

# How to Compute the Coefficients of the Elliptic Modular Function $j(z)$

Harald Baier  
Darmstadt Center of IT-Security  
Darmstadt University of Technology  
D - 64283 Darmstadt  
hbaier@dzi.tu-darmstadt.de

Günter Köhler  
Department of Mathematics  
University of Würzburg  
D - 97074 Würzburg  
koehler@mathematik.uni-wuerzburg.de

May 21, 2003

## Abstract

We discuss various methods to compute the Fourier coefficients of the elliptic modular function  $j(z)$ . We present run times to compute the coefficients in practice. If possible we discuss the theoretical complexity of the corresponding method, too. We conclude that in practice an approach due to Kaneko and Zagier turns out to be most efficient.

## 1 Introduction

The Fourier coefficients  $c(n)$  of the elliptic modular function

$$j(z) = \sum_{n=-1}^{\infty} c(n)q^n, \quad (1)$$

where  $q = e(z) = e^{2\pi iz}$  for  $z$  in the upper half plane, are important for several purposes. For instance, they can be used to compute singular values of  $j(z)$  and Hilbert class polynomials, and they are needed to compute modular equations. The standard definition of  $j(z)$  is

$$j(z) = \frac{E_4^3(z)}{\Delta(z)}, \quad (2)$$

where

$$\Delta(z) = q \cdot \prod_{n=1}^{\infty} (1 - q^n)^{24} = 12^{-3} \cdot (E_4^3(z) - E_6^2(z)) \quad (3)$$

is the discriminant function and

$$E_4(z) = 1 + 240 \cdot \sum_{n=1}^{\infty} \sigma_3(n)q^n, \quad E_6(z) = 1 - 504 \cdot \sum_{n=1}^{\infty} \sigma_5(n)q^n \quad (4)$$

with  $\sigma_r(n) = \sum_{d|n} d^r$  are the Eisenstein series of weights 4 and 6, respectively.

We discuss various approaches to compute the Fourier coefficients  $c(n)$ . We show that the respective performance in practice is very different. Furthermore, we give evidence that a method proposed by Zagier ([Zagier, 1996]) and Kaneko ([Kaneko, 1999]) is the most efficient one. For most of the methods discussed in this paper we present the performance in both theory and practice. More precisely, by the performance in theory we mean the number of multiplications of integers. Thus we do not consider the contribution of integer additions. If we speak of the performance in practice, we mean the CPU time of our implementation.

One of our aims is that our results are easily verifiable by an 'ordinary' user. Hence all our run times are measured on a PC using freely available libraries. Our practical tests are performed on an Athlon XP1600+ running Linux 2.4.10 at 1.4 GHz and having 1 GByte main memory. All programs are implemented in C++ using the GNU compiler gcc 3.0.1 and the GNU multiprecision package gmp 3.2.1 . As stated above, all software is freely available. The implementation of the methods in this paper may be downloaded from the WWW-site of the first author (<http://www.cdc.informatik.tu-darmstadt.de/~hbaier>).

The rest of the paper is organized as follows: First, in Section 2, we discuss two algorithms using the definition of the modular function  $j$ . Second, in Section 3 we present two methods proposed by Rademacher [Rademacher, 1938] and Mahler [Mahler, 1976], respectively. Next, in Section 4 we investigate an algorithm which makes use of ideas due to Herrmann [Herrmann, 1973]. In Section 5 we present two methods due to the second author. This approach uses Hecke series. The last method due to Zagier and Kaneko is discussed in Section 6.

## 2 Two Approaches Using the Definition of $j$

In this section we discuss the performance of our first two algorithms. Both methods base on the defining Equations (2), (3), and (4). We give evidence that for our purposes none of them turns to be efficient in practice.

In order to make use of the defining equations, we have to know the coefficients of the denominator in Equation (2). It is well known that  $\Delta$  may be written in terms of a Fourier series, that is we have

$$\Delta(z) = \sum_{n=1}^{\infty} \tau(n)q^n . \quad (5)$$

The coefficients  $\tau(n)$  in Equation (5) are called *Ramanujan numbers*. A first approach would be to compute  $\tau(n)$  by means of the infinite product for  $\Delta(z)$  or by means of  $E_4^3(z)$  and  $E_6^2(z)$ , respectively. However, in the first author's thesis it is shown that both ideas turn out to be rather slow ([Baier, 2002]).

