ANALYTIC COMPUTATION OF THE PRIME-COUNTING FUNCTION

BY

WILLIAM FLOYD GALWAY

B.A., University of Utah, 1976
M.S., University of Utah, 1984

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Mathematics
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2004

Urbana, Illinois

Abstract

The main topic of this dissertation is the Lagarias-Odlyzko analytic algorithm for computing $\pi(x)$—the number of primes less than or equal to x. This algorithm is asymptotically the fastest known algorithm for the computation of $\pi(x)$. It uses numerical integration of a function related to the Riemann zeta function, $\zeta(s) := \sum_{n=1}^{\infty} n^{-s}$, in combination with a summation involving a "kernel" function evaluated at prime powers near x.

Our work resolves many issues left untreated in the original paper by Lagarias and Odlyzko and makes several original contributions—some of which have applications in other areas. In this dissertation we

- introduce a kernel function which appears to be more effective than one suggested by Lagarias and Odlyzko;
- perform a careful analysis of various sources of truncation error;
- give choices of parameters which bound the truncation error while keeping computation to a minimum;
- develop two new methods for enumerating primes in intervals, which require much less memory than previously known sieving methods and which are much faster than methods which test primality of single numbers;
- describe a new method for computing $\zeta(s)$ which gives more accurate values with less complexity than classical methods.

In Honor of my Father
Floyd I. Galway
and to the memory of my Mother
Desma H. Galway

Acknowledgments

I'm grateful for the advice, support, encouragement, and friendship of the many individuals—and of a few organizations—who contributed to this dissertation. In many cases I give further details on their contributions in the acknowledgments at the ends of individual chapters. I have also tried to place entries for each contributor in the **Index and Glossary** which begins on page 163.

First, I thank my advisor: Harold G. Diamond, who has been an unflagging guide. His support and his excellent mathematical judgment–especially his judgment of when a result is good-enough for the task at hand–have helped me to express myself mathematically. I also thank Harold and his wife Nancy for graciously hosting me, twice, as a houseguest during the final stages of completing this dissertation.

I also thank the other members of my dissertation committee: Bruce Berndt, Adolf Hildebrand, Andrew Odlyzko, and Frank Stenger.

Professor Berndt and Professor Hildebrand were always happy to discuss issues as they arose during my research. Some of their specific contributions are detailed in the Acknowledgments for Chapters 5 and 6. Professor Hildebrand also advised me on some of the finer points of working with LaTeX.

Both before and after joining my committee, Andrew Odlyzko always found time in his busy schedule to advise me on fine points of the algorithms which formed the starting point for this dissertation: the Lagarias-Odlyzko analytic algorithm for computing $\pi(x)$; the Meissel, Lehmer, Lagarias, Miller, Odlyzko (extended Meissel-Lehmer) algorithm for computing $\pi(x)$; and the Odlyzko-Schönhage algorithm for computing $\zeta(s)$.

Long before he joined my committee, Professor Stenger was always happy to discuss the questions I posed to him on numerical analysis. Although the analyses in this dissertation are self contained, and I rarely cite Professor Stenger's work, I believe that the approach to numerical quadrature used here fits under the rubric of "sinc methods"—a field which Professor Stenger is largely responsible for developing.

Both Jeff Lagarias and Victor Miller gave advice, historical background on algorithms, and encouragement for this research.

Dan Bernstein introduced me to some of the key ideas leading to my development of the dissected sieve, as described in Chapter 5. Carl Pomerance provided the inspiration to do the research leading to the hybrid sieve, described in Chapter 6.

Richard Crandall gave several suggestions, which I have used, on the computation of special functions. He also asked probing questions about preliminary versions of this work, and I hope that my exposition has been improved in responding to those questions.

Some of the research described here was completed while I was supported as a fellow at the Pacific Institute for Mathematical Sciences (PIMS) at Simon Fraser University in Burnaby, British Columbia, Canada.

During the research for and the writing of this dissertation I made heavy use of computers. I thank the many system administrators and their staff who have supported the smooth operation of those resources: Robert Zeh, Mike Hollyman, Carl Freeland, Jason Hoos, Ken Hamer, David Fiske, Bob Berry, Mary Stevens, Jennifer Shannon, Jonathan Manton, and Patrick Szuta at the University of Illinois; and Brent Kearney at the Pacific Institute for Mathematical Sciences.

Sabrina Hwu and Teresa Johnson, at the Center for Reliable and High Performance Computing, University of Illinois Urbana-Champaign, provided information on contemporary computer architectures and cache memories.

Shortly after the death of my mother, while I was preparing to re-enter the academic world, my friend John Anderson encouraged and assisted me in my application for entrance to the University of Illinois.

My wife, Miriam Paschetto, has supported me both emotionally and financially. She also did the bulk of the packing to move our household first to the Pacific Coast, and later to the Atlantic Coast. I thank her for her time, work, and support, and for both her patience and her impatience as I neared the completion of this work.

Most of the calculations reported here were performed using programs written in the C programming language. I also used the *Mathematica*® software system[1] and the PARI/GP package developed by Henri Cohen, Karim Belebas, and many other collaborators. PARI/GP is currently available at `http://pari.math.u-bordeaux.fr/`.

[1] *Mathematica* is a registered trademark of Wolfram Research, Inc.

The layout of this dissertation is defined by a LaTeX document class which I developed from David Hull's `uiucthesis.sty`. Algorithms are presented using Carsten Heinz's **listings** package for LaTeX.

Many of the figures in this dissertation were created using *Mathematica*. In many cases the figures were further annotated using the Unix utility `xfig` and/or LaTeX's `PSfrag` package.

Some of the computations described in Chapter 6 made use of the Condor system[2] for "high throughput computing" on a distributed system of computers.

Finally, I thank the following friends, colleagues, and correspondents: Hiro Asari, David Bailey, Rob Ballantyne, Michael Bennett, Robert Bennion, Maarten Bergvelt, Jonathan Borwein, Peter Borwein, David Bradley, Richard Brent, Jared Bronski, Jeff and Sharon Brueggeman, Imin Chen, Stephen Choi, Darrin Doud, Helaman Ferguson, Gilbert Gosseyn, Gergely Harcos, Steve Harding, Martin Huxley, François Morain, Richard Pinch, Ekkehart Vetter, Ulrike Vorhauer, Ron Weeden, and Alexandru Zaharescu.

[2]The Condor Software Program (Condor) was developed by the Condor Team at the Computer Sciences Department of the University of Wisconsin-Madison.

Contents

List of Tables

List of Figures

1 Introduction

1.1 Background

The function $\pi(x)$, (Figure 1.1) counts the number of primes up to x. More specifically,

$$\pi(x) := \sum_{p \leq x} 1,$$

where the sum is taken over primes.

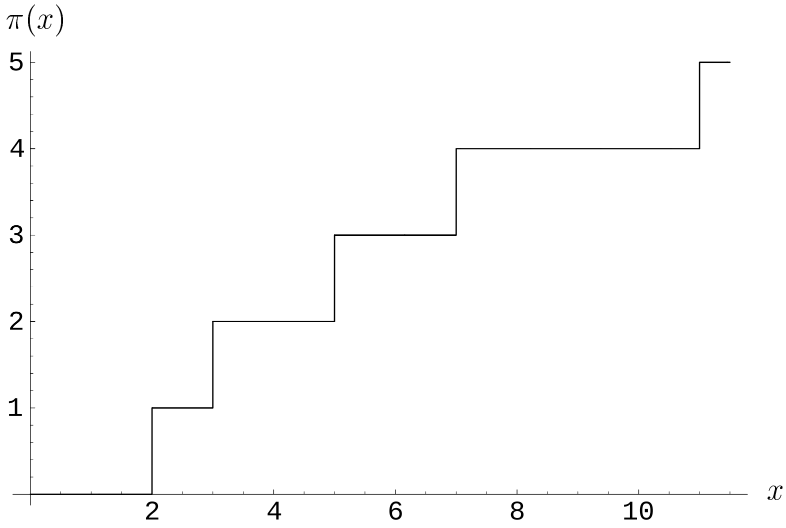


Figure 1.1: The function $\pi(x)$, $0 \leq x \leq 11.5$

In 1982 [LO84] Jeff Lagarias and Andrew Odlyzko described two new algorithms for computing $\pi(x)$. One method, their "analytic" algorithm, uses numerical integration of a function related to the Riemann zeta function, $\zeta(s) := \sum_{n=1}^{\infty} n^{-s}$, in combination with summation of a function evaluated at prime powers "near" x. ("Prime powers" means all numbers of the form p^m, p prime, $m \geq 1$.) In a later paper, after Odlyzko and A. Schönhage [OS88] had developed a fast method for computing $\zeta(s)$, Lagarias and Odlyzko showed that their analytic algorithm could compute $\pi(x)$ in $x^{1/2+\epsilon}$ time and $x^{1/4+\epsilon}$ space [LO87]. (We give more precise definitions of "time" and "space" in Section 1.5.)

