respectively}).$$

. As $\Psi_{m_k}(z^{e_k}) \cdots \Psi_{m_{k-7}}(z^{e_{k-7}})$ comprise $2^k - 2^{k-8} - 8$ of the $2^k - 1$ subterms, so we expect there to be some speed-up compared to procedures SPS and SPS2 regardless. This is consistent with our timings (see section 4).

Let

$$\begin{aligned}f(z) &= \sum a(i) z^i = \Psi_{m_k}(z^{e_k}) \cdots \Psi_{m_j}(z^{e_j}) \pmod{z^{\lfloor D/2 \rfloor + 1}}, \\ g(z) &= f(z) \cdot \Psi_{m_{j+1}}(z^{e_{j+1}}),\end{aligned} \tag{3.23}$$

and let D and D_g be the degrees of f and g, respectively. Suppose we have $a(0), a(1), \ldots, a(\lfloor D/2 \rfloor)$, the terms of f truncated to degree $\lfloor D/2 \rfloor$, and we want to compute the terms of the product $g = f(z) \cdot \Psi_{m_{j+1}}(z^{e_{j+1}})$ to as minimal degree as necessary with the aim of eventually computing $\Phi_n(z)$. We need not immediately compute terms of g with degree greater than $\lfloor \frac{D_g}{2} \rfloor$, as we can use lemma 4 to retrieve higher-degree terms of g. In addition, for our purposes of computing $\Phi_n(z)$ we need not consider terms of degree greater than $\frac{\phi(n)}{2}$. In tandem with the fact that $a(i) = 0$ for $i > D$, we need to retrieve $a(i)$ for i at most $\min(D, \lfloor \frac{D_g}{2} \rfloor, \frac{\phi(n)}{2})$. We do this repeatedly while constructing the product

Once we have the product (3.19), we divide by $1 - z^{n/p_j}$ for $1 \leq j \leq k$. We detail this approach in procedure SPS3.

Procedure SPS3(n) : The second revision of the **SPS** algorithm

The Iterative Sparse Power Series (SPS) Algorithm

Input: $n = p_1 p_2 \dots p_k$, a squarefree, odd integer with prime divisors $p_1 < p_2 < \cdots < p_k$
Output: $a(0), \dots, a(\frac{\phi(n)}{2})$, the first half of the coefficients of $\Phi_n(z)$

$m_1 \longleftarrow 1, e_k \longleftarrow 1$
for $j = 1$ **to** k **do**
  $m_{j+1} \longleftarrow m_j \cdot p_j$
  $e_{k-j} \longleftarrow e_{k-j+1} \cdot p_{k-j+1}$
//D is the degree of our current polynomial f
//D_g is the degree of the next polynomial g
//D_Φ is the degree to which we truncate our final result
$D \longleftarrow 0$, $D_g \longleftarrow \phi(m_k)$, $D_\Phi \longleftarrow \frac{\phi(n)}{2}$
$a(0), a(1), a(2), \dots, a(D_\Phi) \longleftarrow 1, 0, 0, \dots, 0$
for $j = k$ **down to** 2 **do**
  $D \longleftarrow D_g$
  //Multiply by $\Phi_{m_j}(z^{e_j})$:
  for $d | m_j$ *such that* $0 < d < m_j$ **do**
    if $\mu(\frac{n}{d}) = 1$ **then**
      for $i = \min(\lfloor \frac{D}{2} \rfloor, D_\Phi)$ **down to** de_j **by** -1 **do**
        $a(i) \longleftarrow a(i) - a(i - de_j)$
    else
      for $i = de_j$ **to** $\min(\lfloor \frac{D}{2} \rfloor, D_\Phi)$ **do**
        $a(i) \longleftarrow a(i) + a(i - de_j)$
  //Use lemma 4 to construct terms of higher degree:
  $D_g \longleftarrow D + \phi(m_{j-1}) e_{m_{j-1}}$
  if $D \equiv 0 \pmod 2$ **then**
    for $i = \frac{D}{2} + 1$ **to** $\min(D, \frac{D_g}{2}, D_\Phi)$ **do**
      $a(i) \longleftarrow a(D - i)$
  else
    for $i = \frac{D+1}{2}$ **to** $\min(D, \frac{D_g}{2}, D_\Phi)$ **do**
      $a(i) \longleftarrow a(D - i)$
//Divide by $(1 - z^{n/p_j})$:
for $j = 1$ **to** k **do**
  $d \longleftarrow \frac{n}{p_j}$
  for $i = d$ **to** D_Φ **do** $a(i) \longleftarrow a(i - d)$
return $a(0), a(1), \dots, a(D_\Phi)$