We investigate two different representations of the Ramanujan numbers in what follows. The first one is due to Ramanujan himself ([Ramanujan, 1927]), the second one was proposed by Niebur ([Niebur, 1975], [Gouvêa, 1997]).

In 1916 Ramanujan ([Ramanujan, 1927], p. 152) proved the recursion formula

$$\tau(1) = 1, \quad \tau(n) = -\frac{24}{n-1} \cdot \sum_{k=1}^{n-1} \sigma_1(n-k)\tau(k) \quad \text{for } n > 1. \quad (6)$$

Once the values  $\sigma_1(n)$  are known, it is obvious how to evaluate and implement Equation (6) for  $n \geq 2$ . First, we are not aware of any reasonable estimation of the computational complexity to compute  $\sigma_1(n)$ . However, if we use trial division and reasonable values of  $n$ , say  $n \leq 50000$ , we may use machine types and hence fast arithmetic for the computation of  $\sigma_1(n)$ . Indeed, as explained below, our practical test shows that only about 5% of the CPU time is spent to compute the values  $\sigma_1(n)$ .

Second, we assume that the division by  $n-1$  in Equation (6) takes the same time as a multiplication. Let  $R(N)$  denote the total number of multiplications to compute  $\tau(n)$  for  $2 \leq n \leq N$  using Ramanujan's recursion formulae. Then it is easy to see that

$$R(N) = \sum_{n=2}^N (n+1) = \frac{N^2}{2} + \frac{3N}{2} - 2. \quad (7)$$

Finally, if we set  $N = 50000$ , this approach takes us 6 minutes, 15 seconds in practice to compute  $\tau(n)$  for  $2 \leq n \leq N$ . The CPU time to get all relevant values  $\sigma_1(n)$  is 19 seconds.

Let us turn to the second method of this section. It is due to Niebur ([Niebur, 1975], [Gouvêa, 1997]). Niebur shows

$$\tau(n) = n^4\sigma_1(n) - 24 \cdot \sum_{k=1}^{n-1} (35k^4 - 52k^3n + 18k^2n^2) \sigma_1(k)\sigma_1(n-k). \quad (8)$$

We describe how we evaluate Equation (8). As above, we leave out the cost of the computation of  $\sigma_1(n)$  in our following discussion. Let  $n$  and  $k$  be given. We compute (in this order)  $k^2$ ,  $kn$ ,  $k^4 = k^2 \cdot k^2$ ,  $k^3n = k^2 \cdot kn$ , and  $k^2n^2 = (kn)^2$ . In all we have to perform 10 multiplications to compute an addend in the sum of Equation (8). Thus the number of multiplications to compute  $\tau(n)$  is  $10(n-1) + 4 = 10n - 6$ . Let  $Ni(N)$  be the total amount of multiplications to compute  $\tau(n)$  for  $2 \leq n \leq N$  using Niebur's formula. Then we have

$$Ni(N) = \sum_{n=2}^N (10n - 6) = 5N^2 - N - 4. \quad (9)$$

Compared to  $R(N)$ , the number of multiplications of this method is about 10 times larger. Indeed, the run time in practice is much slower. If we compute  $\tau(n)$  for  $n$  up to 50000, the CPU time is 22 minutes, 21 seconds.

We will see in Section 6, that if we use a method due to Kaneko and Zagier, the whole computation of  $c(n)$  up to  $n = 50000$  takes less than 9 minutes in practice. Thus we skip the further computation of the  $c(n)$  by the methods of this section.

### 3 Approaches due to Rademacher and Mahler

This section deals with two further methods to compute the coefficients  $c(n)$ . The first one is due to Rademacher [Rademacher, 1938], the second one due to Mahler [Mahler, 1976].

H. Rademacher [Rademacher, 1938] used the circle method to prove a formula which expresses  $c(n)$  as a convergent infinite series in terms of Bessel functions and Kloosterman sums. He realized, however, that the convergence of the series is rather slow and that “the coefficients ... can be found [from the formula] by troublesome computations, which for higher  $n$  are practically inexecutable ...”. For this reason we did not check how Rademacher’s formula performs using the computing power now available. A modern account of Rademacher’s and related later work is given in [Knopp, 1990].