The other method of Lagarias and Odlyzko—the extended Meissel-Lehmer method—with later improvements by Marc Deléglise and Joel Rivat, computes $\pi(x)$ in $O(x^{2/3}/\ln^2 x)$ time and $O(x^{1/3}\ln^3(x)\ln\ln(x))$ space [DR96a]. Lagarias, Miller, and Odlyzko used the original version of this algorithm to compute values of $\pi(x)$ up to $\pi(4\cdot 10^{16})$ [LMO85]. Deléglise and Rivat improved the algorithm and continued computations up to $\pi(10^{18})$, and more recently to $\pi(10^{20})$ [DR96a, DR96b].

The extended Meissel-Lehmer algorithm has been further improved by Xavier Gourdon [Gou01a]. His most important improvement has been to modify the algorithm so that it is well suited to distributed computation, with little communication needed between processors. Gourdon has implemented an ongoing project of computing $\pi(x)$ using resources provided by several contributors on the Internet. As of March 2001, computation had proceeded through $\pi(4\cdot 10^{22})$ [Gou01b]. While attempting to compute $\pi(10^{23})$, Gourdon found an error in his program—two slightly different computations gave two different values for $\pi(10^{23})$. As of early 2004, $\pi(4\cdot 10^{22})$ appears to be the "record computation" of $\pi(x)$.

Until the research of this dissertation, little work has gone towards carefully analyzing the analytic algorithm—other than the few papers by Lagarias and Odlyzko cited above, and an analysis given by Ekkehart Vetter in his Diplomarbeit [Vet91, Chapter 2]. Although the analytic method has better asymptotic running time than other methods, the O-constant is expected to be large.

The following crude argument suggests that it is only for very large values of x that the analytic algorithm is faster than the extended Meissel-Lehmer algorithm. Ignoring factors of x^ϵ we assume that the running time for the extended Meissel-Lehmer algorithm has the form $a_1 x^{b_1}$, and that the running time for the analytic algorithm has the form $a_2 x^{b_2}$. This implies that the latter will be faster for $x \geq (a_2/a_1)^{1/(b_1-b_2)}$. If we assume that $b_1 = 2/3$ and that $b_2 = 1/2$, this implies a crossover point at $x = (a_2/a_1)^6$. However, fitting the timing data given in [DR96a] gives $b_1 \approx 0.625$ (Figure 1.2). If we further assume (arbitrarily, but perhaps realistically) that $b_2 \approx 0.525$ near the crossover point, we find crossover near $x = (a_2/a_1)^{10}$.

Without an implementation it is difficult to estimate what a_2/a_1 might be, but values from 100 to $10,000$ seem plausible, implying crossover in the range $10^{12} \leq x \leq 10^{40}$. In any case, it is clear that a careful implementation is

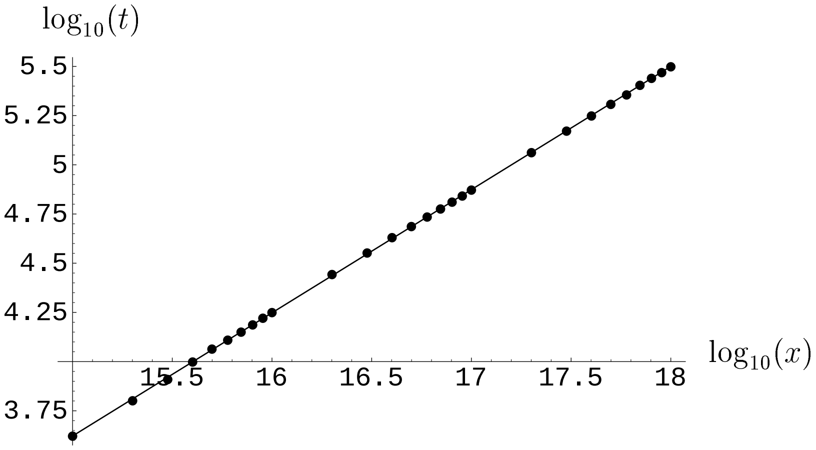


Figure 1.2: Log-Log fit to Deléglise-Rivat timings
$\log_{10}(t) = -5.768 + 0.626 \log_{10}(x)$
(t in seconds)

required to bring the crossover point into a practical range. This dissertation attempts to give nearly optimal solutions for the tasks that must be solved to produce an implementation.

1.2 Notational conventions

Many of our notational conventions are outlined in Table 1.1 on the following page. Although most of our notation is standard, there are a few fine points and nonstandard conventions which we describe below.

Given $f(z)$ and $g(z)$, possibly complex valued, we write $f(z) = O(g(z))$ (equivalently, $f(z) \ll g(z)$, $g(z) \gg f(z)$) if there is a constant $C > 0$ (an *O-constant*) such that $|f(z)| \leq C\,|g(z)|$ over some domain (that should be clear from the context). We write $f(z) \asymp g(z)$ to denote that $f(z)$ and $g(z)$ are of the same order, i.e., $f(z) \ll g(z)$ and $f(z) \gg g(z)$. A subscript in the notation indicates a variable that the constant depends upon. For example, $f(z) = O_\rho(g(z))$ means that there is a function $C(\rho)$, independent of z, for which $|f(z)| \leq C(\rho)\,|g(z)|$.

We use the nonstandard notation $f(z) = C\,O(g(z))$ to specify an explicit O-constant. Our meaning is that $|f(z)| \leq C\,|g(z)|$ for the given C. We write $f(z) = h(z) + O(g(z))$ to mean $f(z) - h(z) = O(g(z))$, and similarly

Notation	Meaning
$\blacksquare$	Terminates a proof, or an argument supporting a conjecture.
$\square$	Terminates a definition or remark.
$:=$	Equality defining the left side.
$=:$	Equality defining the right side.
$f(x)\vert_a^b$	Denotes $f(b) - f(a)$. Also written $f(x)\vert_{x=a}^{x=b}$.
$\lfloor x \rfloor$	Greatest integer $\leq x$.
$\lceil x \rceil$	Least integer $\geq x$.
$x \bmod b$	The value $x - b\lfloor x/b \rfloor$. The expression $x \bmod b = y$ often implies a computation yielding y as its value. In contrast, $x \equiv y \pmod b$ states that a congruence holds, without implying computation.
$\vert\mathcal{S}\vert$	Cardinality of a set or sequence $\mathcal{S}$.
$\overline{z}$	Complex conjugate of $z \in \mathbb{C}$.
$\mathrm{Re}(z)$	Real part of $z \in \mathbb{C}$.
$\mathrm{Im}(z)$	Imaginary part of $z \in \mathbb{C}$.
$\ln(z)$	Natural logarithm of z.
$\mathrm{Ln}(z)$	Principal branch of $\ln(z)$, satisfies $-\pi < \mathrm{Im}\,\mathrm{Ln}(z) \leq \pi$, $e^{\mathrm{Ln}(z)} = z$.
$\log_b(x)$	Logarithm of x to the base b, $\log_b(x) = \ln(x)/\ln(b)$.
$\mathbb{N}$	The set of strictly positive integers, $\mathbb{N} = \{1, 2, \ldots\}$.
$\mathbb{Z}_0$	The set of non-negative integers, $\mathbb{Z}_0 = \{0, 1, \ldots\}$.
$\mathcal{P}$	The set of prime numbers.
p, ℓ	Unless stated otherwise, $p \in \mathcal{P}$ and $\ell \in \mathcal{P}$. E.g., $\sum_p \ldots$ denotes a sum over all primes.
ϵ	A positive number that can be taken arbitrarily close to zero. Not to be confused with ε.
ε	Denotes a fixed error bound, $\varepsilon > 0$. We often write $z + \varepsilon\, O(1)$ to denote a floating point approximation to a number, as discussed in Section 1.4.
§	Indicates a section number in citations, as in "§10.11.II".
$\mathrm{sgn}(x)$	The "sign" or "signum" function, $\mathrm{sgn}(x) := -1, 0, 1$ for $x < 0$, $x = 0$, $x > 0$, respectively.
$\displaystyle\sum_{n\leq x}{}' \ldots$	Sum with a weighting of $1/2$ if $n = x$. Similarly for $\displaystyle\sum_{p\leq x}{}' \ldots$ and for $\displaystyle\sum_{p^m\leq x}{}' \ldots$.

