MATHEMATICS OF COMPUTATION
Volume 87, Number 310, March 2018, Pages 1013–1028
http://dx.doi.org/10.1090/mcom/3275
Article electronically published on November 1, 2017

COMPUTATIONS OF THE MERTENS FUNCTION AND IMPROVED BOUNDS ON THE MERTENS CONJECTURE

GREG HURST

ABSTRACT. The Mertens function is defined as $M(x) = \sum_{n\leq x} \mu(n)$, where $\mu(n)$ is the Möbius function. The Mertens conjecture states $|M(x)/\sqrt{x}| < 1$ for $x > 1$, which was proven false in 1985 by showing $\liminf M(x)/\sqrt{x} < -1.009$ and $\limsup M(x)/\sqrt{x} > 1.06$. The same techniques used were revisited here with present day hardware and algorithms, giving improved lower and upper bounds of -1.837625 and 1.826054. In addition, $M(x)$ was computed for all $x \leq 10^{16}$, recording all extrema, all zeros, and 10^8 values sampled at a regular interval. Finally, an algorithm to compute $M(x)$ in $O(x^{2/3+\varepsilon})$ time was used on all powers of two up to 2^{73}.

1. INTRODUCTION

The Möbius function $\mu(n)$ is an arithmetic function defined by

$$\mu(n) = \begin{cases} (-1)^{\omega(n)} & \text{if } n \text{ is a square-free integer,} \\ 0 & \text{otherwise,} \end{cases}$$

where $\omega(n)$ is the number of prime factors of n. The Mertens function is the summatory function of the Möbius function, i.e.,

$$M(x) = \sum_{n\leq x} \mu(n).$$

This is a well-known function in number theory, appearing in many identities. Its Mellin transform gives

$$\frac{1}{\zeta(s)} = s\int_1^\infty M(x)x^{-s-1}dx \quad \text{for} \quad \mathrm{Re}(s) > 1,$$

where $\zeta(s)$ is the Riemann zeta function. If $M(x) = O(x^{1/2+\varepsilon})$, the integral would converge for $\mathrm{Re}(s) > 1/2$, implying that $1/\zeta(s)$ has no poles in this region and that the Riemann hypothesis is true. Conversely, if $M(x) = \Omega(x^\alpha)$ for some $\alpha > 1/2$, then the Riemann hypothesis is false.

Defining $q(x) = M(x)/\sqrt{x}$, a long standing conjecture of Mertens stated $|q(x)| < 1$ for $x > 1$. In 1985 this was shown to be false by Odlyzko and te Riele who showed $\liminf q(x) < -1.009$ and $\limsup q(x) > 1.06$ [5]. However, no explicit counterexample was found. Since then Best and Trudgian have improved these bounds to $\liminf q(x) < -1.6383$ and $\limsup q(x) > 1.6383$ [4]. This paper describes

Received by the editor October 28, 2016, and, in revised form, March 4, 2017.
2010 *Mathematics Subject Classification.* Primary 11A25, 11M26, 11Y35, 11Y55, 11Y70.
Key words and phrases. The Mertens function, the Mertens conjecture.

techniques similar to those of Odlyzko and te Riele and establishes $\liminf q(x) < -1.837625$ and $\limsup q(x) > 1.826054$.

To better understand $M(x)$ and $q(x)$, some have computed $M(x)$ at every integer up to a given bound. The most recent and extensive results are due to Kotnik and van de Lune, who computed $M(x)$ for all $x \leq 10^{14}$ [3]. In this paper, these results are extended by computing $M(x)$ for all $x \leq 10^{16}$. For x in this range:

- all extrema,
- all zeros of $M(x)$ (366 567 325 in total),
- all values of $M(x)$ for x a multiple of 10^8,

are reported.

Finally, an algorithm is discussed that was used to compute $M(2^n)$ for all positive integers $n \leq 73$, including

$$M(2^{73}) = -6524408924.$$

Section 2 describes the sieve used to compute $M(n)$ for all $n \leq 10^{16}$ and used in the main algorithm in the subsequent section. Section 3 derives a formula and incorporates it into an algorithm used to calculate $M(x)$ at an isolated value. Section 4 discusses the machinery used to derive bounds on $|q(x)|$. This entails analytic formulas relating to $M(x)$ and a lattice basis reduction scheme. Section 5 discusses all implementation details, which include low level tricks to speed up common calculations and the choice of hardware specific parameters. Section 6 presents and discusses the results of the computations. These include extrema of $M(x)$, properties of the zeros of $M(x)$, the values of $M(x)$ at isolated values, and various bounds on $q(x)$. Finally, section 7 summarizes all results and considers possible extensions.

2. Sieving Algorithm

The functions $\mu(n)$ and $M(n)$ can be computed naively for all $n \leq x$ as follows [2]:

```
Compute and store all primes p ≤ √x
Initialize an array m of 1's of length ⌊x⌋
for each prime p ≤ √x do
    For all 1 ≤ n ≤ x divisible by p, set m[n] ← −p · m[n]
    For all 1 ≤ n ≤ x divisible by p², set m[n] ← 0
for 1 ≤ n ≤ x do
    If m[n] = 0, do nothing
    If |m[n]| = n, set m[n] ← sign(m[n])
    Otherwise, set m[n] ← −sign(m[n])
The array m now stores μ(n) at position n
Cumulatively add the values in m into another array. This array stores M(n)
  at position n
```

The runtime complexity of this sieve is determined by the first loop and is

$$O\left(\sum_{p\leq\sqrt{x}}\left(\frac{x}{p}+\frac{x}{p^2}\right)\right) = O(x\log\log x).$$

There are two problems that render this algorithm impractical for large x. The first is that it requires $O(x\log\log x)$ multiplications, which can be costly. The second is that the array m must contain integers rather than bytes, which is less cache friendly. The problem of cache misses is discussed in further detail in section 5. To address these issues a variation of this algorithm, similar to the one described in [1], is used. Define $\theta(x)$ as the unit step function and $lsb(x)$ as the least significant bit of x, and sieve as follows:

```
Create byte-arrays l of length ⌊√x⌋ and m of length ⌊x⌋
for 1 ≤ j ≤ √x do
    l[j] ← ⌊log2 pj⌋|1, where pj is the jth prime and | is bitwise OR
for 1 ≤ n ≤ x do
    m[n] ← 0x80 (set the most significant bit to 1 and the rest to 0)
for 1 ≤ j ≤ √x do
    For all 1 ≤ n ≤ x divisible by pj, set m[n] ← l[j] + m[n]
    For all 1 ≤ n ≤ x divisible by pj^2, set m[n] ← 0
for 1 ≤ n ≤ x do
    If the leading bit in m[n] is 0, set m[n] ← 0
    If m[n] < ⌊log2 n⌋ − 5 − 2θ(n − 2^20), set m[n] ← 2lsb(m[n]) − 1
    Otherwise, set m[n] ← 1 − 2lsb(m[n])
```

The idea of this algorithm is the same as the first one, except it works in log-space. This allows multiplication to be replaced with addition and data to be stored in byte-arrays. Though the time complexity remains the same, these changes reduce implementation overhead.