We next discuss Mahler’s approach. In [Mahler, 1976], p. 91, K. Mahler proved a system of recursion formulas for  $c(n)$ . They read

$$c(4n) = c(2n + 1) + \frac{1}{2}(c(n)^2 - c(n)) + \sum_{k=1}^{n-1} c(k)c(2n - k), \quad (10)$$

$$\begin{aligned} c(4n + 1) &= c(2n + 3) - c(2)c(2n) + \frac{1}{2}(c(n + 1)^2 - c(n + 1)) \\ &\quad + \frac{1}{2}(c(2n)^2 + c(2n)) + \sum_{k=1}^n c(k)c(2n - k + 2) \\ &\quad - \sum_{k=1}^{2n-1} (-1)^{k-1} c(k)c(4n - k) + \sum_{k=1}^{n-1} c(k)c(4n - 4k), \end{aligned} \quad (11)$$

$$c(4n + 2) = c(2n + 2) + \sum_{k=1}^n c(k)c(2n - k + 1), \quad (12)$$

$$\begin{aligned} c(4n + 3) &= c(2n + 4) - c(2)c(2n + 1) - \frac{1}{2}(c(2n + 1)^2 - c(2n + 1)) \\ &\quad + \sum_{k=1}^{n+1} c(k)c(2n - k + 3) - \sum_{k=1}^{2n} (-1)^{k-1} c(k)c(4n - k + 2) \\ &\quad + \sum_{k=1}^n c(k)c(4n - 4k + 2). \end{aligned} \quad (13)$$

As soon as the values  $c(-1), \dots, c(5)$  are known, the sequence of  $c(n)$  is uniquely determined

by Mahler's recursion formulas.

We next investigate the number of multiplications to evaluate Mahler's equations. Let  $N \in \mathbb{N}$ ,  $4 \mid N$ . By  $M(N)$  we denote the number of multiplications to compute the Fourier coefficients  $c(n)$  up to  $n = N$  by Mahler's approach. We do not consider a factor  $\frac{1}{2}$ , as this operation is only a right shift. We fix  $1 \leq n \leq \frac{N}{4}$ . Obviously Equations (10) - (13) yield a contribution of  $n$ ,  $4n + 1$ ,  $n$ , and  $4n + 3$  multiplications to  $M(N)$ , respectively. Thus for fixed  $n$  the contribution is  $10n + 4$  multiplications. As we assume  $c(-1), \dots, c(5)$  to be known, we conclude

$$M(N) = \sum_{n=1}^{\frac{N}{4}-1} (10n + 4) + \frac{N}{4} - 6 = \frac{5N^2}{16} - 10. \quad (14)$$

In this way, a table of  $c(n)$  for  $n \leq 50000$  was computed in the first author's thesis [Baier, 2002]. The run time in [Baier, 2002] compared to the method of Section 6 is much larger. Although both hardware and libraries in use of [Baier, 2002] are inferior to our environment, we expect an implementation of Mahler's Equations (10) - (13) to be inferior to the method of Kaneko on our platform, too.

## 4 Herrmann's Method

We next present two methods for the computation of the  $c(n)$  which both base on an article of O. Herrmann [Herrmann, 1973]. The first method is due to Herrmann himself. In an early work in the field of algorithmic number theory Herrmann computed a table of  $c(n)$  for  $n \leq 6002$  as explained below. The second approach is our variant of Herrmann's algorithm. It turns out that our algorithm is slightly faster in practice than the original method.

The crucial observation is that we may write the discriminant function  $\Delta$  in terms of Dedekind's  $\eta$ -function. More precisely, we have  $\Delta = \eta^{24}$ , where

$$\eta(z) = e\left(\frac{z}{24}\right) \cdot \prod_{n=1}^{\infty} (1 - q^n) = \sum_{n=1}^{\infty} \left(\frac{12}{n}\right) \cdot e\left(\frac{n^2 z}{24}\right) = e\left(\frac{z}{24}\right) \cdot \sum_{n=-\infty}^{\infty} (-1)^n q^{n(3n+1)/2}, \quad (15)$$