Table 1.1: Notational Conventions

$f(z) = h(z) + C\,O(g(z))$ means $f(z) - h(z) = C\,O(g(z))$. In particular,

$$f(z) = h(z) + \varepsilon\,O(1) \tag{1.1}$$

means that $h(z)$ approximates $f(z)$ with an error bounded by ε.

Given $g(z)$, possibly complex valued, which satisfies $g(z) \neq 0$ in a neighborhood of z_0, we write $f(z) \sim g(z)$ as $z \to z_0$ to mean $\lim_{z\to z_0} f(z)/g(z) = 1$, and write $f(z) = o(g(z))$ as $z \to z_0$ to mean $\lim_{z\to z_0} f(z)/g(z) = 0$. We also write $f(z) = h(z) + o(g(z))$ if $f(z) - h(z) = o(g(z))$. In many cases the z_0 should be clear from the context, and will not be specified.

For some functions, say $f(x)$ to be specific, we will only give definitions valid when $x > 0$. In such cases we let $f(0) := \lim_{\delta\to 0+} f(\delta)$ whenever the limit is well defined.

Many of our functions, such as $\phi(u; x, \lambda)$, depend on a "variable" u and other "parameters" x, λ, etc., which will typically be fixed while u varies. In such cases we may write $\phi(u)$ for $\phi(u; x, \lambda)$, with the understanding that the parameters remain fixed.

The integrals used in this dissertation can be taken as Riemann-Stieltjes integrals. Although most of our sums and integrals converge absolutely, some are only conditionally convergent. To ensure that these are well defined, we use the conventions that

$$\sum_{n\in\mathbb{Z}} \cdots := \sum_{n=-\infty}^{\infty} \cdots := \lim_{N\to\infty} \sum_{n=-N}^{N} \cdots,$$
$$\int_{-\infty}^{\infty} \cdots\, dt := \lim_{T\to\infty} \int_{-T}^{T} \cdots\, dt.$$

Integrals of the form $\int_0^\infty \ldots\, du$ are best thought of as the improper integral $\int_{-\infty}^{\infty} \ldots\, d\tau$, under the change of variables $u := e^\tau$.

Empty sums and products (sums and products with no terms) always denote the additive or multiplicative identity, respectively. For example, $\sum_{k=1}^{0} k = 0$ while $\prod_{k=1}^{0} k = 1$.

The **Index and Glossary** beginning on page 163 serves as a guide to notation introduced in other sections of this dissertation.

1.3 Algorithmic notation

Much in the same way that we number theorems, definitions, etc., we identify our algorithms with a numeric label of the form $C.N$, where C denotes the chapter where the algorithm is defined, and N numbers the algorithm within the chapter. We also give each algorithm an alphanumeric name. Algorithm 1.1 (`ExampleErfc`), on page 10, illustrates this convention. All the algorithms in this dissertation are indexed under their name, and under the entry for "Algorithms", in the **Index and Glossary**.

Algorithms are presented in a mixture of mathematical notation and notation based on the C and C++ computer languages.

We emphasize important conditions or truths with **assert** statements: which assert that a predicate holds without implying any computation. We start short comments with "//" which run to the end of the line. Longer comments may be interspersed between lines of an algorithm, as in the comments following line 2 of Algorithm 1.1 (`ExampleErfc`).

Assignment is denoted by $\leftarrow$, and x++ is shorthand for $x \leftarrow x + 1$. We may use assignments as values—for example to assign a value to n and then compare that value against x_2 we could use the expression $(n \leftarrow u_1^2 + u_2^2) \leq x_2$.

We write && for the conditional conjunction of boolean expressions. That is, in evaluating `expr1` && `expr2` the second subexpression is evaluated only if `expr1` is `TRUE`, ultimately yielding the logical *and* of the two subexpressions. For unconditional conjunction (logical *and*) we write `expr1` $\wedge$ `expr2`.

We use the standard **while** and **for** looping constructs of the C language and also use nonstandard instances of **for** as illustrated below:

for $(n \leftarrow 1\,; n \leq 10\,; n{+}{+})$ ... // *Standard* **for** *loop.*
for $(1 \leq n \leq 10)$ ... // *Shorthand for the line above.*
for $(p \in [2, 100] \cap \mathcal{P})$ ... // *Loop over primes p,* $2 \leq p \leq 100$.

Of course, in addition to these looping constructs, we also use traditional mathematical notation such as $\sum_{m=1}^{M} \cdots$ and $\prod_{m=1}^{M} \cdots$ to indicate sums and products taken over a range of integers.

Without complete consistency, we follow some rules-of-thumb concerning the scope of variables and other names used in our algorithms. Variables passed as arguments can be assumed local to an algorithm, as can most other variables used within the algorithm. Names of algorithms such as `ExampleErfc` are treated as global, as are standard functions such as $\sin(z)$

and well-known constants such as $\pi = 3.14159\ldots$.

We do not use a rigorously typed language—instead, we use a variety of methods to indicate the type of a variable when we consider its type to be important. When we do not indicate the type it is because we believe the type to be either obvious or immaterial. Methods for indicating type include comments at the start of an algorithm, the choice of typeface used for a variable, or a *declaration* such as $x \in \mathbb{Z}$.

Without going into details of their representation, we assume that integers may have any magnitude and are represented exactly. Given $z \in \mathbb{Z}$ we call z an n-bit integer provided $|z| < 2^n$. It follows from this convention that an n-bit integer is also an N-bit integer for any $N \geq n$. We assume that $z \in \mathbb{Z}$ can be represented using $O(\ln(2 + |z|))$ bits, from which it follows that an n-bit integer can be represented using $O(1 + n)$ bits.

We assume that a rational number, say x/y, is represented by a pair $x \in \mathbb{Z}$, $y \in \mathbb{N}$, not necessarily coprime. (We rarely use rationals, and the requirement that $\gcd(x, y) = 1$ is unnecessary for our purposes.)

Unless they are obviously elements of $\mathbb{Z}$ or $\mathbb{Q}$, real numbers are assumed to be approximated by floating point numbers, following conventions which are detailed in the next section, and which are defined as follows:

Definition 1.1 Given $z \in \mathbb{R}$, we say that z is a *floating point* number provided $z = 2^E M$, $E \in \mathbb{Z}$, $M \in \mathbb{Z}$.

We say that the pair $E \in \mathbb{Z}$, $M \in \mathbb{Z}$ define an n-bit floating point representation of $z = 2^E M$ provided $|M| < 2^n$ and $|E| = O(n)$. We do *not* require that $|M| \geq 2^{n-1}$. The O-constant in the bound on $|E|$ will remain unspecified but it could be deduced by careful analysis of the algorithms presented in this dissertation. □

Remarks Instead of saying that z has an n-bit floating point representation, for ease of writing we are often willing to confuse z with its representation and will say that z is an n-bit floating point number if it has such a representation.

Assuming that an n-bit floating point number $z = 2^E M$ is represented by the pair E, M, it follows that z requires $O\big(\ln(2+|E|+|M|)\big) = O(1+n)$ bits for its representation. We do not require that floating point numbers fit within the precision of any particular hardware, but instead allow arbitrary precision, following conventions spelled out in Section 1.4. □

As would be expected, unless they are obviously elements of $\mathbb{R}$, complex numbers are assumed to be approximated by numbers of the form $x + iy$, with x and y represented as floating point numbers. We call $z = x + iy$ an n-bit (complex) number if both x and y are n-bit floating point numbers.

1.4 Floating point numbers and error analysis

Given arbitrary $z \in \mathbb{R}$ or $z \in \mathbb{C}$, we make heavy use of the notation $z+\varepsilon\, O(1)$ to denote a number which differs from z by an amount bounded by ε in absolute value.

The "error term" denoted by $\varepsilon\, O(1)$ serves two, slightly different, purposes. One use is as a bookkeeping tool to document the accumulation of errors during a computation. These errors may arise both from truncation error, such as the error due to approximating integrals and infinite sums as finite sums; and from roundoff error, which is the error due to approximating arbitrary numbers with finite-precision floating point numbers.

When describing computations executed within an algorithm—as opposed to discussing computations within an **assert** statement, or within a theorem (say)—the inclusion of $\varepsilon\, O(1)$ in a sum has the further meaning of indicating the rounding of the sum to a certain number of bits of precision. More specifically, given $z \in \mathbb{R}$ we assume that a computed result of the form $z + \varepsilon\, O(1)$ denotes $2^E M$, with $E := \lfloor \log_2(\varepsilon) \rfloor$ and with M chosen so that $\left|2^E M - z\right| < 2^E$, and thus $\left|2^E M - z\right| < \varepsilon$.