3.3 Calculating $\Phi_n(z)$ and $\Psi_n(z)$ recursively.

Algorithm SPS3 depended on the identity (3.17), which describes $\Phi_n(z)$ in terms of a product of inverse cyclotomic polynomials of decreasing order and index. We derive an analog for $\Psi_n(z)$. Let m_i and e_i be as defined in section 3.2, and again let $n = p_1 p_2 \cdots p_k$ be a product of k distinct odd primes where $p_1 < p < 2 < \dots p_k$.

Then by repeated application of lemma 1:

$$\begin{aligned}\Psi_n(z) &= \Phi_{m_k}(z^{e_k})\Psi_{m_k}(z^{e_{k-1}}) \\ &= \Phi_{m_k}(z^{e_k})\Psi_{m_{k-1}}(z^{e_{k-1}})\Phi_{m_{k-1}}(z^{e_{k-2}}) \\ &\dots \\ &= \Phi_{m_k}(z^{e_k}) \cdots \Phi_{m_1}(z^{e_1})\Psi_{m_1}(z^{e_1}).\end{aligned} \tag{3.24}$$

As $m_1 = 1$ and $\Psi_1(z) = 1$, we thus have that

$$\Psi_n(z) = \prod_{j=1}^{k} \Phi_{m_j}(z^{e_j}). \tag{3.25}$$

(3.17) and (3.25) suggest a recursive method of computing $\Phi_n(z)$. For purposes of simplicity we will not leverage the palindromic properties. Consider the example of $\Phi_{105}(z)$ from section 3.2 (equation (3.18)). To calculate $\Phi_{105}(z)$ via algorithm SPS3, we would calculate the product in (3.18) from left to right. However, in light of (3.25), we can treat $\Psi_{m_i}(z^{e_i})$ as a product of cyclotomic polynomials of smaller index:

$$\begin{aligned}\Psi_{15}(z) &= \Phi_5(z)\Phi_1(z^3), \\ \Psi_3(z^7) &= \Phi_1(z^7);\end{aligned} \tag{3.26}$$

and one can use (3.17) yet again and apply it to $\Phi_5(z)$:

$$\Phi_5(z) = (1 - z^1)^{-1} \cdot (1 - z^5). \tag{3.27}$$

We compute $\Phi_n(z)$ by recursing into the factors of n.

As we recurse we can potentially lower the degree up to which we must compute terms. Suppose, again, that we have some product of cyclotomic polynomials $f(z)$, truncated to degree $\frac{D_{max}}{2}$, and we want to compute the terms of

$$g(z) = f(z) \cdot \Psi_m(z^e), \tag{3.28}$$

up to degree $\frac{D_{max}}{2}$. By (3.25) we can express $\Psi_m(z^e)$ as a product of cyclotomic polynomials of descending order. Suppose that $D_g < D_{max}$. Then as we multiply by the subterms of $\Psi_m(z^e)$, we need only truncate to degree $\frac{D_g}{2}$, after which we can apply lemma 4 to retrieve higher-degree terms of $g(z)$. Procedure SPS4 describes this method.

To calculate the first half of the coefficients of $\Phi_n(z)$, one would merely set

$$a(0) = 1, a(i) = 0 \text{ for } 0 < i \le \phi(n), \tag{3.29}$$

and call SPS4(n,1,true,0,$\phi(n), a$). Similarly, to calculate the first half of $\Psi_n(z)$ we would call SPS4(n,1,false,0,$n - \phi(n)$,a).

4. PERFORMANCE AND TIMINGS

We timed our implementations on a system with a 2.67GHz Intel Core i7 quad-core processor and 6 GB of memory. All of our aglorithms are implemented in C and are single-threaded. Here we time our 64-bit precision implementations of procedures SPS1-4, each of which check for integer overflow using inline assembly.

Our implementation of algorithm 1 calculates $\Phi_n(z)$ modulo two 32-bit primes and reconstructs $\Phi_n(z)$ by Chinese remaindering.