where  $\left(\frac{12}{n}\right)$  is a quadratic residue symbol. Herrmann ([Herrmann, 1973]) used Equation (15) to compute the values of  $c(n)$  for  $n \leq 6002$  as follows. He avoided the computation of the power  $E_4^3$  in Equation (2) by means of the identity

$$E_4^3 = E_{12} + \frac{432000}{691} \Delta, \quad (16)$$

where

$$E_{12}(z) = 1 + \frac{65520}{691} \cdot \sum_{n=1}^{\infty} \sigma_{11}(n) q^n \quad (17)$$

is the normalized Eisenstein series of weight 12. Then he divided  $E_{12}(z)$  repeatedly 24 times by  $\eta(z)$ . This works well since Euler's series  $\sum_{n=-\infty}^{\infty} (-1)^n q^{n(3n+1)/2}$  is sparse with very few coefficients  $\pm 1$  and all others equal to 0.

In order to implement Herrmann's proposal, we mention the following observations. First, using the relation  $\Delta = \eta^{24}$  and Equations (2), (16), we get  $(j - \frac{432000}{691}) \cdot \eta^{24} = E_{12}$ . Second, it is obvious that we may write the right sum in Equation (15) as  $\sum_{n=0}^{\infty} e(n)q^n$  with  $e(n) \in \{-1; 0; 1\}$ . Thus we get

$$\left( c(-1) + \left( c(0) - \frac{432000}{691} \right) q + \sum_{n=1}^{\infty} c(n)q^{n+1} \right) \cdot \left( \sum_{n=0}^{\infty} e(n)q^n \right)^{24} = 1 + \frac{65520}{691} \cdot \sum_{n=1}^{\infty} \sigma_{11}(n)q^n . \quad (18)$$

Once the coefficients  $e(n)$  and  $\sigma_{11}(n)$  are known Equation (18) shows how to recover the Fourier coefficients of the modular function  $j$ . The computation of  $\sigma_{11}(n)$  is straightforward. In addition the computation of the coefficients  $e(n)$  is very fast. An algorithm may be found for instance as Algorithm 7.3 in the first author's thesis [Baier, 2002]. We remark that in our implementation we multiply both sides in Equation (18) by 691 to work with integers.

We estimate the number of multiplications to get the  $c(n)$  up to  $n = N$  by this method. It is obvious that one division by the series  $\sum_{n=0}^{\infty} e(n)q^n$  in Equation (18) takes  $\sum_{k=1}^N k = \frac{N(N+1)}{2}$  multiplications. Thus in all the number of multiplications using Herrmann's method is at least  $12N(N+1)$ . However, the multiplications are trivial, as one factor is a coefficient  $e(n)$  and therefore in  $\{-1; 0; 1\}$ . Additionally, the case  $e(n) = 0$  is by far the most common. Thus we cannot compare this number of multiplications directly to the number  $M(N)$  of Section 3. The CPU time of this method to compute  $c(n)$ ,  $-1 \leq n \leq 50000$ , was 39 minutes, 39 seconds.

We next explain our similar method. The fundamental difference to Herrmann's approach is that instead of successively dividing by  $\sum_{n=0}^{\infty} e(n)q^n$  in Equation (18), we first compute a series representing the 24-th power of  $\sum_{n=0}^{\infty} e(n)q^n$ . More precisely, let  $e_{24}(n)$  denote the Fourier coefficients of the series  $(\sum_{n=0}^{\infty} e(n)q^n)^{24}$ . Thus we set  $\sum_{n=0}^{\infty} e_{24}(n)q^n = (\sum_{n=0}^{\infty} e(n)q^n)^{24}$ . Again Equation (18) yields an obvious recursion formula for the  $c(n)$ , once the values  $e_{24}(n)$  and  $\sigma_{11}(n)$  are known. In contrast to the coefficients  $e(n)$  the computation of the values  $e_{24}(n)$  is more burdensome.

In Section 5 we use a Hecke representation of  $\eta^8$  to get the Fourier coefficients of the series  $(\sum_{n=0}^{\infty} e(n)q^n)^8$ . Similar as above we denote these coefficients as  $e_8(n)$ . Then we use standard exponentiation methods to compute the coefficients  $e_{24}(n)$ .