In this setting, note that we have $\left|2^E M - z\right|$ strictly less than ε. It follows that after executing a statement like

$z_1 \leftarrow (z + \varepsilon\, O(1)) + \varepsilon\, O(1)\,;$

we can conclude that $z_1 = z + \varepsilon\, O(1)$, and not just the weaker conclusion that $z_1 = z + 2\varepsilon\, O(1)$. More generally, when we round an already rounded expression to another precision we have

$$\big(z + \varepsilon_1\, O(1)\big) + \varepsilon_2\, O(1) = z + \max(\varepsilon_1, \varepsilon_2)\, O(1). \tag{1.2}$$

In the case of complex numbers, given $z = x + iy$ we assume that a computed result of the form $z + \varepsilon\, O(1)$ denotes the rounded result

$$\left(x + \frac{\varepsilon}{2} O(1)\right) + i\left(y + \frac{\varepsilon}{2} O(1)\right),$$

where the real and imaginary parts are floating point numbers.

Although not strictly necessary, when $z + \varepsilon\, O(1)$ denotes a floating point number we often use ε of the form $\varepsilon = 2^E$ both to emphasize that z is approximated by a floating point number, and because ε of this form are themselves easily represented as floating point numbers.

By the conventions of Definition 1.1 on page 7, and the subsequent discussion, it follows that $z + \varepsilon\, O(1)$ is an $O(\ln(2 + |z/\varepsilon|))$-bit number—both for $z \in \mathbb{R}$ and for $z \in \mathbb{C}$; and provided that $\varepsilon \ll \ln(2 + |z/\varepsilon|)$, with an O-constant for the bound which is related in an obvious way to the O-constant mentioned in Definition 1.1. Our complexity analyses of algorithms will depend on these estimates of the number of bits in the representations of numbers, as explained further in Section 1.5.

In our presentation of algorithms we assume that all arithmetic operations are computed exactly and then rounded as indicated. In particular, we assume that operations are exact when they yield integers, rationals, or boolean values. This assumption extends to calculations which involve results outside the realm of $\mathbb{Z}$ or $\mathbb{Q}$. For example, we assume that boolean expressions such as $x_1^{1/m} \le p \le x_2^{1/m}$ are evaluated unambiguously provided x_1, x_2, m and p are in $\mathbb{N}$. Similarly, given $z = 2^E M$ we assume that expressions such as $\lceil \log_2(z) \rceil$ are computed exactly.

Our convention that computations are rounded from exact results implies what happens when we sum several rounded quantities, namely that

$$\sum_k (z_k + \varepsilon_k O(1)) = \sum_k z_k + \left(\sum_k \varepsilon_k \right) O(1), \tag{1.3}$$

under the standard assumption that all $\varepsilon_k > 0$. Computations such as those illustrated in (1.2) or (1.3) need not occur within a single expression—they would yield the same result even if the computations were spread over several statements in an algorithm. Note that Equation (1.3) would certainly not hold if our algorithms used floating-point arithmetic as implemented in contemporary hardware. We can assume that it holds since we assume that floating-point arithmetic, as defined in this section, adjusts the number of bits of precision as needed to ensure that our assumptions are met.

Note that our convention that computations are performed exactly and then rounded allows considerable leeway in the amount of detail we give in specifying how a computation is performed. For example, we may write $\pi + 2^{-32}O(1)$ and leave it to an implementor to determine the details of how such a computation would be done. Perhaps less reasonably, we could also write something like

$$\varepsilon\, O(1) + \sum_{k\geq 0} \sqrt{k}\, \pi^{-k},$$

and again let the implementor determine how the rounded result could be found.

Algorithm 1.1 (`ExampleErfc`), below, illustrates many of the concepts discussed here and in the previous section. The algorithm implements a standard formula for $\operatorname{erfc}(z)$ from [AS92, Entry 7.1.5], which can be stated as $\operatorname{erfc}(z) = 1 - 2S_\infty/\sqrt{\pi}$, where

$$S_\infty := \sum_{k\geq 0} \frac{(-1)^k z^{2k+1}}{(2k+1)k!}. \tag{1.4}$$

Algorithm 1.1 (`ExampleErfc`: Approximate $\operatorname{erfc}(z)$)
Given $z \in \mathbb{R}$, $|z| \leq 1/2$, and ε, $0 < \varepsilon < 1$, returns $\operatorname{erfc}(z) + \varepsilon\, O(1)$.

1 **ExampleErfc** $(z \in \mathbb{R}, \varepsilon)$ {

2 **assert** $0 < \varepsilon < 1$;

Store the sign of z in s and then negate z if $z < 0$. This ensures an alternating series, which simplifies error analysis.

3 **if** $(z \geq 0)$ $s \leftarrow 1$; **else** $\{s \leftarrow -1;\ z \leftarrow -z;\}$

Subdivide allowable error amongst two sources of error: truncation error and roundoff error. We bound each by $\varepsilon_0 := \varepsilon/2$.

4 $\varepsilon_0 \leftarrow \varepsilon/2$;

Since $|z| \leq 1/2$ and $\varepsilon < 1$ it is easy to show that K bounds the total number of terms accumulated in line 12 and line 14, below.

5 $K \leftarrow \lceil \log_2(1/\varepsilon)/2 \rceil$;

We use ε_1 to determine the precision(s) to which we round some results. Our choice of ε_1 is explained further in the **Remarks** *below.*

6 $\varepsilon_1 \leftarrow \varepsilon_0/(2K)$; $\varepsilon_1 \leftarrow 2^{\lfloor \log_2(\varepsilon_1) \rfloor}$;

7 $t \leftarrow z$;

8 $S \leftarrow 0$; // *S serves to approximate S_∞.*

9 **for** $(k \leftarrow 0; |t|/(2k+1) > \varepsilon_0; k{+}{+})$ {

10 // *The following assertion is easily proved by induction, using $|z| < 1/2$.*

```
11      assert  t = (−1)^k z^{2k+1}/k! + ε_1 O(1) ;
12      S ← S + t/(2k + 1) + ε_1 O(1) ;
13      t ← −zt/(k + 1) + ½ ε_1 O(1) ; }
14    S ← S + t/(2k + 1) + ε_1 O(1) ;
15    assert  S = S_∞ + (ε/2) O(1) ;
```

Apply the identity $\operatorname{erfc}(-z) = 2 - \operatorname{erfc}(z)$ *as we return our result, recalling that* s *denotes the original sign of* z.

```
16    return  1 − (2/√π + (ε/4) O(1)) sS ; }
```

Remarks The arguments z and ε which are passed to `ExampleErfc` must, of course, be floating point numbers. Although Algorithm 1.1 carefully bounds the error in its computation of $\operatorname{erfc}(z)$, we have not analyzed the error implicit in the possibility that $z = z_0 + \delta$ for some small δ, and that our intention may have been to find $\operatorname{erfc}(z_0)$. To analyze this issue, we could note that $|d/dz \ \operatorname{erfc}(z)| \leq 2/\sqrt{\pi}$, so from Taylor's Theorem it follows that

$$\operatorname{erfc}(z_0 + \delta O(1)) = \operatorname{erfc}(z_0) + \frac{2\delta}{\sqrt{\pi}} O(1)$$

and that

$$\texttt{ExampleErfc}(z_0 + \delta O(1), \varepsilon) = \operatorname{erfc}(z_0) + (2\delta/\sqrt{\pi} + \varepsilon) O(1).$$

In line 6 we choose ε_1 so the total roundoff error accumulated in lines 12–14 is bounded by $\varepsilon_0/2$. It suffices to set $\varepsilon_1 \leftarrow \varepsilon_0/(2K)$, but, as explained above, setting ε_1 to a power of two better characterizes how $\cdots + \varepsilon_1 O(1)$ denotes the act of rounding to a given precision.

To establish that $\texttt{ExampleErfc}(z, \varepsilon)$ *does* return $\operatorname{erfc}(z) + \varepsilon\, O(1)$, it suffices to establish each of the assertions made in the body of the algorithm, and then to establish that the value computed and returned in line 16 is sufficiently accurate. Assuming the validity of the assertion of line 15, the accuracy of the value returned follows, since

$$\begin{aligned} 1 - \left(\frac{2}{\sqrt{\pi}} + \frac{\varepsilon}{4} O(1)\right) S &= 1 - \left(\frac{2}{\sqrt{\pi}} + \frac{\varepsilon}{4} O(1)\right) \left(S_\infty + \frac{\varepsilon}{2} O(1)\right) \\ &= \operatorname{erfc}(z) + \frac{\varepsilon}{4} S_\infty O(1) + \frac{2}{\sqrt{\pi}} \frac{\varepsilon}{2} O(1) + \frac{\varepsilon^2}{8} O(1) \\ &= \operatorname{erfc}(z) + \left(\frac{1}{4} + \frac{1}{\sqrt{\pi}} + \frac{1}{8}\right) \varepsilon\, O(1) = \operatorname{erfc}(z) + \varepsilon\, O(1). \qquad (1.5) \end{aligned}$$

In (1.5) we have first used the fact that $\varepsilon < 1$, and the easily established result that $0 \leq S_\infty < 1$ for $z \geq 0$. A simple calculation establishes that $1/4 + 1/\sqrt{\pi} + 1/8 \leq 1$, and the final result of (1.5) follows. $\square$

1.5 Computational model

To characterize the complexity of our algorithms, we use essentially the same model of computation as that used by Lagarias, Miller and Odlyzko [LMO85], and by Deléglise and Rivat [DR96a]—although those authors focused on (arithmetic) operational complexity while our fundamental measure will be bit complexity, as discussed below. We use a Random Access Machine (RAM) model of computation. A rigorous description of this model of computation is given in [AHU75].