After the third loop, the leading bit of each element $m[n]$ indicates whether n is divisible by a square. This leaves 7 bits in $m[n]$ to add logarithms. Fortunately for all $n \leq 10^{16}$, the maximum possible amount of logarithms that can be added will not overflow to the eighth bit. In fact overflow will not occur until about $n = 10^{30}$. The least significant bit of each element $m[n]$ counts the parity of the number of prime factors encountered. If it is 0 there were an even amount and if it is 1 there were an odd amount. This is achieved by setting the least significant bit in each element of l to 1.

Finally, logarithms are summed to determine if n has a prime factor larger than $\sqrt{n}$ that was not accounted for in the sieve. For $n \leq 2^{20}$, all primes will be accounted for if and only if $\sum_j \lfloor \log_2 p_j \rfloor | 1 < \lfloor \log_2 n \rfloor - 5$, where all cases can be verified exhaustively. The validity for larger n is shown by the following theorem.

Theorem 2.1. *If* $2^{20} < n \leq 10^{16}$*, and* $n = p_1 \cdots p_k$ *is square-free, then*

$$\sum_{j=1}^{k} \lfloor \log_2 p_j \rfloor | 1 \geq \lfloor \log_2 n \rfloor - 7 \tag{1}$$

and

$$\sum_{j=1}^{k-1} \lfloor \log_2 p_j \rfloor | 1 < \lfloor \log_2 n \rfloor - 7 \quad \textit{when} \quad p_k > \sqrt{n}. \tag{2}$$

Proof. To show that (1) is true, a value of n is sought that gives a sum which deviates below $\log_2 n$ as far as possible. This will happen when there are many prime factors (allowing for more error), all $\lfloor \log_2 p_j \rfloor$ are odd (so the bitwise OR will not increment the sum), and each p_j is just less than a power of 2 (making the fractional part as large as possible). Under these constraints there are a manageable number of cases to test manually. The largest deviation from $\lfloor \log_2 n \rfloor$ is -7 and first occurs at

$$n = 3 \cdot 11 \cdot 13 \cdot 53 \cdot 59 \cdot 61 \cdot 229 \cdot 241 \cdot 251 \approx 1.13 \cdot 10^{15}.$$

Additionally, the first occurrence of -8 is at

$$n = 3 \cdot 13 \cdot 47 \cdot 53 \cdot 59 \cdot 61 \cdot 229 \cdot 239 \cdot 241 \cdot 251 \approx 1.16 \cdot 10^{18},$$

which means this algorithm will need to be slightly modified to reach that value.

To show (2) is true, observe that

$$\begin{aligned}\sum_{j=1}^{k-1} \lfloor \log_2 p_j \rfloor | 1 &\leq \sum_{j=1}^{k-1} \lfloor \log_2 p_j \rfloor + k - 1 \\ &\leq \log_2 n + k - 1 - \log_2 p_k \\ &\leq \lfloor \log_2 n \rfloor - 7 + (k + 6 - \log_2 \sqrt{n}\,).\end{aligned}$$

Now

$$\log_2 \sqrt{n} = \sum_{j=1}^{k} \log_2 \sqrt{p_j},$$

and $\log_2 \sqrt{p_j} > 2$ for $j \geq 7$. This leaves only a finite number of cases where $k + 6 < \log_2 \sqrt{n}$ might be false. Checking (2) manually on each of these cases confirms its validity. □

Finally, this sieving algorithm can be segmented into blocks small enough for a computer to store all generated data in RAM. Using a block size B that is a divisor of x, compute $\mu(n)$ and $M(n)$ for all $(j-1)x/B + 1 \leq n \leq jx/B$ and let j span from 1 to B. For each block, only the primes up to $\sqrt{jx/B}$ need to be considered.

3. Combinatorial algorithm

To compute $M(x)$ at an isolated value, just as in [1] and [2], start with the identity

$$\sum_{n \leq x} M(\lfloor x/n \rfloor) = 1.$$

Observing $\lfloor x/n \rfloor$ takes on roughly $2\sqrt{x}$ distinct values, let $\nu_x = \lfloor \sqrt{x}\,\rfloor$, $\kappa_x = \lfloor x/(\nu_x + 1) \rfloor$ and rewrite the identity as

$$\begin{aligned}\sum_{n \leq \kappa_x} M(x/n) &= 1 - \sum_{n \leq \nu_x} \left(\left\lfloor \frac{x}{n} \right\rfloor - \left\lfloor \frac{x}{n+1} \right\rfloor \right) M(n) \\ &= 1 + \kappa_x M(\nu_x) - \sum_{n \leq \nu_x} \left\lfloor \frac{x}{n} \right\rfloor \mu(n).\end{aligned}$$

From an implementation standpoint, the second line is more cache friendly since the values of μ can be stored in an array of bytes. Moreover, when $\mu(n) = 0$, the quotient it is multiplied by does not need to be computed.

For any $\nu_x < u < x$ define

$$S(y, u) = 1 - \sum_{y/u < n \leq \kappa_y} M(y/n) + \kappa_y M(\nu_y) - \sum_{n \leq \nu_y} \left\lfloor \frac{y}{n} \right\rfloor \mu(n),$$

which gives

$$\sum_{n\le x/u} M(x/n) = S(x,u).$$

Applying generalized Möbius inversion yields the following result.

Theorem 3.1.

$$M(x) = \sum_{n\le x/u} \mu(n) S(x/n,u).$$

Now notice when computing this summand for all $n \le x/u$, only the *square-free* n needs to be considered, as $\mu(n) = 0$ otherwise. This means that about $1 - 6/\pi^2 \approx 39\%$ of summands need not be computed.