Unfortunately, we are not able to count the number of multiplications of this method to compute the coefficients  $c(n)$ ,  $n \leq N$ . Hence we cannot present a theoretical estimation of the complexity of our approach. Nevertheless, our practical tests give evidence, that our method is slightly faster than the original method of Herrmann. For example,  $N = 50000$  yields a run time of 35 minutes, 39 seconds. Furthermore this approach seems to be faster than using Mahler's formula of Section 3.

## 5 Computations Via Hecke Series

The method of this section is similar to the approach of Section 4. We use the formula

$$j(z) = \gamma_2^3(z) \quad \text{with} \quad \gamma_2(z) = \frac{E_4(z)}{\eta^8(z)}. \quad (19)$$

However, it is known from Schoeneberg [Schoeneberg, 1953] and later writers ([Serre, 1985], [Köhler, 1988]) that several powers of  $\eta(z)$  are represented by theta series with a Hecke character on an imaginary quadratic number field and that, therefore, their Fourier expansion is lacunary. Specifically we have (see [Köhler, 1988], p. 84)

$$\eta^8(z) = \frac{1}{6} \cdot \sum_{\mu \in \mathbb{Z}[\omega]} \chi(\mu) \mu^3 e\left(\frac{1}{3} \mu \bar{\mu} z\right),$$

where  $\omega = e\left(\frac{1}{6}\right) = \frac{1}{2}(1 + \sqrt{-3})$  and

$$\chi(x + y\omega) = \left(\frac{x - y}{3}\right)$$

for  $x, y \in \mathbb{Z}$ , with a quadratic residue symbol on the right hand side. We collect the contribution of associated and conjugate elements in  $\mathbb{Z}[\omega]$  and obtain the expansion

$$\eta^8(z) = \sum_{\substack{n > 0 \\ n \equiv 1 \pmod{3}}} a_8(n) e\left(\frac{nz}{3}\right), \quad (22)$$

$$a_8(n) = \sum_{\substack{x > 0, \\ x^2 = n}} \left(\frac{x}{3}\right) \cdot x^3 + \sum_{\substack{1 \leq y < x, \\ x^2 + xy + y^2 = n}} \left(\frac{x - y}{3}\right) \cdot (x - y)(2n + 3xy). \quad (23)$$

In Section 4 we introduced coefficients  $e_8(n)$  defined as Fourier coefficients of the series  $(\sum_{n=0}^{\infty} e(n)q^n)^8$ . The relation  $e_8((n-1)/3) = a_8(n)$  is obvious from Equation (22). Thus Equation (23) yields an efficient algorithm to compute a table of the coefficients  $e_8(n)$ .

It is well known that  $\gamma_2(z) = q^{-1/3} \sum_{n=0}^{\infty} g(n)q^n$  with integers  $g(n)$ . We combine Equations (19) and (4) to get a recursion formula for the coefficients  $g(n)$ . More precisely, it is easy to see that for  $n \geq 1$  we have

$$g(n) = 240 \cdot \sigma_3(n) - \sum_{k=0}^{n-1} g(k) e_8(n - k) \quad (24)$$

Again we then make use of standard exponentiation techniques to compute the values of  $c(n)$  from the relation  $j = \gamma_2^3$ .

Although this method is very similar to our variant of Herrmann's algorithm, it turns out to be much slower for  $N = 50000$ . The run time is 202 minutes, 33 seconds. The reason is that using the power function is rather slow for the large coefficients  $g(n)$  compared to

the coefficients  $e_8(n)$ . The CPU time to compute the values  $g(n)$ ,  $n \leq N + 1$ , is only 8 minutes, 16 seconds. Hence 95.9% of the run time is spent to compute  $c(n)$  from  $g(n)$ .

There is a variant of this method. We observe that

$$j = \frac{E_4^3 - E_6^2 + E_6^2}{\Delta} = 12^3 + \frac{E_6^2}{\Delta} = 12^3 + \gamma_3^2 \quad (25)$$

with

$$\gamma_3 = \frac{E_6}{\eta^{12}}. \quad (26)$$

There is no representation of  $\eta^{12}$  as a theta series with Hecke character. But we have ([Köhler, 1988], p. 88)

$$\eta^6(z) = \frac{1}{4} \cdot \sum_{\mu \in \mathbb{Z}[\sqrt{-1}]} \chi(\mu) \mu^2 e\left(\frac{1}{4} \mu \bar{\mu} z\right), \quad (27)$$