Our measure of complexity is *bit complexity* (*logarithmic cost criterion* in the terminology of [AHU75]). This measure counts the number of bit-operations required to perform a computation. However, our complexity analyses will focus on higher-level operations such as the arithmetic operations of addition and multiplication of integers. As explained below, the corresponding bit complexity is easily derived from a count of arithmetic operations and a bound on the size of the numbers operated on.

A good survey of results on the bit complexity of arithmetic operations is given in Chapters 6, 7, and 10 of [BB87]. Another source discussing the bit complexity of algorithms for arithmetic, especially the bit complexity of multiplication, is [Knu81, §4.3].

Throughout this dissertation we will assume that a fixed, but unspecified, algorithm is used for the multiplication of integers. We let $\mathcal{M}(n)$ denote an upper bound on the number of bit-operations required to multiply two integers of n bits. Following [BB87, §6.4], we also assume that $\mathcal{M}(n)$ is nondecreasing in n, that

$$2\,\mathcal{M}(n) \leq \mathcal{M}(2n) \leq 4\,\mathcal{M}(n), \tag{1.6}$$

and that $\mathcal{M}(n)$ is interpolated consistent with these conditions to be defined for all $n > 0$, $n \in \mathbb{R}$.

In [BB87] the authors show that addition, subtraction, multiplication, and division of n-bit integers all require $O(\mathcal{M}(n))$ bit-operations. Recalling that operations on floating point numbers can be reduced to operations on integers, it is easy to show that these operations on floating point representations of real or complex numbers of n bits also require $O(\mathcal{M}(n))$ bit-operations. Similarly, extraction of mth roots requires $O_m(\mathcal{M}(n))$ bit-operations.

Definition 1.2 Throughout this dissertation, the term *arithmetic operation* and the unqualified term *operation* will mean any operation working on numbers of $O(n)$ bits and requiring $O(\mathcal{M}(n))$ bit-operations, where the value of n will either be stated or will be clear from the context. $\square$

When z is an n-bit number, the elementary transcendental functions: e^z, $\ln(z)$, $\sin(z)$, etc., require $O(\ln(2n)\mathcal{M}(n))$ bit-operations. (We take $\ln(2n)$ rather than $\ln(n)$ to handle the nearly trivial case where $n = 1$.)

We will informally use the term *time* (or *running time*) to mean the number of operations (on numbers, or on bits, as appropriate) performed by an algorithm. Of course, in other settings *time* simply means the number of seconds (say) that the algorithm takes to complete a computation.

We use a "logarithmic" measure for the space used by a program, so all references to the "space" (or "storage") required will be in units of bits. In analogy to the C-language construct `sizeof`, we will write `BitSizeOf`(v) to denote the number of bits of storage required to represent some quantity (or data structure) v.

It should be noted that on contemporary computers the RAM model of computation may give only a crude idea of the actual performance of an algorithm. This is illustrated in our discussion of the timing data presented in Table 5.2 on page 118. The problem of finding more appropriate models of computation is discussed further in [McG01].

As discussed in Section 1.1, we expect the analytic algorithm for $\pi(x)$ to be faster than other methods only when x is very large. To compute $\pi(x)$ past $x = 10^{20}$, or so, in a practical length of time, we will need to use a parallel version of the algorithm. However, we will not discuss parallel models of computation beyond noting without proof that the analytic algorithm is well suited to parallel implementation. In particular, if we use the "hybrid" sieve of Chapter 6, we can efficiently implement the analytic algorithm using as many as $O(x^{1/4})$ processors.

1.6 Properties of the Mellin transform

The analytic algorithm for computing $\pi(x)$ is based on properties of the Mellin transform, which we summarize in this section.

Definition 1.3 Given a function $\phi(u)$, defined for $u > 0$, we say that $\phi(u)$ is of type (α, β) if for $\alpha < \sigma < \beta$ we have $\int_0^\infty |\phi(u)|\, u^{\sigma-1} du < \infty$. □

We mention in passing that a function of type (α, β) is also of type (α_1, β_1) for any $\alpha_1 \geq \alpha$, $\beta_1 \leq \beta$.

Theorem 1.4 *Let $\phi(u)$ be a function of type (α, β) and of bounded variation on finite intervals. For $\alpha < \mathrm{Re}(s) < \beta$ define $\widehat{\phi}(s)$ as*

$$\widehat{\phi}(s) := \int_0^\infty \phi(u) u^{s-1}\, du. \tag{1.7}$$

Then given $u > 0$ and $\alpha < \sigma < \beta$ we have

$$\frac{\phi(u+) + \phi(u-)}{2} = \frac{1}{2\pi i} \int_{\sigma - i\infty}^{\sigma + i\infty} \widehat{\phi}(s) u^{-s}\, ds. \tag{1.8}$$

Proof: See [Hen91, §10.11.II]. ■

Definition 1.5 The function $\widehat{\phi}(s)$ defined by Equation (1.7) is called the *Mellin transform* of ϕ, and we always write $\widehat{f}(s)$ to denote the Mellin transform of a function $f(u)$. Equation (1.8) defines the *inverse Mellin transform* of $\widehat{\phi}$. Together, ϕ and $\widehat{\phi}$ form a *Mellin transform pair.*

Functions like $\phi(u)$ in the integrand of a transformation formula are often called kernel functions—in this dissertation we reserve the term *kernel function* for a function of bounded variation on finite intervals. We also introduce the term *evenly-stepped* function for a function $\phi(u)$ satisfying $\phi(u) = \lim_{\delta \to 0}(\phi(u+\delta) + \phi(u-\delta))/2$. □

In the following discussion we will assume that $\phi(u)$ is of type (α, β) and that s always lies in the range $\alpha < \mathrm{Re}(s) < \beta$. For convenience, throughout the rest of this dissertation we will require our kernel functions to be evenly-stepped.

Provided $\phi(u)$ is evenly-stepped, Theorem 1.4 states that

$$\phi(u) = \frac{1}{2\pi i} \int_{\sigma - i\infty}^{\sigma + i\infty} \widehat{\phi}(s) u^{-s}\, ds. \tag{1.9}$$

Setting $u = n$ in Equation (1.9), then summing over n, suggests the formula

$$\frac{1}{2\pi i}\int_{\sigma-i\infty}^{\sigma+i\infty} \widehat{\phi}(s)\sum_{n\geq 1} a_n n^{-s}\, ds = \sum_{n\geq 1} a_n \phi(n). \tag{1.10}$$

Theorem 1.6 and Corollary 1.7, below, give conditions under which (1.10) holds. (Theorem 1.6 and its proof are based on a similar result in Ekkehart Vetter's Diplomarbeit [Vet91, Satz 2.1].)

Theorem 1.6 *Let $\phi(u)$ be an evenly-stepped kernel function of type (α, β). Let $\sum_{n\geq 1} a_n\phi(n)$ be convergent and let σ_a denote the abscissa of absolute convergence for the Dirichlet series $\sum_{n\geq 1} a_n n^{-s}$. Given fixed $\sigma > \sigma_a$ with $\alpha < \sigma < \beta$, let the convergence of*

$$\int_{\sigma-iT}^{\sigma+iT} \widehat{\phi}(s) n^{\sigma-s}\, ds \tag{1.11}$$

as $T \to \infty$ be uniform in $n \in \mathbb{N}$. Then Equation (1.10) *holds.*

Proof: Given $T > 0$, $n \in \mathbb{N}$, and recalling that σ is fixed, let

$$R(T, n) := n^{\sigma}\left(\phi(n) - \frac{1}{2\pi i}\int_{\sigma-iT}^{\sigma+iT} \widehat{\phi}(s) n^{-s}\, ds\right),$$

so that

$$\frac{1}{2\pi i}\int_{\sigma-iT}^{\sigma+iT} \widehat{\phi}(s) n^{-s}\, ds = \phi(n) - n^{-\sigma} R(T, n). \tag{1.12}$$

Since $\phi(u)$ is an evenly-stepped kernel function, we may apply Equation (1.9) to find that $R(T, n) \to 0$ as $T \to \infty$. Our requirement that (1.11) converges uniformly in n implies there is a function of T, say R_T, such that $|R(T, n)| \leq R_T$ and that $R_T \to 0$ as $T \to \infty$.