To find each sum within each S, a segmented sieve can be applied to compute all required values of μ and M. The time complexity of this algorithm is thus the time spent sieving plus the time computing each $S(x/n,u)$. This gives a total time complexity of

$$O\left(u^{1+\varepsilon} + \sum_{n\le x/u} \nu_{x/n}\right) = O(u^{1+\varepsilon} + x/\sqrt{u}\,).$$

The choice of $u = O(x^{2/3+\varepsilon})$ minimizes this runtime complexity at $O(x^{2/3+\varepsilon})$. When performing the sieve, a sieving block size of $O(\sqrt{u}\,)$ can be used to obtain space complexity of $O(x^{1/3+\varepsilon})$.

4. Analytic algorithm

The bounds on $\liminf q(x)$ and $\limsup q(x)$ can be extended using the approach of Odlyzko and te Riele in [5], which begins with the following observation.

Theorem 4.1 (Tichmarsh [6]). *Assuming the Riemann hypothesis and all zeros of the zeta function are simple, then for $x > 0$,*

$$M(x) = \sum_{i=1}^{\infty}\left(\frac{x^{\rho_i}}{\rho_i\zeta'(\rho_i)} + \frac{x^{\overline{\rho_i}}}{\overline{\rho_i}\zeta'(\overline{\rho_i})}\right) + R(x) + \sum_{n=1}^{\infty} \frac{(-1)^{n-1}(2\pi/x)^{2n}}{(2n)!n\zeta(2n+1)}.$$

Here $R(x) = -2$ for $x \notin \mathbb{Z}$, $R(x) = -2 + \mu(x)/2$ for $x \in \mathbb{Z}$, and ρ_i is the ith nontrivial zero of ζ with positive imaginary part.

Grouping terms in this formula gives

$$q(x) = 2\sum_{i=1}^{\infty} a_i \cos(\gamma_i \log x + \psi_i) + O(x^{-1/2}), \tag{3}$$

where $a_i = 1/|\rho_i\zeta'(\rho_i)|$, $\gamma_i = \mathrm{Im}(\rho_i)$, and $\psi_i = \arg(\rho_i\zeta'(\rho_i))$. Now defining $f(t) = (1-t)\cos(\pi t) + \sin(\pi t)/\pi$ and

$$h(y,N) = 2\sum_{i=1}^{N} a_i f\left(\gamma_i/\gamma_N\right) \cos(\gamma_i y + \psi_i),$$

the following holds.

Theorem 4.2 (Ingham [7]). *For any real y and any positive integer N,*

$$\liminf q(x) \leq h(y,N) \leq \limsup q(x).$$

One should note that unlike Theorem 4.1, this theorem does not assume the Riemann hypothesis. Additionally this is the main result that the analytic algorithm depends on. Roughly speaking, a trick to bound $q(x)$ is hence finding a y and N such that $|h(y,N)|$ is large. Moreover, since $\sum_i a_i$ diverges and $f(t) > 0$ for $0 < t < 1$, if all $\gamma_i y + \psi_i$ were close to multiples of 2π, then $h(y,N)$ could be an arbitrarily large positive number. Similarly if all $\gamma_i y + \psi_i + \pi$ were close to multiples of 2π, then $h(y,N)$ could be an arbitrarily large negative number.

More explicitly, for any sequence of integers m_i where $\gamma_i y + \psi_i - 2\pi m_i$ is sufficiently small, $h(y,N)$ can be approximated with

$$\begin{aligned} h(y,N) &\approx 2\sum_{i=1}^{N} a_i \cos(\gamma_i y + \psi_i) \\ &= 2\sum_{i=1}^{N} a_i \cos(\gamma_i y + \psi_i - 2\pi m_i) \\ &\approx 2\sum_{i=1}^{N} a_i - \sum_{i=1}^{N} \left(\sqrt{a_i}(\gamma_i y + \psi_i - 2\pi m_i)\right)^2. \end{aligned}$$

This means if m_i were found such that each $\sqrt{a_i}(\gamma_i y + \psi_i - 2\pi m_i)$ is small, $h(y,N)$ should be large. This can be achieved *via* lattice reduction. Lattice reduction takes in a basis of integer vectors and returns a new integer basis spanning the same space, where each vector has a small Euclidean norm. Fixing N, the initial basis is

$$\begin{pmatrix} -\lfloor \sqrt{a_1}\psi_1 2^\nu \rfloor \\ -\lfloor \sqrt{a_2}\psi_2 2^\nu \rfloor \\ \vdots \\ -\lfloor \sqrt{a_N}\psi_N 2^\nu \rfloor \\ 2^\nu N^4 \\ 0 \end{pmatrix}, \begin{pmatrix} \lfloor \sqrt{a_1}\gamma_1 2^{\nu-10} \rfloor \\ \lfloor \sqrt{a_2}\gamma_2 2^{\nu-10} \rfloor \\ \vdots \\ \lfloor \sqrt{a_N}\gamma_N 2^{\nu-10} \rfloor \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} \lfloor 2\pi\sqrt{a_1} 2^\nu \rfloor \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

$$\begin{pmatrix} 0 \\ \lfloor 2\pi\sqrt{a_2} 2^\nu \rfloor \\ \vdots \\ 0 \\ 0 \\ 0 \end{pmatrix}, \dots, \begin{pmatrix} 0 \\ 0 \\ \vdots \\ \lfloor 2\pi\sqrt{a_N} 2^\nu \rfloor \\ 0 \\ 0 \end{pmatrix},$$

where ν is any integer satisfying $2N \leq \nu \leq 4N$.

Since $2^\nu N^4$ is much larger than every other element and no other vector has a nonzero $(N+1)$st component, there should be exactly one reduced vector with a nonzero $(N+1)$st term and it will equal $\pm 2^\nu N^4$. Call this vector $v = (v_1, v_2, \dots, v_{N+2})^\intercal$ and without loss of generality assume $v_{N+1} = 2^\nu N^4$.