where  $\chi(x + y\sqrt{-1}) = (-1)^y$  if  $x \not\equiv y \pmod{2}$  and  $\chi(x + y\sqrt{-1}) = 0$  otherwise. Thus we can tabulate  $\eta^6$  as efficiently as  $\eta^8$ . Squaring  $\eta^6$  yields  $\eta^{12}$ , a division gives  $\gamma_3$ , and squaring again gives the coefficients  $c(n)$  of  $j(z)$ . In practice we observe that this variant is far more efficient than the first one via  $\gamma_2$ . The total CPU time is 95 minutes, 53 seconds.

## 6 A Formula of Kaneko and Zagier

In this section we describe a method which was discovered by D. Zagier [Zagier, 1996] and M. Kaneko [Kaneko, 1999]. The main Formula (31) makes use of coefficients  $t(n)$  introduced by Zagier. Once the  $t(n)$  are known, Equation (31) promises to be highly efficient since it requires just additions, but essentially no multiplications. Indeed we will see that this method turns out to be the most efficient one to compute the coefficients  $c(n)$ .

Zagier defined the sequence of numbers  $t(n)$  using certain singular values of  $j(z)$ . We call the numbers  $t(n)$  *Zagier coefficients*. Zagier gave an equivalent definition of the  $t(n)$  by means of the Fourier expansion of a meromorphic modular form of weight  $\frac{3}{2}$ , namely,

$$g(z) = -\frac{E_4(4z)\theta_1(z)}{\eta(4z)^6} = \sum_d t(d)q^d, \quad (28)$$

where  $\theta_1(z) = \sum_{n=-\infty}^{\infty} (-1)^n q^{n^2}$  is one of Jacobi's theta series. We have

$$t(-1) = -1, \quad t(0) = 2, \quad t(d) = 0 \quad \text{if } d < -1 \quad \text{or } d \equiv 1, 2 \pmod{4}. \quad (29)$$

Zagier proved the recursion formulas

$$\sum_{r \in \mathbb{Z}} r^2 t(4n - r^2) = -480\sigma_3(n), \quad \sum_{r \in \mathbb{Z}} t(4n - r^2) = 0 \quad (30)$$

for  $n \geq 1$ . It is obvious that the relations of Equation (30) uniquely determine the values  $t(3)$ ,  $t(4)$ ,  $t(7)$ ,  $\dots$ . Using this, Kaneko proved

$$c(n) = \frac{1}{n} \left( \sum_{r \in \mathbb{Z}} t(n - r^2) + \sum_{r > 0, r \text{ odd}} ((-1)^n t(4n - r^2) - t(16n - r^2)) \right) \quad (31)$$

for  $n \geq 1$ . When we use this formula to compute a table of  $c(n)$  for  $n \leq N$ , we need to compute a table of the Zagier coefficients  $t(d)$  for  $d \leq 16N$ . As  $t(d) = 0$  for  $d \equiv 1, 2 \pmod{4}$ , this is essentially a table of length  $8N$ . Once the Zagier coefficients are known we just have to do additions to compute the  $c(n)$  using Equation (31).

We explain how to recursively compute the Zagier coefficients. It is obvious from the formulas in Equation (30) that if some  $n \geq 1$  is given, we get the following recursion:

$$\begin{aligned} t(4n - 1) &= -240\sigma_3(n) - \sum_{r=2}^{\sqrt{4n+1}} r^2 t(4n - r^2), \\ t(4n) &= -2 \sum_{r=1}^{\sqrt{4n+1}} t(4n - r^2). \end{aligned}$$

Most of the CPU time is spent to compute the values  $t(d)$ . If  $N = 50000$ , their computation takes us 8 minutes, 19 seconds. The run time of the whole computation of the  $c(n)$ ,  $n \leq N$ , is 8 minutes, 43 seconds. Thus 95% of the CPU time is spent to compute the table of the Zagier coefficients.

In Section A we list the coefficient  $t(800000)$ . We choose this coefficient, as for  $N = 50000$  we have to compute the values  $t(d)$  up to  $d = 799999$ . We remark that  $t(800000)$  is an integer of bitlength 4056.

