Lecture 1

Fast Algorithms for Elementary Functions, π, and γ[*]

[*]Six lectures on Algorithms, Trinity term 1999.

Summary

- High-precision arithmetic
- Use of Newton's method
- Some algorithms for exp and ln
- The arithmetic-geometric mean
- Faster algorithms for exp and ln
- Gauss-Legendre algorithm for π
- Some other algorithms for π
- Other elementary functions
- Euler's constant γ
- A connection with Ramanujan

High-precision arithmetic

We are concerned with high-precision computations on integers or floating-point approximations to real numbers.

If N is a (large) integer we can represent N using base (or radix) β, say

$$N = \pm \sum_{i=0}^{t-1} d_i \beta^i \, ,$$

where the d_i are "base β digits", i.e.

$$0 \le d_i < \beta \, .$$

The sign can be handled in several ways. We usually assume $N \ge 0$ for simplicity.

On a binary computer with a fixed word length, say w bits, it is convenient to choose the β so that base-β digits can be represented exactly in a single word. Thus, we choose

$$2 \le \beta \le 2^w \, .$$

It is often convenient to choose β significantly smaller than 2^w. For example, if $w = 32$, we might choose $\beta = 2^{31}$ or 2^{14} or 10^4.

Addition

The number of operations required to add two t-digit numbers is $O(t)$. If $\beta = 2$ this is called the "bit complexity" to distinguish it from the "operation complexity" (which is simply 1).

On a machine with fixed wordlength w bits the "bit complexity" and the "word complexity" only differ by a constant factor so we ignore the distinction (though in practice the constant is important).

Since we are considering a serial computer and ignoring constant factors, we usually call the bit complexity "time".

Multiplication

The time required to multiply two t-digit numbers $\sum a_i\beta^i$ and $\sum b_j\beta^j$ by the "schoolboy" method is $O(t^2)$. However, this is not optimal. Most of the computation involves forming convolutions

$$c_k = \sum_{i+j=k} a_i b_j$$

and we can do this by the *Fast Fourier Transform* (FFT) in $O(t \log t)$ operations.
In fact, to ensure that the c_k are computed exactly, these "operations" are more than single word operations, and we get another $\log t$ factor. A more complicated algorithm, the *Schönhage-Strassen* algorithm, reduces this to $\log \log t$.

Thus, the conclusion is that the time required to multiply t-digit numbers is

$$M(t) = O(t \log t \log \log t)\ .$$

Optimality

Is the Schönhage-Strassen algorithm optimal ?

The answer depends on the model of computation. On certain models $O(t)$ is attainable. However, for realistic models Schönhage-Strassen is the best known algorithm.

In case the Schönhage-Strassen algorithm is not optimal on your model of computation (or you choose to implement a simpler algorithm), we express the time bounds in terms of the function $M(t)$ rather than explicitly using $M(t) = O(t \log t \log \log t)$.

Assumptions

We need to assume that $M(t)$ satisfies some reasonable smoothness properties. For example, assume that $\exists\ \alpha, \beta \in (0,1)$, for all sufficiently large t,

$$M(\alpha t) \le \beta M(t)\ .$$

This assumption certainly holds if

$$M(t) \sim ct \log t \log \log t\ .$$

For convenience, we also assume that $M(t) = 0$ for $t < 1$.

Our assumptions ensure that

$$\sum_{k=0}^{\infty} M(\lceil t/2^k \rceil) = O(M(t))$$

which is useful in analysing the complexity of Newton's method.

MP Floating Point

We can represent multiple-precision "floating-point" numbers by pairs (e, N) of (multiple-precision) integers e (the exponent) and N (the scaled fraction).

The pair (e, N) is interpreted as $\beta^e N$ or perhaps as $\beta^{e-t} N$.

Clearly we can perform addition and multiplication on such floating-point numbers, obtaining a t-digit result, i.e. a result with relative error $O(\beta^{-t})$, by performing operations on the exponents and (shifted) fractions, in time $O(M(t))$.