For each $1 \le i \le N$, this vector has components

$$v_i = z\lfloor \sqrt{a_i}\gamma_i 2^{\nu-10} \rfloor - \lfloor \sqrt{a_i}\psi_i 2^\nu \rfloor - m_i \lfloor 2\pi \sqrt{a_i} 2^\nu \rfloor$$

for some integers $z, m_1, m_2, \dots, m_N$. Now because v_{N+1} is so large these terms should be small, which means

$$\sqrt{a_i}(\gamma_i z/2^{10} - \psi_i - 2\pi m_i)$$

will also be small. Hence setting $y = z/2^{10}$ should give a value where $h(y, N)$ is large and positive, where the value z is known, as $z = v_{N+2}$.

To find a y that makes $h(y, N)$ large and negative, simply replace ψ_i with $\psi_i + \pi$ in the call to the lattice reduction algorithm.

Finally, to improve results, the zeros ρ_i can by sorted by a_i, rather than sorted by γ_i as was done above. This will ensure the largest a_i's will have their corresponding cosines near ± 1, making the sum even larger.

5. Implementation details

5.1. **Sieve.** When performing the sieve in section 2, the bottleneck is accounting for multiples of small prime powers, i.e., 2, 3, 2^2, etc. To circumvent this, these values can be presieved. This implementation is presieved with multiples of 2, 3, 2^2, 5, 7, 3^2, and 11. To do this the sieve was applied, only using these numbers, on an array of length $2 \cdot 2 \cdot 3 \cdot 3 \cdot 5 \cdot 7 \cdot 11 = 13860$. When the main sieve was called, the array m was assembled by joining many copies of this precomputed array.

Because computing all 10^{16} values of M at once would have required storing an array too large for RAM, the segmented version of the sieve was used. Computations were done in blocks of length 8 728 473 600, and used roughly 46 GB of RAM. During the main loop of the sieve, each block was further divided into smaller blocks to allow m to fit in the L3 cache. However, once the size of the primes became substantially larger than the length of m, too much time was spent iterating over primes that were never used. To address this, the length of m was increased and no longer fit in the L3 cache. After each block was computed, each value of $M(n)$ was recorded if it was an extremum, zero, or if n was a multiple of 10^8.

Finally, when identifying elements that correspond to a multiple of p or p^2 in the sieve, integer division is required and is very costly. A way around this is to use methods described in [8], which turns integer division into one 128 bit multiplication, one addition, and two bit shifts. This requires precomputing two constants for each denominator used in the scheme.

5.2. **Combinatorial.** To compute $M(x)$, the value $u = \lceil 0.5x^{2/3} \rceil$ was chosen since it gave the fastest results. This means that when computing $M(2^{73})$, each $M(n)$ for all $n \le 3.5 \cdot 10^{14}$ were computed through a segmented sieve. During this sieving process a block size of roughly $96\sqrt{2u}$ was chosen, giving a total of about $0.0073\sqrt{u}$ blocks to sieve through. Once a block of μ and M values were computed, they were accounted for in each $S(x/n, u)$. Therefore all $S(x/n, u)$ were computed once the sieve finished.

Computing all S as stated in section 3 requires $O(x^{2/3})$ integer divisions, and this is extremely costly. Fortunately when computing a value of S, both sequences of quotients that appear have the same numerator and each denominator successively increments by 1. This means all successive quotients y/n with $\sqrt[3]{2y} \leq n \leq \sqrt{y}$ can be computed using a Bresenham style method. This scheme computes a quotient based off the value of the previous quotient, and is described in detail in [9]. For all denominators $n < \sqrt[3]{2y}$, the same technique used in the sieve to turn a quotient into a multiplication, addition, and bit shifts can be employed [8]. The precomputing of constants for this method requires exactly one quotient to be computed per denominator. This reduces the number of integer divisions from $O(x^{2/3})$ to $O(x/u) = O(x^{1/3})$.

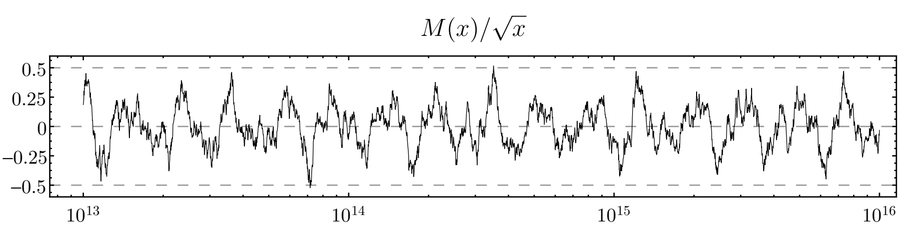
5.3. **Analytic.** Computing bounds on $q(x)$ requires many digits of ρ_i and $\zeta'(\rho_i)$ and a fast lattice reduction routine. Mathematica was used to compute ρ_i to 10 000 digits of precision for all $i \leq 14\,400$ and subsequently compute each $\zeta'(\rho_i)$ to roughly 8151 digits of precision. The results were verified using PARI/GP.

The lattice reduction library chosen was fplll [10]. Its implementation has a runtime complexity of $O(N^{4+\varepsilon}\nu(N+\nu))$, which is faster than the original algorithm's runtime complexity of $O(N^{6+\varepsilon}\nu^3)$ [11]. For each call to fplll, the optional parameter values $(\delta, \eta) = (0.9999, 0.99985)$ were used. The choices of these parameters were intended to speed up the runtime, with the tradeoff of a less optimal solution.

5.4. **Hardware.** The computations of ρ_i and $\zeta'(\rho_i)$ were performed on a 360 core cluster on the Wrangler system at the Texas Advanced Computing Center. All other computations were run on a 2.7 GHz 12-core Intel Xeon E5 processor with a 32 MB L3 cache and 64 GB of RAM. The code was compiled with g++ and where possible, routines were parallelized using OpenMP.

6. Results

6.1. **Sieve.** Computing $M(n)$ for all $n \leq 10^{16}$ took roughly 7.5 months and was heavily influenced by cache misses. The frequency of these misses increased with n. For comparison, the first 10^{14} values took 1 day to compute, the next 10^{14} values took 1.35 days, and this gradually increased until the final 10^{14} values took 2.8 days. Results were periodically verified throughout the computation using the algorithm described in [2] to compute $M(n)$ and compare values. No discrepancies were found.