Since $\sigma > \sigma_a$, our Dirichlet series converges absolutely, so we can exchange integration with summation and then apply (1.12) to find:

$$\begin{aligned} \frac{1}{2\pi i}\int_{\sigma-iT}^{\sigma+iT} \widehat{\phi}(s)\sum_{n\geq 1} a_n n^{-s}\, ds &= \sum_{n\geq 1} a_n \frac{1}{2\pi i}\int_{\sigma-iT}^{\sigma+iT} \widehat{\phi}(s) n^{-s}\, ds \\ &= \sum_{n\geq 1} a_n\left(\phi(n) - n^{-\sigma} R(T, n)\right) = \sum_{n\geq 1} a_n\phi(n) - \sum_{n\geq 1} a_n n^{-\sigma} R(T, n). \end{aligned} \tag{1.13}$$

We justify the convergence of the rightmost sum in (1.13) by the fact that $|R(T,n)| \leq R_T$ uniformly in n, and so $|a_n n^{-\sigma} R(T,n)| \leq R_T |a_n| n^{-\sigma}$. Thus the rightmost sum converges (absolutely) by the absolute convergence of our Dirichlet series. Further, this implies

$$\left| \sum_{n\geq 1} a_n n^{-\sigma} R(T,n) \right| \leq R_T \sum_{n\geq 1} |a_n| n^{-\sigma},$$

and the theorem follows since $R_T \to 0$ as $T \to \infty$. ■

Corollary 1.7 *Let $\phi(u)$ be an evenly-stepped kernel function of type (α, β). Let $\sum_{n\geq 1} a_n \phi(n)$ be convergent and let σ_a denote the abscissa of absolute convergence for the Dirichlet series $\sum_{n\geq 1} a_n n^{-s}$. Given fixed $\sigma > \sigma_a$ with $\alpha < \sigma < \beta$, Equation* (1.10) *holds provided $\int_{\sigma-i\infty}^{\sigma+i\infty} \widehat{\phi}(s)\, ds$ converges absolutely.*

Proof: Along the path of integration in (1.11) we have $|n^{\sigma-s}| = 1$, so

$$\left| \int_{\sigma-i\infty}^{\sigma+i\infty} \widehat{\phi}(s) n^{\sigma-s}\, ds \right| \leq \int_{-\infty}^{\infty} \left| \widehat{\phi}(\sigma+it) \right| dt < \infty.$$

Thus (1.11) converges uniformly in n, satisfying the one remaining condition needed in Theorem 1.6. ■

1.7 The analytic algorithm

Given fixed $x > 0$, let

$$\chi(u) := \chi(u;x) := \begin{cases} 1 & u < x, \\ \frac{1}{2} & u = x, \\ 0 & u > x. \end{cases} \tag{1.14}$$

It is easy to show that $\chi(u)$ is an evenly-stepped kernel function of type $(0, \infty)$, and that its Mellin transform is $\widehat{\chi}(s) = \widehat{\chi}(s;x) = x^s/s$. It can be shown that $\widehat{\chi}(s;x)$ satisfies the conditions of Theorem 1.6. Thus, for $\sigma > \max(0, \sigma_a)$ we have

$$\frac{1}{2\pi i} \int_{\sigma-i\infty}^{\sigma+i\infty} \frac{x^s}{s} \sum_{n\geq 1} a_n n^{-s}\, ds = \sideset{}{'}\sum_{1\leq n\leq x} a_n. \tag{1.15}$$

Equation (1.15) is often called Perron's formula [Apo76, §11.12].

Recall that for $\mathrm{Re}(s) > 1$ the Riemann zeta function $\zeta(s)$ is defined as $\zeta(s) = \sum_{n\geq 1} n^{-s}$. Euler's product formula for $\zeta(s)$ states that, for $\mathrm{Re}(s) > 1$,

$$\zeta(s) = \prod_p (1 - p^{-s})^{-1},$$

where p runs through all prime numbers. For $\sigma > 1$ define $\ln \zeta(\sigma)$ to be the real branch of the logarithm, and more generally define $\ln \zeta(\sigma + it)$ by

$$\ln \zeta(\sigma + it) = \ln \zeta(\sigma) + \int_{\sigma}^{\sigma+it} \frac{\zeta'(z)}{\zeta(z)}\, dz. \tag{1.16}$$

With this definition of $\ln \zeta(s)$, Euler's product formula gives, for $\mathrm{Re}(s) > 1$,

$$\begin{aligned} \ln \zeta(s) &= -\sum_p \ln(1 - p^{-s}) && (1.17)\\ &= \sum_{n\geq 1} a_n n^{-s}, && (1.18) \end{aligned}$$

where

$$a_n = \begin{cases} 1/m & n = p^m, \\ 0 & \text{otherwise.} \end{cases} \tag{1.19}$$

Let $\pi^*(x) := \sum'_{p^m \leq x} 1/m$ (see Figure 1.3 on the following page). From Equation (1.18) and Equation (1.15) (Perron's formula) it follows that for $x > 0$ and fixed $\sigma > 1$

$$\pi^*(x) = \frac{1}{2\pi i} \int_{\sigma - i\infty}^{\sigma+i\infty} \frac{x^s}{s} \ln \zeta(s)\, ds. \tag{1.20}$$

Equation (1.20) is due to Bernhard Riemann (see [Rie90]), and is the basis for the first proofs of the prime number theorem, which (in one form) states that $\pi(x) \sim x/\ln(x)$ as $x \to \infty$ [Edw74].

For now, to simplify the exposition by avoiding the discontinuities of $\pi^*(x)$, we will assume that x is not a prime power. Under this assumption,

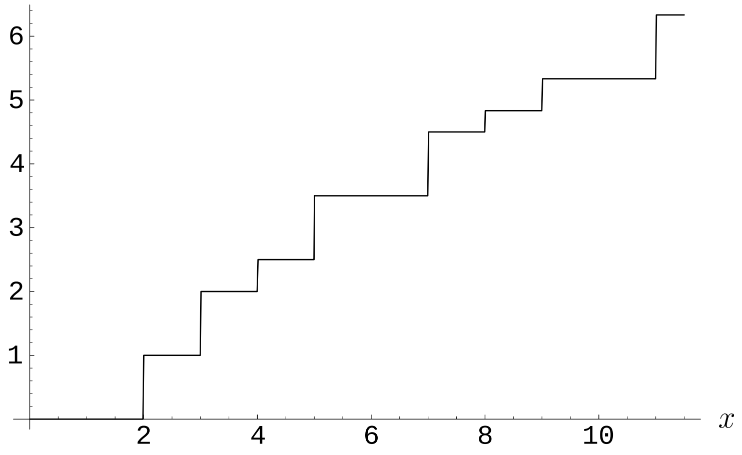


Figure 1.3: The function $\pi^*(x)$, $0 \le x \le 11.5$. Although it is not apparent in the figure, note that $\pi^*(x)$ is evenly-stepped in the sense of Definition 1.5.

$\pi(x)$ is related to $\pi^*(x)$ by

$$\pi^*(x) = \sum_{m\geq 1} \frac{1}{m}\pi(x^{1/m})$$

$$\pi(x) = \pi^*(x) - \sum_{m\geq 2} \frac{1}{m}\pi(x^{1/m}). \tag{1.21}$$

We see that a fast method for approximating $\pi^*(x)$ gives a fast algorithm for $\pi(x)$, since computation of (1.21) requires subtracting from $\pi^*(x)$ a sum with roughly $\log_2(x)$ non-zero terms, and since $\sum_{m\geq 2} \pi(x^{1/m})/m$ may be computed in $x^{1/2+o(1)}$ operations on numbers of $O(\ln(2+|x|))$ bits. Ensuring that the error in the resulting approximation of $\pi(x)$ is less than $1/2$ lets us then round to the nearest integer to find $\pi(x)$ exactly.