Reciprocals by Newton's method

Newton's iteration for finding a simple zero of a function f is

$$x_{i+1} = x_i - f(x_i)/f'(x_i) \ .$$

Apply this to the function

$$f(x) = a - 1/x \ ,$$

where $a \neq 0$ is constant. Provided the initial approximation x_0 is not too bad, the sequence converges to the (unique) zero of f, i.e. $x = 1/a$. Also, the iteration simplifies to

$$x_{i+1} = x_i + x_i(1 - ax_i) = x_i(2 - ax_i)$$

which *only involves additions/subtractions and multiplications.*

Convergence is quadratic, as we can easily see by taking $\epsilon_i = 1 - ax_i$ and observing that

$$\epsilon_{i+1} = 1 - (1 - \epsilon_i)(1 + \epsilon_i) = \epsilon_i^2 \ .$$

Thus, the number of correct digits approximately *doubles* at each iteration.

Complexity of reciprocals

To obtain a t-digit reciprocal, we need to perform about $\lg(t)$ Newton iterations (here, as usual, lg denotes $\log_2$). If these are performed using t-digit arithmetic then the time required is $T_{rec}(t) = O(M(t) \log t)$.

However, we can improve the bound by observing that Newton's algorithm is *self-correcting*, so we can start with a small number of digits and (almost) double it at each step. It is most efficient to contrive that the last iteration is performed with (slightly more than) t digits.

For example, to get a 100-digit reciprocal, we might use $2, 2, 3, 5, 8, 14, 26, 51, 101$ digits at successive iterations.

By our smoothness assumptions, the total time is only $T_{rec}(t) = O(M(t))$. We have avoided the factor $\log(t)$.

It is also possible to show that $M(t) = O(T_{rec}(t))$, so in a sense multiplication and reciprocation are *equivalent.*

Division

To perform a t-digit "floating point" division, we can use

$$a/b = a \times (1/b)$$

where the reciprocal $1/b$ is computed by Newton's method. Thus, floating-point division also takes time $O(M(t))$.

For integer division, it is easy to obtain the correct quotient and remainder in time $O(M(t))$.

If b is small, then the integer division of a by b takes time $O(t)$.

Square roots

Newton's method can be applied to compute t-digit square root approximations in time $O(M(t))$, using the classical iteration

$$x_{i+1} = \frac{1}{2}\left(x_i + \frac{a}{x_i}\right) \ .$$

In fact, it is slightly more efficient to approximate the *inverse square root* $a^{-1/2}$ by Newton's method (avoiding divisions), then use

$$a^{1/2} = a \times a^{-1/2} \ .$$

Algorithms for $\exp(x)$ and $\ln(x)$

$\exp(x)$ and $\ln(x)$ are inverse functions, i.e.

$$\exp(\ln(x)) = x \ ,$$

at least for real positive x. Thus, if we can compute one then we can compute the other in the same time (up to a small constant factor) using Newton's method.

Whenever I describe an algorithm for $\exp(x)$, one for $\ln(x)$ is implied, and *vice versa.*

To avoid technicalities, we assume that the argument x is real and in some bounded, closed interval $[a, b]$. Also, in the case of $\ln(x)$, we assume $a > 0$.

$\exp(x)$ – Algorithm 1

The most obvious way to approximate $\exp(x)$ is to sum a sufficient number of terms in the power series

$$\exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!} \ .$$

Using Stirling's approximation to $n!$, it is not hard to see that we need $\Theta(t/\log t)$ terms to obtain t-digit accuracy. Thus, the time required for $\exp(x)$ is

$$T_{exp}(t) = O\left(M(t)\frac{t}{\log t}\right) \ .$$

Using Newton's method, we deduce that the time required for $\ln(x)$ is

$$T_{ln}(t) = O\left(M(t)\frac{t}{\log t}\right) \ .$$

$\exp(x)$ – Algorithm 2

The power series

$$\exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

converges faster for smaller $|x|$. Thus, an obvious idea is to apply the identity

$$\exp(x) = (\exp(x/2))^2$$

k times before summing the power series.