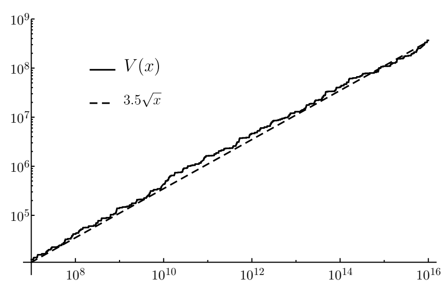
The largest absolute values $M(n)$ attain for $n \leq 10^{16}$ are $-35\,629\,003$ and $40\,371\,499$, and the largest absolute values $q(n)$ attain in this interval are -0.525 and 0.571. Below is a select list of extrema corresponding to prominent peaks of M:

n	$M(n)$	$q(n)$	n	$M(n)$	$q(n)$
6631245058	-31206	-0.383	5197159385733	-689688	-0.303
7766842813	50286	**0.571**	10236053505745	1451233	0.454
15578669387	-51116	-0.410	21035055623987	-1740201	-0.379
19890188718	60442	0.429	21036453134939	-1745524	-0.381
22867694771	-62880	-0.416	23431878209318	1903157	0.393
38066335279	-81220	-0.416	30501639884098	-1930205	-0.349
48638777062	76946	0.349	36161703948239	2727852	0.454
56808201767	-87995	-0.369	36213976311781	2783777	0.463
101246135617	-129332	-0.406	**71578936427177**	-4440015	**-0.525**
108924543546	170358	0.516	146734769129449	3733097	0.308
148491117087	-131461	-0.341	175688234263439	-5684793	-0.429
217309283735	-190936	-0.410	212132789199869	5491769	0.377
297193839495	207478	0.381	212137538048059	5505045	0.378
330508686218	-294816	-0.513	304648719069787	-5757490	-0.330
402027514338	271498	0.428	351246529829131	9699950	0.518
661066575037	331302	0.407	1050365365851491	-13728339	-0.424
1440355022306	-368527	-0.307	1211876202620741	16390637	0.471
1653435193541	546666	0.425	2458719908828794	-20362905	-0.411
2087416003490	-625681	-0.433	3295555617962269	18781262	0.327
2343412610499	594442	0.388	3664310064219561	-23089949	-0.381
3270926424607	-635558	-0.351	4892214197703689	24133331	0.345
4098484181477	780932	0.386	**6287915599821430**	**-35629003**	-0.449
5191164528277	-668864	-0.294	**7332940231978758**	**40371499**	0.471

All zeros of $M(n)$ for $n \leq 10^{16}$ were recorded. A natural question to ask is for any x, how many zeros are less than x? Defining $V(x)$ to be the number of zeros less than x, a theorem of Landau [12] states $V(x) = \Omega(\log x)$. This, however, is expected to be a weak lower bound.

Treating $M(n)$ as a random walk with probability of staying stationary $1 - 6/\pi^2$ and with both probabilities of moving up and down $3/\pi^2$, it would follow that $V(x) = \sqrt{\pi x/3} + o(\sqrt{x}\,)$. In practice, however, M cannot be modeled as a random walk because there is regularity, e.g., $M(4n+3) = M(4n+4)$, etc. Nonetheless, the data suggest $V(x) = \Theta(x^{1/2+\varepsilon})$. In fact $3.5\sqrt{x}$ or even $\sqrt{x}\log\log x$ seem like good approximations.

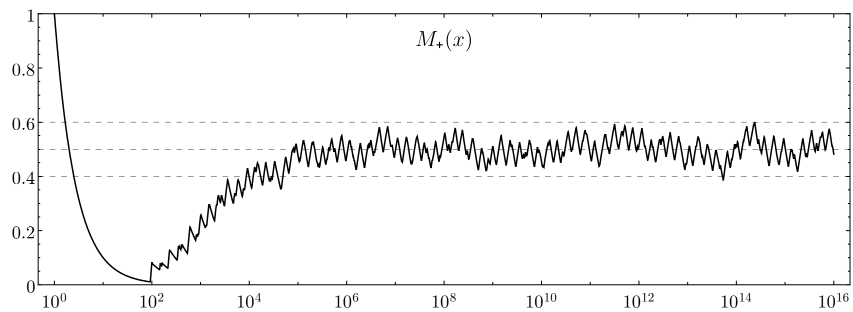
n	$V(10^n)$	n	$V(10^n)$
1	1	9	141121
2	6	10	431822
3	92	11	1628048
4	406	12	4657633
5	1549	13	12917328
6	5361	14	40604969
7	12546	15	109205859
8	41908	16	366567325



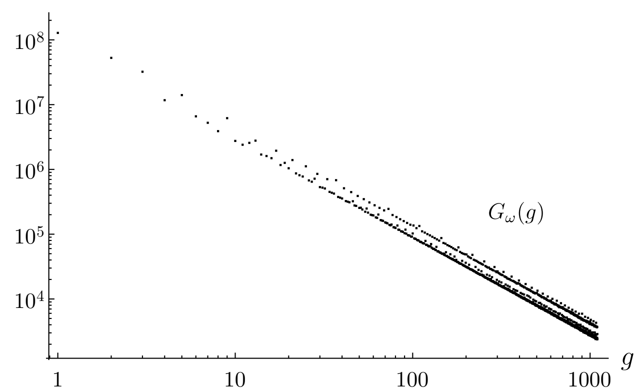
A property these zeros can help investigate is whether M tends to have a bias towards being positive or negative. Define $M_+(x)$ to be the percentage of $M(n)$ that are positive for $n \leq x$, that is,