As the number of distinct prime factors of n plays a significant role in the cost of computing $\Phi_n(z)$, we list the factors of n (table 2) and $A(n)$ (table 3) for n appearing in table 1.

For the SPS and SPS4 algorithms, we have implemented, in addition to the 64-bit version, 8-bit, 32-bit, and 128-bit

Procedure SPS4(m, e, λ, D, D_{max}, a) : Multiply by $\Phi_m(z^e)$ or $\Psi_m(z^e)$

A recursive algorithm to multiply by $\Phi_m(z)$ or $\Psi_m(z)$

Input:

- m, a positive, squarefree odd integer
- λ, a boolean
- a, an array of integers $a(0), a(1), \ldots$ satisfying $f(z) \equiv \sum a(i)z^i \pmod{z^{\lfloor \frac{D_{max}}{2} \rfloor + 1}}$ modulo $z^{\lfloor \frac{1}{2} D_{max} \rfloor} + 1$, where $f(z)$ is some product of cyclotomic polynomials
- D, the degree of $f(z)$
- D_{max}, a bound on the degree

Output:
D^*, the degree of the resulting polynomial. If λ is true, we compute $g(z) = f(z)\Phi_m(z^e)$, truncated to degree $\frac{1}{2}D_{max}$. Otherwise, we compute $g(z) = f(z)\Psi_m(z^e)$, truncated to degree $\frac{1}{2}D_{max}$. We write the coefficients of g to array a, and return the degree of g, D_g.

if λ **then** $D_g \longleftarrow D + \phi(m)e$ **else**
$D_g \longleftarrow D + (m - \phi(m))e$
`//`D^*_{max} `is our new degree bound`
$D^*_{max} \longleftarrow \min(D_g, D_{max})$

$e^* \longleftarrow e$, $m^* \longleftarrow m$, $D^* \longleftarrow D$
while $m^* > 1$ **do**
  $p \longleftarrow$ largest prime divisor of m^*
  $m^* \longleftarrow \frac{m}{p}$
  `//multiply by` $\Phi_{m^*}(z^{e^*})$ `(or` $\Psi_{m^*}(z^{e^*})$`)`
  $D^* \longleftarrow$ SPS4(m^*, e^*, not λ, D^*, D^*_{max}, a)
  $e^* \longleftarrow e^* p$
if λ **then**
  for *each prime* $p|m$ **do**
    $d \longleftarrow \frac{m \cdot e}{p}$ `//Divide by` $1 - z^{m \cdot e/p}$
    for $i = d$ **to** $\lfloor \frac{D^*_{max}}{2} \rfloor$ **do** $a(i) \longleftarrow a(i - d)$
  $d \longleftarrow m \cdot e$ `//Multiply by` $1 - z^{m \cdot e}$
  for $i = \lfloor \frac{D^*_{max}}{2} \rfloor$ **down to** d **do** $a(i) \longleftarrow a(i - d)$

`//Get higher-degree terms of` $g(z)$ `as necessary`
if $D_g \equiv 0 \bmod 2$ **then**
  for $i = \lfloor \frac{D_g}{2} \rfloor + 1$ **to** $\min(D_g, \frac{D_{max}}{2})$ **do**
  $a(i) \longleftarrow a(D_g - i)$
else
  for $i = \lfloor \frac{D_g}{2} \rfloor + 1$ **to** $\min(D_g, \frac{D_{max}}{2})$ **do**
  $a(i) \longleftarrow -a(D_g - i)$
return D_g

precision versions. We also have a version of SPS and SPS4 which calculates images of $\Phi_n(z)$ modulo 32-bit primes, writes the images to the harddisk, and then reconstruct $\Phi_n(z)$ from the images by way of Chinese remaindering. This implementation is most useful for $\Phi_n(z)$ which we cannot otherwise fit in main memory. We do not use GMP multiprecision integer arithmetic; it is too slow for our specfic purposes.