In other words, evaluate $\exp(x/2^k)$ using the power series, then compute

$$\exp(x) = \left(\exp\left(x/2^k\right)\right)^{2^k}$$

by squaring the result k times.

To get t-digit accuracy by this method we need $\Theta(t/k)$ terms in the power series, so the overall time is

$$T_{exp}(t) = O\left(M(t)\left(k + \frac{t}{k}\right)\right) \ .$$

Optimal choice of k

Choosing $k \sim \sqrt{t}$ gives

$$T_{exp}(t) = O(M(t)\sqrt{t})$$

which is better by a factor $\sqrt{t}/\log t$ than the bound obtained from Algorithm 1.

Although there are asymptotically faster algorithms (as we shall see), Algorithm 2 is the fastest algorithm in practice for a wide range of t.

Historical notes

Algorithm 2 was proposed and analysed in my 1976 paper “The complexity of multiple-precision arithmetic” [7].

For the history of other results, e.g. T_{rec}, see the bibliographic notes in Aho, Hopcroft and Ullman [1, Ch. 8].

The AGM

Can we do better than Algorithm 2 ?
Yes, but first we need to describe the *arithmetic-geometric mean* (AGM) iteration.

Let a_0, b_0 be given initial values. For simplicity we assume they are positive reals (though an extension to the complex plane is possible so long as care is taken with the branch of the square root below). The iteration is defined by

$$a_{i+1} = \frac{a_i + b_i}{2}$$

$$b_{i+1} = \sqrt{a_i b_i}$$

Gauss studied the AGM and showed that a_i and b_i converge to a common limit

$$\mathrm{AGM}(a_0, b_0)$$

which can be expressed in terms of a complete elliptic integral.

Complete elliptic integrals

We define the *complete elliptic integral of the first kind* by

$$K(\phi) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1 - \sin^2\phi \sin^2\theta}} \, .$$

Similarly, the *complete elliptic integral of the second kind* is

$$E(\phi) = \int_0^{\pi/2} \sqrt{1 - \sin^2\phi \sin^2\theta}\, d\theta \, .$$

Gauss showed that

$$\mathrm{AGM}(1, \cos\phi) = \frac{\pi}{2K(\phi)} \, . \qquad (1)$$

For a proof, see Borwein and Borwein, “Pi and the AGM” [4, §1.2].

Also, if $c_0 = \sin\phi$ and $c_{i+1} = (a_i - b_i)/2$, then

$$1 - \frac{E(\phi)}{K(\phi)} = \sum_{i=0}^{\infty} 2^{i-1} c_i^2 \, . \qquad (2)$$

Thus, both $K(\phi)$ and $E(\phi)$ can be computed via the AGM, given $\cos\phi$.

Remarks

There is no loss of generality in assuming that $a_0 \ge b_0$, since

$$\mathrm{AGM}(a_0, b_0) = \mathrm{AGM}(b_0, a_0) \, .$$

Also, there is no real loss of generality in assuming that $a_0 = 1$, since

$$\lambda\, \mathrm{AGM}(a_0, b_0) = \mathrm{AGM}(\lambda a_0, \lambda b_0) \, .$$

Thus, we can assume $a_0 = 1$ and $b_0 = \cos\phi$, as above.

Notation

Instead of the argument ϕ, the argument

$$k = \sin\phi$$

is often used. k is known as the *modulus* and

$$k' = \cos\phi$$

is known as the *complementary modulus*. We also define $\phi' = \pi/2 - \phi$ so $k' = \sin\phi'$.

Quadratic convergence of the AGM

The reason why the AGM is of computational interest is that it converges quadratically. In fact, if we define ϵ_i by

$$b_i/a_i = 1 - \epsilon_i \, ,$$

then it is easy to verify that

$$\epsilon_{i+1} = \epsilon_i^2/8 + O(\epsilon_i^3) \, .$$

Also, if $b_i/a_i \ll 1$ then

$$b_{i+1}/a_{i+1} = \frac{2\sqrt{b_i/a_i}}{1 + b_i/a_i} \approx 2\sqrt{b_i/a_i} \, ,$$

so after about $\lg(a_0/b_0)$ iterations we get $a_i/b_i = O(1)$. In other words, even if a_0/b_0 is large, it does not take long before quadratic convergence sets in.