$$M_+(x) = \frac{1}{x} \sum_{\substack{n \leq x \\ M(n) > 0}} 1.$$

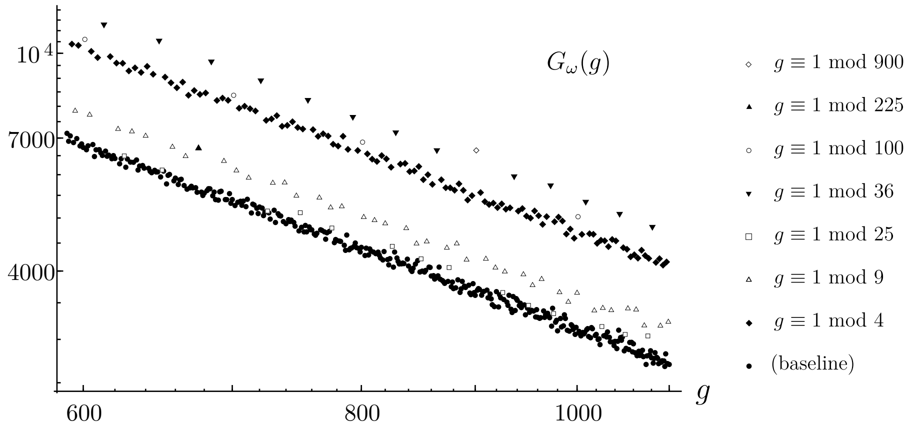
A direct consequence of work by Ng [13] is that under certain conjectures the average value of $M_+(x)$ should be $1/2$, i.e., no bias should exist. Computing μ at each zero of M, the sign of M can be determined between consecutive zeros which can be used to compute M_+. For $x \le 10^5$ there is a clear negative bias, but for $10^5 \le x \le 10^{16}$ there is no longer any apparent bias. For $10^5 \le x \le 10^{16}$ the extreme values are $M_+(53\,961\,131\,760\,658) \approx 0.385$ and $M_+(238\,469\,701\,201\,412) \approx 0.601$.



Another characteristic of the zeros worth consideration is the gap between two consecutive zeros. To examine these gaps, let $G_m(g)$ be the number of gaps of length g that occur for the first m zeros. For a fixed value of m, this function can be plotted to show how the number of gaps of certain lengths vary. Letting $\omega = V(10^{16}) = 366\,567\,325$, gives the following plot:



As seen above, there are distinct bands present and each looks to roughly follow a power law, all with the same exponent. Zooming in, it appears each band is represented by all g congruent to 1 modulo a product of distinct primes squared.



Defining $b_m(g)$ to be the baseline band (which can be approximated by a power law) and P_g to be the set of all primes p where $g \equiv 1 \bmod p^2$, it seems these bands are expressed with the multiplier

$$G_m(g) = \left(\sum_{S \in \mathcal{P}(P_g)} \prod_{p \in S} \frac{1}{p^2 - 2} \right) b_m(g).$$

For example, if $g_0 \equiv 1 \bmod 4$ and $g_0 \not\equiv 1 \bmod p^2$ for $p \neq 2$, then $P_{g_0} = \{2\}$ and $G_m(g_0)$ should be above $b_m(g_0)$ by a multiplicative factor of 3/2.

6.2. **Combinatorial.** Calculating $M(x)$ at powers of two scaled roughly as $O(x^{2/3})$, i.e., $M(2^{x+1})$ was about $2^{2/3} \approx 1.59$ times slower to compute than $M(2^x)$. However, as in the sieve above, cache misses became more frequent for larger x resulting in scale factors around 1.63. The results are as follows:

n	$M(2^n)$	n	$M(2^n)$	n	$M(2^n)$	n	$M(2^n)$	time (s)
0	1	19	-125	38	38729	57	51885062	236.02
1	0	20	257	39	-135944	58	-15415164	374.60
2	-1	21	-362	40	101597	59	-89014828	594.65
3	-2	22	228	41	15295	60	-48425659	943.63
4	-1	23	-10	42	-169338	61	220660381	1494.41
5	-4	24	211	43	259886	62	-248107163	2378.21
6	-1	25	-1042	44	-474483	63	580197744	3815.14
7	-2	26	329	45	1726370	64	-851764249	6263.46
8	-1	27	330	46	-3554573	65	809210153	10376.5
9	-4	28	-1703	47	-135443	66	-1220538763	17235.2
10	-4	29	6222	48	3282200	67	-925696220	28404.4
11	7	30	-10374	49	1958235	68	2092394726	46429.7
12	-19	31	9569	50	-1735147	69	-3748189801	75680.8
13	22	32	1814	51	6657834	70	9853266869	123189
14	-32	33	-10339	52	-13927672	71	-12658250658	200574
15	26	34	-3421	53	-11901414	72	9558471405	326068
16	14	35	8435	54	48662015	73	-6524408924	529127
17	-20	36	38176	55	-48361472			
18	24	37	-28118	56	23952154			

The correctness of the implementation was verified in 3 ways:

- Tests on many already known values were run.
- When computing $M(x)$, $M(x/128)$ was simultaneously computed.
- Formula (3) was used to estimate the first couple digits of $M(x)$ and its order of magnitude.

All values were found to agree.

6.3. **Analytic.** The results of deriving bounds on $q(x)$ can be summarized with the following theorem.

Theorem 6.1. *The function* $q(x) = M(x)/\sqrt{x}$ *has bounds*

$$\liminf q(x) < -1.837625$$

and

$$\limsup q(x) > 1.826054.$$

Proof. To derive these bounds, the lattice reduction algorithm covered in section 4 was run with inputs $\nu = 17\,000$ and $N = 800$. Both calls took roughly 35 days to finish, giving y values

$$y_{-} \approx 1.50546 \cdot 10^{5096} \quad \text{and} \quad y_{+} \approx -2.58842 \cdot 10^{5097},$$

where their exact values can be found in the appendix below. Evaluating $h(y_{\pm}, 14400)$ gives the extreme values

$$h(y_{-}, 14400) \approx -1.837625 \quad \text{and} \quad h(y_{+}, 14400) \approx 1.826054. \qquad \square$$