Formula (1.20) is unsuited to computation because the integrand approaches zero slowly as $|\mathrm{Im}(s)| \to \infty$. Lagarias and Odlyzko noted that $\pi^*(x)$ could be computed efficiently by choosing a different kernel—designed in part to ensure more rapid convergence. Before describing their method, we first characterize kernel functions which might be used in their method:

Definition 1.8 We say that $\phi(u)$ is a *suitable kernel function* (for the analytic algorithm) provided $\phi(u)$ is an evenly-stepped kernel function of type (α, ∞) for some $\alpha \le 1$ and provided Equation (1.10) holds for $\sigma > 1$ when the a_n in that equation are the Dirichlet coefficients of $\ln \zeta(s)$, as given by

Equation (1.19). □

It follows immediately from Definition 1.8 that given any suitable kernel function $\phi(u)$, for $\sigma > 1$ we have

$$\frac{1}{2\pi i}\int_{\sigma-i\infty}^{\sigma+i\infty} \widehat{\phi}(s)\ln\zeta(s)\,ds = \sum_{p^m}\frac{1}{m}\phi(p^m), \tag{1.22}$$

and this yields the *key formula* for the analytic algorithm:

$$\pi^*(x) = \frac{1}{2\pi i}\int_{\sigma-i\infty}^{\sigma+i\infty} \widehat{\phi}(s)\ln\zeta(s)\,ds + \sum_{p^m}\frac{1}{m}\big(\chi(p^m;x)-\phi(p^m)\big). \tag{1.23}$$

If we choose $\phi(u)$ to closely approximate the step-function $\chi(u;x)$, and also so that $\widehat{\phi}(s)$ damps out quickly as $|\mathrm{Im}(s)| \to \infty$, as in Figure 1.4(b) on the next page, then we may closely approximate (1.23) by truncating both the integral and the sum to reasonably short intervals:

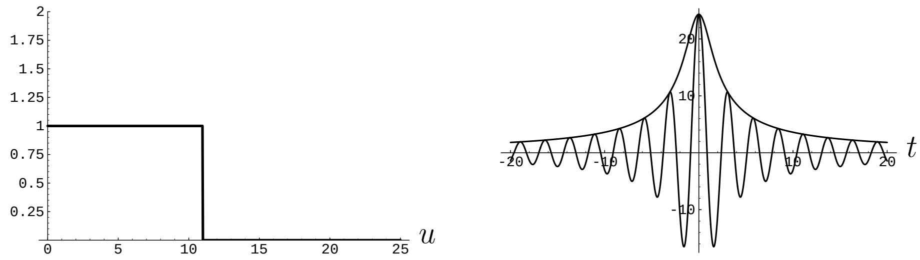
$$\begin{aligned}\pi^*(x) \approx{}& \frac{1}{2\pi i}\int_{\sigma-iT}^{\sigma+iT} \widehat{\phi}(s)\ln\zeta(s)\,ds \\ &+ \sum_{x_1\le p^m\le x_2}\frac{1}{m}\big(\chi(p^m;x)-\phi(p^m)\big).\end{aligned} \tag{1.24}$$

The analytic algorithm approximates the integral in (1.24) by numerical quadrature, and computes the sum over $p^m \in [x_1, x_2]$ by enumerating primes and prime powers.

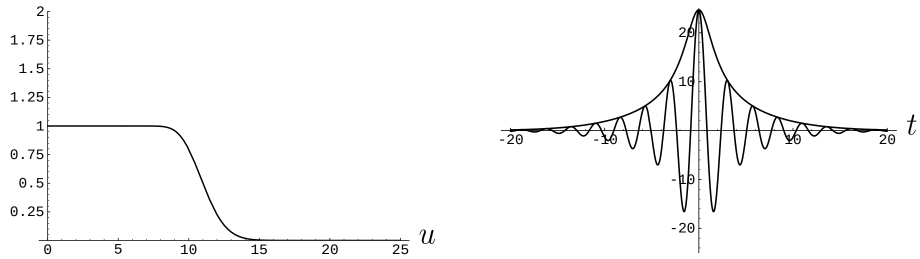
Lagarias and Odlyzko proposed a kernel $\phi(u;x,y,k)$ where $x > y > 0$. They begin with the polynomial $f_k(v) = c_k v^k(1-v)^k$, where c_k is chosen to make $\int_0^1 f_k(v)\,dv = 1$, and then define $g_k(w)$ as $\int_0^w f_k(v)\,dv$, and finally define $\phi(u;x,y,k)$ as

$$\phi(u;x,y,k) = \begin{cases} 1 & u \le x-y, \\ g_k((x-u)/y) & x-y < u < x, \\ 0 & u \ge x. \end{cases}$$

As illustrated in Figure 1.7, the kernel function $\phi(u;x,y,k)$ is a step-like function which drops along a smooth polynomial spline over the middle section $x-y < u < x$. (Smooth in the sense that all derivatives through



(a) $\chi(u)$ and $\widehat{\chi}(s)$



(b) $\phi(u)$ and $\widehat{\phi}(s)$

Figure 1.4: Two Mellin transform pairs. The graphs on the right show real part and modulus as a function of $s = 1.5 + it$.

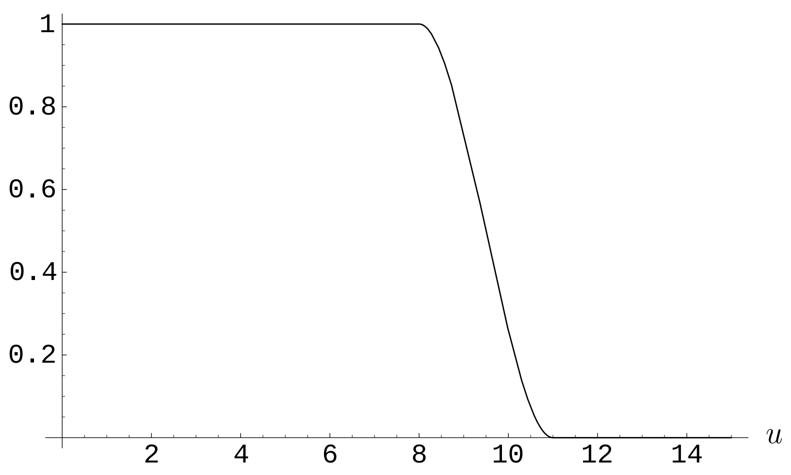


Figure 1.5: The Lagarias-Odlyzko kernel $\phi(u; x, y, k)$. $x = 11$, $y = 3$, $k = 1$

the kth derivative are defined and zero at the endpoints $x - y$ and x.) The parameters k and y are chosen to roughly minimize the running time of the algorithm. A careful analysis of the Lagarias-Odlyzko algorithm using this kernel is given in Ekkehart Vetter's Diplomarbeit [Vet91].

As summarized below, we propose a different kernel and make several other modifications to the algorithm. We also develop several new algorithms to support the implementation of the analytic algorithm.

1.8 Summary of following chapters

In order to minimize the length of the truncated intervals in Formula (1.24), we want a kernel $\phi(u)$ that closely approximates a step-function, and whose Mellin transform $\widehat{\phi}(s)$ approaches zero rapidly as $|\mathrm{Im}(s)| \to \infty$. In Chapter 2, we introduce a different kernel than that used by Lagarias and Odlyzko, which we believe better achieves these goals (as explained in Section 2.5 on page 33). We also restate Formula (1.23) after introducing our kernel along with some associated notation.

In addition to a good kernel, we need good error estimates for the tails of our truncated sum and integral, so that we do not use unnecessarily wide intervals. These error estimates and choices of truncation points are treated in Chapter 3. Chapter 3 also gives our proposed quadrature algorithm, and treats the choice of its associated parameters, such as the step size h.

To compute our sum over prime powers we need an efficient method for enumerating primes. Some form of the sieve of Eratosthenes would seem to be well suited for this, but this would require roughly $\sqrt{x}$ bits of memory to sieve efficiently in the neighborhood of x. In Chapter 4 we give a brief survey of methods for enumerating primes, focusing on the tradeoff between speed and memory needs.

In Chapter 5 we describe a new "dissected" sieving algorithm that reduces the memory requirements to roughly $x^{1/3}$ bits. In Chapter 6 we describe a "hybrid" sieving algorithm that we conjecture requires even less memory. The hybrid sieve works in two phases, using a mixture of sieving to exclude "small" prime factors and a simple probable primality test. An initial phase creates a table of "bad" pseudoprimes which would otherwise remain undetected by either sieving or a probable primality test. The second phase uses this table to enumerate all primes in the interval.

The most time-consuming part of computing our key integral is likely to be the many computations of $\zeta(s)$ required in approximating the integral by a sum. When $\mathrm{Re}(s) = 1/2$ the method of choice for computing $\zeta(s)$ has been to use the Riemann-Siegel formula (an asymptotic expansion). Although Wolfgang Gabcke has derived good error bounds for the Riemann-Siegel formula in the case $\sigma = 1/2$ [Gab79], the analysis becomes significantly more complicated with each additional term included in the expansion, and good bounds are currently unavailable for arbitrary values of σ. Rather then generalizing Gabcke's analysis, in Chapter 7 we instead outline a method to compute $\zeta(s)$ using numerical quadrature of an integral underlying the Riemann-Siegel formula. This method should have several advantages, including the ability to find $\zeta(s)$ to arbitrary accuracy (for most values of s), and simplicity of error analysis.