Extreme cases

If ϕ is small, then

$$K(\phi) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1 - \sin^2\phi \sin^2\theta}} = \frac{\pi}{2} + O(\phi^2) \ .$$

Also, less obviously, if $k = \sin\phi$ is small and $\phi' = \pi/2 - \phi$, then

$$K(\phi') = \left(1 + O(k^2)\right) \ln\left(\frac{4}{k}\right) \ .$$

See Borwein and Borwein [4, ex. 1.3.4(b)]

$\log(x)$ – Algorithm 3

The last result implies (by exchanging primed and unprimed variables) that, if $a_0 = 1$ and $b_0 = \cos\phi = \epsilon^{1/2}$ is small, then

$$K(\phi) = (1 + O(\epsilon)) \ln\left(\frac{4}{b_0}\right) \ .$$

However, we can compute $K(\phi)$ by the AGM. In other words, if $y = 4/b_0$ is large, then

$$\ln y = \frac{\pi}{2\,\mathrm{AGM}(1, 4/y)} \left(1 + O\left(\frac{1}{y^2}\right)\right) \ .$$

This is fine for computing $\ln y$ provided

1. y is sufficiently large ($y \geq \beta^{t/2}$).
2. We know π to sufficient accuracy.

The first condition is easy to handle. We can take $M \approx \beta^{t/2}$ and compute

$$\ln y = \ln(My) - \ln(M) \ ,$$

using an additional $O(\log t)$ digits to compensate for cancellation.

Approximating π

Regarding the second condition: if we do not already know π to sufficient accuracy, we can compute

$$\frac{\ln(1 + \beta^{-t})}{\pi}$$

by the above method, and use

$$\ln(1 + \beta^{-t}) = \beta^{-t}(1 + O(\beta^{-t}))$$

to find a sufficiently accurate approximation to π. However, we have to work with about $2t$ digits to compensate for cancellation.

This shows that

$$T_\pi(t) = O(M(t) \log t) \ .$$

However, the constant factor implicit in the "O" is large. A better way of finding π, with smaller constant factor, will be discussed soon.

Complexity of ln

Using the results obtained so far, we have

$$T_{ln}(t) = O(M(t) \log t)$$

because the AGM converges quadratically and only $O(\log t)$ iterations are required.

Remark

The AGM is *not* self-correcting. Thus, we can not get rid of the "$\log t$" factor in these bounds by starting with low precision as we did for reciprocals and square roots by Newton's method.

Complexity of exp

Using Newton's method and Algorithm 3 for ln, we obtain

$$T_{exp}(t) = O(M(t) \log t) \ .$$

Historical notes

The $O(M(t)\log t)$ algorithms for log and exp were first published in my 1975 paper "Multiple-precision zero-finding methods and the complexity of elementary function evaluation" [5]. The algorithm for ln is implicit in Beeler, Gosper and Schroeppel's unpublished 1972 MIT Technical Report "HAKMEM" [2].

Different $O(M(t)\log t)$ algorithms for log and exp were published in my 1976 J. ACM paper "Fast multiple-precision evaluation of elementary functions" [6]. These algorithms use incomplete elliptic integrals and Landen transformations. The 1976 paper was actually written before the 1975 paper (J. ACM has a long publication delay).

Legendre's relation

Legendre found a beautiful relation between the four quantities $K(\phi), K(\phi'), E(\phi)$ and $E(\phi')$ where, as usual, $\phi + \phi' = \pi/2$:

$$E(\phi)K(\phi') + E(\phi')K(\phi) - K(\phi)K(\phi') = \frac{\pi}{2}\,.$$

For a proof, see Borwein and Borwein [4, §1.6].