Table 1: Time to calculate $\Phi_n(z)$ (in seconds*)

	algorithm				
n	FFT	SPS	SPS2	SPS3	SPS4
255255	0.40	0.00	0.00	0.00	0.00
1181895	1.76	0.01	0.00	0.00	0.00
4849845	7.74	0.12	0.06	0.02	0.01
37182145	142.37	1.75	0.95	0.23	0.19
43730115	140.62	1.69	0.93	0.23	0.19
111546435	295.19	6.94	3.88	1.45	0.94
1078282205	-	105.61	58.25	12.34	9.29
3234846615	-	432.28	244.44	81.32	49.18

*times are rounded to the nearest hundredth of a second

Table 2: Factorization of n, for n from table 1

n	factorization of n
255255	$3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17$
1181895	$3 \cdot 5 \cdot 11 \cdot 13 \cdot 19 \cdot 29$
4849845	$3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19$
37182145	$5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
43730115	$3 \cdot 5 \cdot 11 \cdot 13 \cdot 19 \cdot 29 \cdot 37$
111546435	$3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23$
1078282205	$5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29$
3234846615	$3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29$

Table 3: $A(n)$ for n from table 1

n	height $A(n)$
255255	532
1181895	14102773
4849845	669606
37182145	2286541988726
43730115	862550638890874931
111546435	1558645698271916
1078282205	8161018310
3234846615	2888582082500892851

5. CURRENT WORK

We have implemented the algorithms in this paper to create a library of data on the heights and lengths of cyclotomic polynomials. This data is available at

`http://www.cecm.sfu.ca/~ada26/cyclotomic/`

A 64-bit implementation of the SPS4 algorithm, written in C but without overflow check, is also made available at the website.

We aim to compute the coefficients of $\Phi_n(z)$, for

$$n = 3 \cdot 5 \cdot 11 \cdot 13 \cdot 19 \cdot 29 \cdot 37 \cdot 43 \cdot 53 = 99660932085.$$

We expect that this cyclotomic polynomial will have very large height. We have previously verified that

$A(\frac{n}{53}) = 64540997036010911566826446181523888971563$ and
$A(\frac{n}{43}) = 67075962666923019823602030663153118803367$

are the smallest two examples of k such that $A(k) > k^4$. Both $A(\frac{n}{53})$ and $A(\frac{n}{43})$ are greater than 2^{135}.

We previously attempted to compute $\Phi_n(z)$ using an implementation of the SPS algorithm. We computed images of $\Phi_n(z)$ modulo 32-bit primes. Storing half of $\Phi_n(z)$ to 32-bit precision takes roughly 76 GB of space. We do not have enough RAM to store these images in main memory,

so we read and wrote intermediate results to the hard disk. This proved to be slow, as each image required us to make $2^9 = 512$ passes over the hard disk. We computed four images of $\Phi_n(z)$, after which the hard disk crashed.

In light of the development of the new variants of the SPS algorithms, we have a new approach to compute $\Phi_n(z)$. We want to minimize hard disk reads and writes. This is because performing the computation on the harddisk is appreciably slower and potentially more error-prone than in memory. We are limited to 16 GB of RAM. We expect that $A(n) < 2^{320}$; that is, 320-bit precision will be sufficient to construct $\Phi_n(z)$. Towards our aim, let

$$f(z) = \Psi_{m_9}(z)\Psi_{m_8}(z^{53}). \tag{5.1}$$

where

$$m_9 = \tfrac{n}{53} = 1,880,394,945 \text{ and}$$
$$m_8 = \tfrac{n}{43\cdot 53} = 43,730,115.$$

By (3.16), we have

$$\Phi_n(z) = f(z)(1-z^{n/53})^{-1}(1-z^{n/43})^{-1}\Phi_{m_8}(z^{43\cdot 53}). \tag{5.2}$$

$f(z)$ has degree less than $2.55 \cdot 10^9$. We can compute images of $f(z)$ modulo 64-bit primes using roughly 10 GB of RAM, then extract $f(z)$ from its images by way of Chinese remaindering. After which we will compute the coefficients of the truncated power series

$$\begin{aligned} g(z) &\equiv f(z)(1-z^{n/53})^{-1}(1-z^{n/43})^{-1} \pmod{z^{\frac{\phi(n)}{2}+1}} \\ &= \sum_{i=0}^{\frac{\phi(n)}{2}} c_i z^i. \end{aligned} \tag{5.3}$$