Finally we remark that if  $N$  is of order of magnitude 50000, this method assumes that a large quantity of main memory is to our disposal, say more than 500 MByte. For example, we terminated this algorithm on a PC having about 100 MByte of main memory after 15 minutes. At this point the CPU usage of our process was less than 5%, while the swapping process took almost all of the time.

## References

- [Baier, 2002] H. Baier. *Efficient Algorithms for Generating Elliptic Curves over Finite Fields Suitable for Use in Cryptography*. PhD thesis, Darmstadt University of Technology, 2002.
- [Gouvêa, 1997] F.Q. Gouvêa. Non-ordinary primes: A story. *Exp. Math.*, 6/3:195–205, 1997.
- [Herrmann, 1973] O. Herrmann. Über die Berechnung der Fourierkoeffizienten der Funktion  $j(\tau)$ . *J. f. d. reine u. angew. Math.*, 274:187–195, 1973.

- [Kaneko, 1999] M. Kaneko. Traces of singular moduli and the Fourier coefficients of the elliptic modular function  $j(\tau)$ . volume 19 of *Number Theory. Fifth Conf. Canad. Number Theory Assoc., Ottawa, Ontario, Canada, Aug. 1996*. AMS, CRM Proc. Lect. Notes, 1999.
- [Köhler, 1988] G. Köhler. Theta series on the Hecke groups  $G(\sqrt{2})$  and  $G(\sqrt{3})$ . *Math. Z.*, 197/1:69–96, 1988.
- [Knopp, 1990] M. Knopp. Rademacher on  $J(\tau)$ , Poincaré series of nonpositive weights and the Eichler cohomology. *Notices Amer. Math. Soc.*, 37:385–393, 1990.
- [Mahler, 1976] K. Mahler. On a class of non-linear functional equations connected with modular functions. *Journal of the Australian Mathematical Society*, 22, Series A:65–120, 1976.
- [Niebur, 1975] D. Niebur. A formula for Ramanujan’s  $\tau$ -function. *Ill. J. Math.*, 19:448–449, 1975.
- [Rademacher, 1938] H. Rademacher. The Fourier coefficients of the modular invariant  $J(\tau)$ . *Amer. J. Math.*, 60:501–512, 1938.
- [Ramanujan, 1927] S. Ramanujan. *Collected Papers*. Cambridge, 1927. Reprinted New York, 1962.
- [Schoeneberg, 1953] B. Schoeneberg. Über den Zusammenhang der Eisensteinschen Reihen und Thetareihen mit der Diskriminante der elliptischen Funktionen. *Math. Ann.*, 126:177–184, 1953.
- [Serre, 1985] J.P. Serre. Sur la lacunarité des puissances de  $\eta$ . *Glasg. Math. J.*, 27:203–221, 1985.
- [Zagier, 1996] D. Zagier. Traces of singular moduli, 1996. Preprint.

## A A Sample Zagier Coefficient

In order to show how large Zagier’s coefficients become we list the coefficient  $t(800000)$ . It is an integer of bitlength 4056 .

$t(800000)$  = 2164260999701052804585981227488262471198748472952\  
9733696975143657080601035731790618173990526143533\  
3656512282742365636119956617463196459050613824587\  
6750440295776998790470423898792992957601904049205\  
2973403996546755959877527784339014288775236748835\  
2863476016634146494249609602402917308097572254788\  
7973319641485734514318500429668348518525812824026\  
0074890731020416830998364778099390291027394641183\  
2430533387006349005324030346999047083617137851988\  
1210160600271788791610210207214991198832620779142\  
5161510503902500820717723125283468679059769785638\  
3625441026469277090619710353216297142058551845001\  
2536352059514829394139977330472898704169335282699\  
2655586854087319998910783830452828265611170156566\  
4610378672848078406365514503837549643769613967403\  
6486510953444974172510427077521129204294180980867\  
9992323250061136672353088366222604434782193190281\  
2550994292744548193301365833106507850565952542288\  
9946965717275618329353414719784089114371714546118\  
1640821077406275518949910112812979406046773037141\  
5907987653061887825150042047996556954159609405596\  
4178054544759745095141452724977338405984760557339\  
4993676514688839783009090058502226547817882109781\  
5745036709112156565866550240890475100791145297982\  
528846788862573350090531733289276113660937680