All we need to obtain a fast algorithm for the computation of π is the special case $\phi = \phi' = \pi/4$. Then, abbreviating $K(\pi/4)$ by K and $E(\pi/4)$ by E, Legendre's relation reduces to

$$2EK - K^2 = \frac{\pi}{2}\,. \qquad (3)$$

Fast evaluation of π

Suppose we perform the AGM iteration with initial values $a_0 = 1$ and $b_0 = \cos(\pi/4) = 1/\sqrt{2}$, obtaining a good approximation to the limit $a = \mathrm{AGM}(1, 1/\sqrt{2})$. Then, by Gauss's result (1) on the limit of the AGM,

$$2a = \frac{\pi}{K}\,.$$

Also, using the relation (2), we can find E/K from quantities occurring during the AGM computation. Hence, dividing each side of (3) by K^2, we obtain a known quantity

$$\frac{2E}{K} - 1$$

on the left, and the quantity

$$\frac{1}{2\pi}\left(\frac{\pi}{K}\right)^2$$

on the right, where everything is known except for the factor $\frac{1}{2\pi}$. Hence, we can find π !

The Gauss-Legendre algorithm for π

Simplifying the above, we obtain a nice algorithm for the computation of π. In my 1975–76 papers [5, 6] I called it the *Gauss-Legendre algorithm* because, as we have seen, it depends on results of Gauss and Legendre. The algorithm was discovered independently by Salamin (1976) [12].

```
$A \leftarrow 1$; $B \leftarrow 2^{-1/2}$;
$T \leftarrow 1/4$; $X \leftarrow 1$;
while $A - B > \beta^{-t}$ do
  begin
  $Y \leftarrow A$;
  $A \leftarrow (A + B)/2$;
  $B \leftarrow \sqrt{B \times Y}$;
  $T \leftarrow T - X \times (A - Y)^2$;
  $X \leftarrow 2 \times X$;
  end;
return $A^2/T$ {or, better, $(A + B)^2/(4T)$}.
```

The rate of convergence is illustrated in the following table.

Convergence of the Gauss-Legendre Method

Iteration	$A^2/T - \pi$	$\pi - (A+B)^2/(4T)$
0	8.6'-1	2.3'-1
1	4.6'-2	1.0'-3
2	8.8'-5	7.4'-9
3	3.1'-10	1.8'-19
4	3.7'-21	5.5'-41
5	5.5'-43	2.4'-84
6	1.2'-86	2.3'-171
7	5.8'-174	1.1'-345
8	1.3'-348	1.1'-694
9	6.9'-698	6.1'-1393

From the table we see that

$$(A+B)^2/(4T) < \pi < A^2/T$$

(this can be proved rigorously). Also, convergence is quadratic, as expected. It takes 9 iterations to get 1000 decimals, 19 iterations to get 10^6 decimals, and only 29 iterations to get 10^9 decimals !.

Complexity of π evaluation

From the algorithms above,

$$T_\pi(t) = O(M(t)\log t)\ .$$

It is an open question whether this bound is the best possible.

Comparison with other algorithms

Archimedes suggested bounding π by the areas of regular n-gons inside/outside the unit circle (or half their lengths). For example, starting with regular hexagons, we can obtain a recurrence for doubling n which gives bounds for $n = 12, 24, 48, 96, \ldots$ (see Borwein & Borwein [4, Ch. 11]).

Each iteration involves a square root and a few multiplications/additions, so is comparable to an AGM iteration. However, convergence is only *linear*, not quadratic. To obtain 10^6 digits of π by such a method would take more than 10^6 iterations (compare 19 for the Gauss-Legendre method).

Improvements on Archimedes

Other methods involve the arctan formula

$$\arctan\theta = \theta - \theta^3/3 + \theta^5/5 - \cdots$$

combined with Machin's formula

$$\frac{\pi}{4} = 4\arctan\frac{1}{5} - \arctan\frac{1}{239}$$

or similar formulae. They are good for moderate precision but give only linear convergence, so Gauss-Legendre must win eventually.

There are many other methods, but they nearly all have linear (or worse) convergence. The only known methods competitive (for high precision computations) with the Gauss-Legendre method are similar methods based on the AGM – see Borwein and Borwein [4].