This will entail two passes over the hard disk, one per division by subterms $1 - z^{n/53}$ and $1 - z^{n/43}$. We produce the coefficients of $g(z)$ in order of ascending degree during the second pass of the harddisk. Storing $g(z)$ or $\Phi_n(z)$ at this precision up to degree $\frac{\phi(n)}{2}$ requires more than 750 GB of storage. We can reorganize the terms of $g(z)$ in a manner which allows us to compute the coefficients of $\Phi_n(z)$ in memory. For $0 \le j < 43 \cdot 53 = 2279$, let

$$g_j(z) = \sum_{0 \le i \cdot 2279 + j \le \frac{\phi(n)}{2}} c_i z^i \tag{5.4}$$

We can construct the $g_j(z)$ as we sequentially produce the terms of $g(z)$. We have that

$$g(z) = \sum_{j=0}^{2278} z^j \cdot g_j(z^{2279}), \tag{5.5}$$

and thus by (5.2),

$$\Phi_n(z) \equiv \sum_{j=0}^{2278} z^j \cdot g_j(z^{2279})\Phi_{m_8}(z^{2279}) \pmod{z^{\frac{\phi(n)}{2}+1}}. \tag{5.6}$$

Thus to produce the first half of the coefficients of $\Phi_n(z)$, it suffices to compute $g_j(z) \cdot \Phi_{m_8}(z)$, for $0 \le j < 2279$. Each of these polynomials has degree less than $2.6 \cdot 10^6$, and be computed to 320-bit precision with less than a GB of memory.

Figure 1: The coefficients of $\Phi_n(z) = \sum a(k)z^k$, for $n = 43730115$.

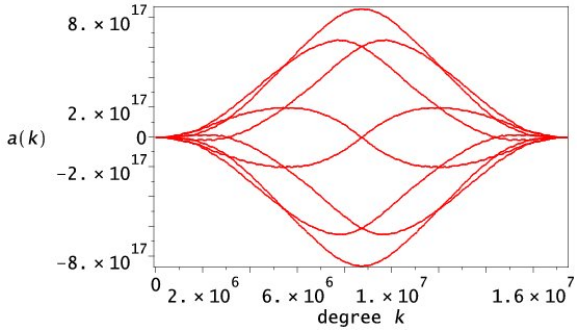
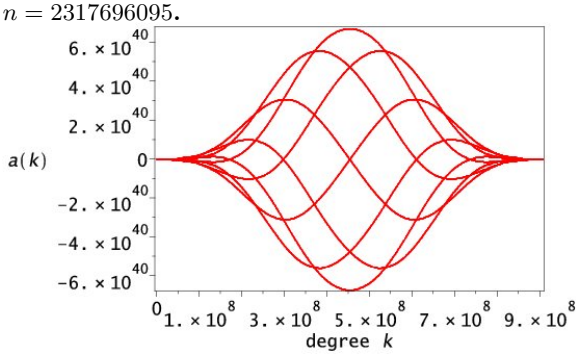


Figure 2: The coefficients of $\Phi_n(z) = \sum a(k)z^k$, for $n = 2317696095$.



We are also interested in the behaviour of the coefficients of terms in order of ascending degree. From figures 1 and 2, we see that cyclotomic polynomials coefficients can exhibit interesting structure.

If this computation of $\Phi_n(z)$ proves to be reasonably fast, we will also attempt to compute $A(n)$, for

$$n = 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 \cdot 31 = 100280245065,$$

the product of the first ten odd primes, by a similar approach.

6. REFERENCES

[1] A. Arnold and M. Monagan. Calculating cyclotomic polynomials. Submitted to *Mathematics of Computation*, available at `http://www.cecm.sfu.ca/~ada26/cyclotomic/`.

[2] P. Erdős and R.C. Vaughn. On the coefficients of the cyclotomic polynomial. *Bull. Amer. Math. Soc.*, 52:179–184, 1946.

[3] K.O. Geddes, S.R. Czapor, and G. Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, Boston, 1992.

[4] Y. Koshiba. On the calculations of the coefficients of the cyclotomic polynomials. *Rep. Fac. Sci. Kagoshima Univ.*, (31):31–44, 1998.

[5] Y. Koshiba. On the calculations of the coefficients of the cyclotomic polynomials. II. *Rep. Fac. Sci. Kagoshima Univ.*, (33):55–59, 2000.

[6] T.D. Noe. Personal communication.