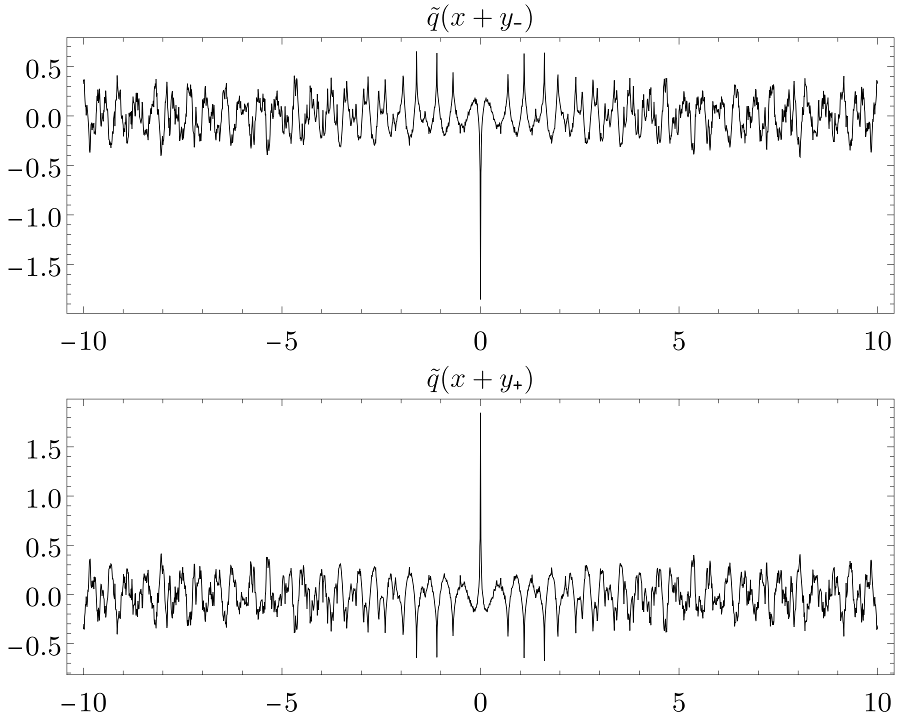
In addition, the lattice reduction algorithm was run on various choices of smaller ν and N. These establish some weaker bounds:

ν	N	y	$h(y, 14400)$	time (d)
5000	400	$-2.78367 \cdot 10^{1493}$	1.61230	0.53
12000	600	$-5.19605 \cdot 10^{3594}$	−1.76011	7.32
12000	600	$9.31709 \cdot 10^{3594}$	1.76382	7.33
15000	700	$2.74696 \cdot 10^{4495}$	−1.81111	19.00
15000	700	$9.69908 \cdot 10^{4495}$	1.81252	18.99
17000	800	$1.50546 \cdot 10^{5096}$	−1.83762	35.07
17000	800	$-2.58842 \cdot 10^{5097}$	1.82605	35.09

Finally, an approximate formula can be used to visualize what $q(x)$ might look like in the neighborhood of $y_{\pm}$. Defining

$$\tilde{q}(x) = 2 \sum_{i=1}^{14400} a_i \cos(\gamma_i x + \psi_i)$$

and assuming $\tilde{q}(x) \approx q(e^x)$ gives plots about these extreme values:



The observation made in [14] that the width of the peaks of $q(x)$ remain constant with respect to $\log x$ seems to hold this far out. Moreover, as seen in the above figures, these peaks appear to be anomalies, as most peaks in the vicinity of $y_\pm$ do not exceed 0.5 in absolute value.

7. Extensions and concluding remarks

7.1. **Sieve.** Computing $M(n)$ for all $n \leq 10^{16}$ took about 7.5 months and the time was dominated by cache misses. To systematically compute $M(n)$ for say $n \leq 2 \cdot 10^{16}$, the cache misses beyond 10^{16} would grow substantially more frequent, causing a drastic slow down. To reduce the number of these misses, additional measures can be taken.

First, rather than storing each value $\mu(n)$ in 1 byte, 4 values of $\mu(n)$ can be encoded together since $\mu(n)$ only takes on 3 possible values, allowing it to be expressed with 2 bits. A similar approach can be taken for $M(n)$ too, but not for $n > 10^{16}$. For computations on shorter intervals though, space can still be saved. For example, $M(n)$ can be stored as a signed 16 bit integer as long as $|M(n)| < 2^{15}$. The first time this inequality is violated is at $n = 7\,613\,644\,886$. Similarly, $M(n)$ can be stored as a signed 24 bit integer for all $n < 348\,330\,855\,359\,510$.

A more robust solution to prevent cache misses is to employ an additional data structure. Recall that during the sieve the array built to store values of μ is segmented into blocks small enough to fit into the L3 cache. However, once the primes being iterated over become too large, much time is wasted iterating over primes that are not used. Currently, this is mitigated by using larger blocks, but these larger blocks no longer fit in the L3 cache. Instead, this problem could be resolved by the following algorithm.

```
Create a hashmap h that maps an integer to a vector of integers
for each prime p ≤ √x do
    Find the first block i with an index corresponding to a multiple of p
    If h(i) is uninitialized, set h(i) ← {p}
    Otherwise append p to the vector h(i)
for each block i do
    for each p in h(i) do
        Sieve block i with p as normal
        Determine the next block j in which p will be used
        If h(j) is uninitialized, set h(j) ← {p}
        Otherwise append p to the vector h(j)
    Clear h(i)
```

Under this approach, the block size can be set to always fit in the L3 cache without having the overhead of iterating over primes that will never be used. Notice here that each prime p will only be present in h at most once. Hence the size of h is only dependent on the number primes used, not the number of blocks being iterated over. For an L3 cache similar in size to the one used, this method could help make it feasible to compute beyond 10^{16}.

7.2. **Combinatorial.** Isolated values of $M(x)$ were computed at powers of 2 up to $M(2^{73}) = -6524408924$, which took roughly 6 days to calculate. At the time this paper was written, to the author's knowledge, there are no known combinatorial identities that lead to a runtime complexity less than $O(x^{2/3+\varepsilon})$. However, a speedup could still potentially be obtained with a combinatorial approach.

Recall that the identity used in the algorithm and stated in Theorem 3.1 is

$$M(x) = \sum_{n \le x/u} \mu(n) S(x/n, u).$$

Since $\mu(n)$ will asymptotically be zero $1 - 6/\pi^2 \approx 39\%$ of the time, one approach could be to look for a sum whose summand is zero more often than this. The closest identity the author found in literature is due to Benito and Varona [15] and is

$$M(x) = \frac{1}{2} \sum_{n \le x/u} f^{-1}(n) G(x/n, u),$$

where

$$\begin{aligned} G(y,u) = -3 + &\sum_{y/u < n \le \kappa_y} (h(n) - h(n-1)) M(y/n) + h(\nu_y) M(\kappa_y) \\ &+ \sum_{n \le \nu_y} \left(3 \left\lfloor \frac{n}{3k} \right\rfloor - 2 \left\lfloor \frac{n-k}{2k} \right\rfloor \right) \mu(n), \end{aligned}$$

and $f^{-1}(n)$ is the Dirichlet inverse of $f(n) = h(n-1) - h(n)$, and

$$h(n) = \begin{cases} 2 & \text{if } n \equiv 0 \mod 6, \\ 0 & \text{if } n \equiv 1 \text{ or } 2 \mod 6, \\ 1 & \text{if } n \equiv 3 \text{ or } 4 \mod 6, \\ -1 & \text{if } n \equiv 5 \mod 6. \end{cases}$$

It turns out that $f^{-1}(n)$ is zero just as often as $\mu(n)$ with the added advantage that $f^{-1}(2) = f^{-1}(4) = 0$, meaning 2 of the 4 most computationally expensive summands need not be computed. The drawback is that no efficient way of computing $f^{-1}(n)$ was found.