Other elementary functions

By considering the AGM for complex arguments, or by using incomplete elliptic integrals and Landen transformations, we can compute any elementary function sin, cos, tan, sinh etc or its inverse arctan etc, in a compact interval not containing any singularities of the function, in time

$$O(M(t)\log t)\ .$$

It is not known whether this result is best possible.

Euler's constant

Euler's constant $\gamma = -\Gamma'(1)$ is usually defined by

$$\gamma = \lim_{n\to\infty} (H_n - \ln(n)) \ ,$$

where

$$H_n = \sum_{k=1}^{n} \frac{1}{k} \ .$$

It is not known whether γ is rational or irrational. Hence, there is some interest in computing γ and its regular continued fraction (perhaps more interest than in computing π, since π is known to be transcendental).

The defining limit converges much too slowly to be useful, but it can be accelerated by approximating the truncation error by an Euler-Maclaurin expansion. The difficulty here is how to quickly generate the Bernoulli numbers required for the Euler-Maclaurin expansion.

Use of Bessel functions

The fastest known method for computing γ, due to Brent and McMillan [9], was derived using results on Bessel functions, but it can be regarded as a "smoothing" of the defining limit. Specifically, let

$$U(n) = \sum_{k=0}^{\infty} \left(\frac{n^k}{k!}\right)^2 H_k \ - \ \ln(n)V(n) \ ,$$

where

$$V(n) = \sum_{k=0}^{\infty} \left(\frac{n^k}{k!}\right)^2 \ .$$

Then

$$0 < \frac{U(n)}{V(n)} - \gamma < \pi e^{-4n} \ ,$$

so computation of $U(n)/V(n)$ gives γ to of order n digits. This can be done in time $O(n^2)$, or even faster if we use rational arithmetic and a binary tree for summation. Thus

$$T_\gamma(t) = O(M(t)(\log t)^2) \ .$$

For details see [9].

Recent computations

γ has recently been computed to 10^6 decimals by Thomas Papanikolaou. By computing 470,006 partial quotients of its continued fraction, Papanikolaou has shown that *if* γ is rational, say $\gamma = p/q$, then

$$q > 10^{242080} \ .$$

By computing more partial quotients, it should be possible to improve this to

$$q > 10^{499900} \ .$$

Papanikolaou's computation improved the 1980 result of Brent and McMillan, who computed γ to 30,100 decimals and showed that $q > 10^{15000}$, using 29,200 partial quotients.

A connection with Ramanujan

Ramanujan published several series for γ, e.g. generalisations of Glaisher's

$$\gamma = 1 - \sum_{k=1}^{\infty} \frac{\zeta(2k+1)}{(k+1)(2k+1)} \ ,$$

but these are not suitable for computation of γ.

In his first notebook [3, I,p.98] Ramanujan states that (in modern notation)

$$\sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{nk} \left(\frac{x^k}{k!}\right)^n = \ln x + \gamma + o(1) \qquad (4)$$

as $x \to +\infty$. Here n is a fixed positive integer.

The case $n = 1$ is correct, in fact the error term is just an exponential integral

$$\int_x^\infty \frac{e^{-u}}{u}\, du = O\left(\frac{e^{-x}}{x}\right)$$

(this result is due to Euler, and has been used by Sweeney and others to compute γ).

Unfortunately, (4) is <u>false</u> if $n > 2$.

What might have been

The case $n = 1$ (with improved error term) is

$$\sum_{k=1}^{\infty} \frac{(-1)^{k-1}x^k}{k!k} = \ln x + \gamma + O\left(\frac{e^{-x}}{x}\right) ,$$

and the sum on the left side can be written as

$$e^{-x} \sum_{k=0}^{\infty} H_k \frac{x^k}{k!} .$$

This is easy to prove, and was known by Ramanujan (see Berndt [3, I, pp. 46–47]). Thus, Ramanujan knew that

$$\sum_{k=0}^{\infty} H_k \frac{x^k}{k!} \Big/ \sum_{k=0}^{\infty} \frac{x^k}{k!} = \ln x + \gamma + O\left(\frac{e^{-x}}{x}\right) .$$

He could have generalised and made the "better" conjecture

$$\sum_{k=0}^{\infty} H_k \left(\frac{x^k}{k!}\right)^n \Big/ \sum_{k=0}^{\infty} \left(\frac{x^k}{k!}\right)^n = \ln x + \gamma + o(1) \tag{5}$$

instead of his incorrect claim (4).