Finally, an analytic approach could be considered. In 1987 Lagarias and Odlyzko [16] described a way to compute $\pi(x)$, the number of primes $\leq x$, in $O(x^{1/2+\varepsilon})$ time. The algorithm uses a completely different approach, expressing $\pi(x)$ in terms of a contour integral in the complex plane. Moreover, the discussion section in [16] states that the same ideas can be applied to compute $M(x)$ in the same time complexity.

In 2010, Platt computed $\pi(x)$ using this algorithm and stated the combinatorial algorithm for $\pi(x)$ would probably be faster until roughly $x \approx 4\cdot 10^{31}$. This is due to overhead, some of which is from the need of multiple precision complex arithmetic [17]. It seems likely the analytic algorithm for $M(x)$ would follow suit.

7.3. **Analytic.** It has been shown $\liminf q(x) < -1.837625$ and $\limsup q(x) > 1.826054$. Extending these bounds further, with the same approach, would take a considerable amount of time. To see why, first notice all values found with fplll, using $(\delta, \eta) = (0.9999, 0.99985)$, resulted in bounds about 95.5% of the optimum for a given N, i.e.,

$$h \approx 1.91 \sum_{i=1}^{N} a_i.$$

Additionally, the runtime of fplll's algorithm scales as $O(N^{4+\varepsilon}\nu(N+\nu))$ [10]. Thus given the timings of previous calls and assuming ν scales linearly with N, these observations can help estimate what is needed to reach a given bound:

bound	estimated N	estimated time
1.90	865	2 months
1.95	985	5 months
2.00	1125	10 months

It therefore appears attaining bounds of ± 2 is within reach with existing hardware and algorithms. Attaining bounds larger than 2 will most likely need ρ_i and $\zeta'(\rho_i)$ computed to higher precision than what was achieved here, or different (δ, η) values. At present, a different approach is likely needed to substantially improve these bounds past 2.

8. Appendix

Access all computed data in a Mathematica notebook at

`https://wolfr.am/mertens`.

Acknowledgments

The author wishes to thank the Texas Advanced Computing Center for providing the computing power to compute ρ_i and $\zeta'(\rho_i)$ to such high precision and Michael Trott for referring him to the TACC. Additionally, a thanks goes out to Matthew Gelber and Eric Rowland for offering suggestions and edits throughout the writing process of this paper. Finally, the author wishes to acknowledge Daniel Fortunato for his collaborations during the inception of this project.

References

[1] E. Kuznetsov, *Computing the Mertens function on a GPU* (arXiv:1108.0135v1 [math.NT], 31 Jul 2011)

[2] M. Deléglise and J. Rivat, *Computing the summation of the Möbius function*, Experiment. Math. **5** (1996), no. 4, 291–295. MR1437219

[3] T. Kotnik and J. van de Lune, *Further systematic computations on the summatory function of the Möbius function* Report MAS-R0313, CWI Amsterdam (November 2003)

[4] D. G. Best and T. S. Trudgian, *Linear relations of zeroes of the zeta-function*, Math. Comp. **84** (2015), no. 294, 2047–2058, DOI 10.1090/S0025-5718-2014-02916-5. MR3335903

[5] A. M. Odlyzko and H. J. J. te Riele, *Disproof of the Mertens conjecture*, J. Reine Angew. Math. **357** (1985), 138–160, DOI 10.1515/crll.1985.357.138. MR783538

[6] E. C. Titchmarsh, *The Theory of the Riemann Zeta-Function*, Oxford, at the Clarendon Press, 1951. MR0046485

[7] A. E. Ingham, *On two conjectures in the theory of numbers*, Amer. J. Math. **64** (1942), 313–319, DOI 10.2307/2371685. MR0006202

[8] T. Granlund and P. Montgomery, *Division by Invariant Integers using Multiplication* Proceedings of SIGPLAN '94 Conference on Programming Language Design and Implementation.

[9] Richard Sladkey, *A Successive Approximation Algorithm for Computing the Divisor Summatory Function* (arXiv:1206.3369 [math.NT], Jun 2012) 13–16

[10] The FPLLL development team, *fplll, a lattice reduction library* (available at `https://github.com/fplll/fplll`) (January 2016)

[11] I. Morel, D. Stehlé, and G. Villard, *H-LLL: using Householder inside LLL*, ISSAC 2009—Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation, ACM, New York, 2009, pp. 271–278, DOI 10.1145/1576702.1576740. MR2742770

[12] E. Grosswald, *Oscillation theorems of arithmetical functions*, Trans. Amer. Math. Soc. **126** (1967), 1–28, DOI 10.2307/1994409. MR0202685

[13] N. Ng, *The distribution of the summatory function of the Möbius function*, Proc. London Math. Soc. (3) **89** (2004), no. 2, 361–389, DOI 10.1112/S0024611504014741. MR2078705

[14] T. Kotnik and J. van de Lune, *On the order of the Mertens function*, Experiment. Math. **13** (2004), no. 4, 473–481. MR2118272

[15] M. Benito and J. L. Varona, *Recursive formulas related to the summation of the Möbius function*, Open Math. J. **1** (2008), 25–34, DOI 10.2174/1874117700801010025. MR2461512

[16] J. C. Lagarias and A. M. Odlyzko, *Computing $\pi(x)$: an analytic method*, J. Algorithms **8** (1987), no. 2, 173–191, DOI 10.1016/0196-6774(87)90037-X. MR890871

[17] D. J. Platt, *Computing $\pi(x)$ analytically*, Math. Comp. **84** (2015), no. 293, 1521–1535, DOI 10.1090/S0025-5718-2014-02884-6. MR3315519

Wolfram Research Inc., 100 Trade Center Drive, Champaign, Illinois 61820

E-mail address: `ghurst@wolfram.com`