Comments on the better conjecture

The case $n = 2$ of (5) is what was used (with a sharper error bound) by Brent and McMillan. The function $(x^k/k!)^n$ acts as a smoothing kernel with a peak at $k \approx x - \frac{1}{2}$.

In fact, (5) is correct and the error term $o(1)$ can be improved to

$$O\left(\exp(-c_n x)\right) ,$$

where

$$c_n = \begin{cases} 1 & \text{if } n = 1, \\ 2n \sin^2(\pi/n) & \text{if } n \geq 2. \end{cases}$$

The case $n = 3$ is interesting because

$$\max_{n=1,2,\ldots} c_n = c_3 = 4.5 ,$$

but no one seems to have used $n > 2$ in a serious computation of γ.

Historical notes

Early computations of γ, up to Knuth (1962), used the Euler-Maclaurin formula.

Sweeney (1963) used essentially the (correct) case $n = 1$ of Ramanujan's (4), with the error term replaced by an asymptotic expansion. I used Sweeney's method and continued fractions in 1977 to show that $q > 10^{10000}$.

In 1980, Brent and McMillan[1] used the case $n = 2$ of the "better" conjecture (proved using results on the asymptotic behaviour of the modified Bessel functions $I_0(z)$ and $K_0(z)$).

In a 1994 paper [10] I noted the connection with Ramanujan[2].

[1]Edwin McMillan (1907–1991), a physicist, is better known for his invention of the synchrotron (independently of Veksler) and for the discovery of neptunium and plutonium (1941). He shared the 1951 Nobel prize in chemistry with Seaborg. See *Nature* 353 (1991), 602.

[2]Ramanujan's story is too well known to need a footnote.

References

[1] Alfred V. Aho, John E. Hopcroft and Jeffrey D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[2] M. Beeler, R. W. Gosper, and R. Schroeppel, *HAKMEM*, MIT AI Lab, 1972.

[3] B. C. Berndt, *Ramanujan's Notebooks*, Parts I–III, Springer-Verlag, New York, 1985–1991.

[4] Jonathan M. and Peter B. Borwein, *Pi and the AGM*, Wiley-Interscience, John Wiley and Sons, New York, 1987.

[5] R. P. Brent, Multiple-precision zero-finding methods and the complexity of elementary function evaluation, in *Analytic Computational Complexity* (edited by J. F. Traub), Academic Press, New York, 1975, 151–176.

[6] R. P. Brent, Fast multiple-precision evaluation of elementary functions, *J. ACM* 23 (1976), 242–251.

[7] R. P. Brent, The complexity of multiple-precision arithmetic, in *The Complexity of Computational Problem Solving*, University of Queensland Press, Brisbane, 1976, 126–165.

[8] R. P. Brent, Computation of the regular continued fraction for Euler's constant, *Mathematics of Computation* 31 (1977), 771–777.

[9] R. P. Brent and E. M. McMillan, Some new algorithms for high-precision computation of Euler's constant, *Mathematics of Computation* 34 (1980), 305–312.

[10] R. P. Brent, Ramanujan and Euler's constant, in *Proceedings of Symposia in Applied Mathematics*, Vol. 48 (edited by W. Gautschi), American Mathematical Society, Providence, Rhode Island, 1994, 541–545.

[11] Donald E. Knuth, *The Art of Computer Programming* Volume 2: *Seminumerical Algorithms* (third edition, 1997), Addison-Wesley, 1997.

[12] E. Salamin, Computation of π using arithmetic-geometric mean, *Mathematics of Computation* 30 (1976), 565–570.