2 The Kernel Function

2.1 Choice of the kernel

We choose to use a "Gaussian" kernel, $\phi(u; x, \lambda)$, based on the complementary error function: $\mathrm{erfc}(z) := \frac{2}{\sqrt{\pi}} \int_z^\infty e^{-r^2} dr$. After stating some properties of $\mathrm{erfc}(z)$ we will proceed with the definition of our kernel.

Lemma 2.1 *For $z \in \mathbb{R}$ we have*

$$\mathrm{erfc}(-z) = 2 - \mathrm{erfc}(z) \tag{2.1}$$

$$\mathrm{erfc}(0) = 1 \tag{2.2}$$

$$\frac{d}{dz}\,\mathrm{erfc}(z) = -\frac{2}{\sqrt{\pi}} e^{-z^2}. \tag{2.3}$$

For $z \geq 0$ we have

$$\frac{e^{-z^2}}{z + \sqrt{z^2 + 2}} < \frac{\sqrt{\pi}}{2}\,\mathrm{erfc}(z) \leq \frac{e^{-z^2}}{z + \sqrt{z^2 + \pi/4}}. \tag{2.4}$$

Proof: These results follow from the definition of $\mathrm{erfc}(z)$ and from results in Abramowitz and Stegun [AS92]—see Entries 7.1.1, 7.1.2, 7.1.9, 7.1.13. ■

The following corollary, which we state without proof, follows easily from the bound (2.4).

Corollary 2.2 *For $z \geq 0$ we have*

$$\int_z^\infty e^{-r^2} dr \leq \frac{2}{\sqrt{\pi}} e^{-z^2} \tag{2.5}$$

$$\mathrm{erfc}(z) \leq \frac{4}{\pi} e^{-z^2} \tag{2.6}$$

$$\mathrm{erfc}(z) \leq \frac{e^{-z^2}}{\sqrt{\pi}\, z}. \tag{2.7}$$

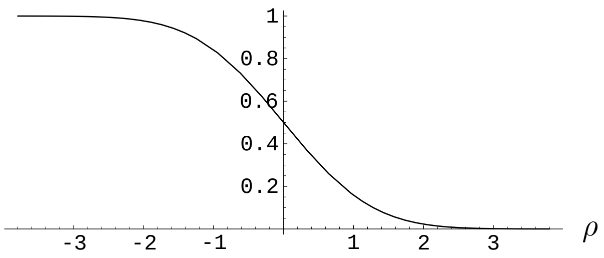
Definition 2.3 Let

$$\Phi(\rho) := \frac{1}{2}\,\mathrm{erfc}(\rho/\sqrt{2}), \tag{2.8}$$

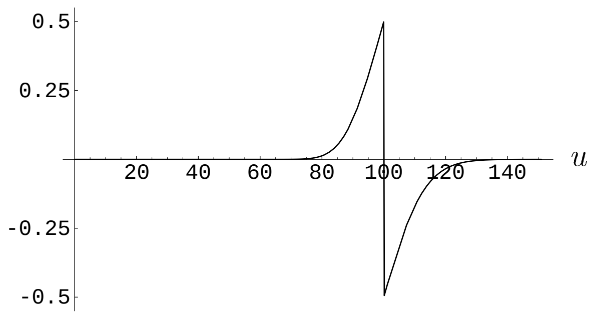
and for $\lambda > 0$, $x > 0$, $u > 0$, let

$$\phi(u; x, \lambda) := \Phi(\ln(u/x)/\lambda) = \frac{1}{2}\,\mathrm{erfc}\left(\frac{\ln(u/x)}{\sqrt{2}\,\lambda}\right). \tag{2.9}$$

For $\lambda = 0$ we let $\phi(u; x, 0) := \lim_{\lambda \to 0} \phi(u; x, \lambda)$. It follows that $\phi(u; x, 0)$ is our "step kernel", $\chi(u; x)$, defined by Equation (1.14). From this point we will usually write $\phi(u; x, 0)$ in preference to $\chi(u; x)$. $\square$



(a) $\Phi(\rho)$



(b) $\phi(u; x, 0) - \phi(u; x, \lambda)$, $x = 100$, $\lambda = 0.1$

Figure 2.1: Functions related to the kernel function $\phi(u; x, \lambda)$

Remark The use of this kernel is not original to this dissertation. For example, Lehman used a closely related Gaussian kernel in his work on the difference between $\pi(x)$ and $\mathrm{li}(x)$ [Leh66]. $\square$

Lemma 2.4 *For $\lambda \geq 0$ we have*

$$\phi(e^{-\tau}x; x, \lambda) = \Phi(-\tau/\lambda) = 1 - \Phi(\tau/\lambda) = 1 - \phi(e^{\tau}x; x, \lambda) \tag{2.10}$$

$$\phi(x; x, \lambda) = 1/2 \tag{2.11}$$

$$\frac{d}{d\rho}\Phi(\rho) = -\frac{1}{\sqrt{2\pi}}e^{-\rho^2/2}. \tag{2.12}$$

For $\lambda > 0$ and $\tau \geq 0$ we have

$$\phi(e^{\tau}x; x, \lambda) = \Phi(\tau/\lambda) \leq \frac{2}{\pi}e^{-\tau^2/(2\lambda^2)}, \tag{2.13}$$

$$\phi(e^{-\tau}x; x, \lambda) = \Phi(-\tau/\lambda) = 1 + O(e^{-\tau^2/(2\lambda^2)}). \tag{2.14}$$

Proof: These results follow easily from Lemma 2.1 and Corollary 2.2. ■

Figure 2.1(a) shows $\Phi(\rho; \lambda)$, while Figure 2.1(b) illustrates the behavior of the difference $\phi(u; x, 0) - \phi(u; x, \lambda)$, which corresponds to the difference $\chi(p^m; x) - \phi(p^m)$ appearing in Equation (1.23).

The *length parameter*, λ, controls the rate at which $\phi(u; x, \lambda)$ "cuts off". The bounds (2.13) and (2.14) imply that $\phi(e^{\tau}x; x, 0) - \phi(e^{\tau}x; x, \lambda) \ll e^{-\tau^2/(2\lambda^2)}$ (for both positive and negative τ). As will be seen in later sections, we are especially interested in situations where both λ and $|\tau|$ are much less than one, so that $e^{\tau}x$ will lie near $x + \tau x$, while $\phi(e^{\tau}x; x, 0) - \phi(e^{\tau}x; x, \lambda)$ will be nearly zero once $|\tau|$ exceeds a few multiples of λ.

2.2 Mellin transform of the kernel

After a preparatory lemma, in Theorem 2.6 on the next page we find the Mellin transform of $\phi(u; x, \lambda)$.

Lemma 2.5 *For fixed $s \in \mathbb{C}$, $\lambda > 0$, we have*

$$\int_{-\infty}^{\infty} e^{-(\tau-\lambda^2 s)^2/(2\lambda^2)}\, d\tau = \sqrt{2\pi}\,\lambda. \tag{2.15}$$

Proof: Letting $z = (\tau - \lambda^2 s)/(\sqrt{2}\,\lambda)$, we have

$$\int_{-\infty}^{\infty} e^{-(\tau-\lambda^2 s)^2/(2\lambda^2)}\, d\tau = \sqrt{2}\,\lambda \int_{-\infty-\lambda s/\sqrt{2}}^{\infty-\lambda s/\sqrt{2}} e^{-z^2}\, dz. \tag{2.16}$$

We then move the path of integration to the real axis. This leaves the value of the integral unchanged since the integrand is an entire function of z and since, for bounded $\mathrm{Im}(z)$, the integral converges absolutely and uniformly as $|\mathrm{Re}(z)| \to \infty$. By the identity $\int_{-\infty}^{+\infty} e^{-z^2}\,dz = \sqrt{\pi}$ (equivalent to Equation (2.2)), we find that (2.16) is $= \sqrt{2}\,\lambda \int_{-\infty}^{+\infty} e^{-z^2}\,dz = \sqrt{2\pi}\,\lambda$. ■

Theorem 2.6 *For* $\lambda \geq 0$, $\mathrm{Re}(s) > 0$, *the Mellin transform of* $\phi(u; x, \lambda)$ *is*

$$\widehat{\phi}(s; x, \lambda) := \int_0^\infty \phi(u; x, \lambda) u^{s-1}\,du = e^{\lambda^2 s^2/2}\,\frac{x^s}{s}. \tag{2.17}$$

Proof: The proof is trivial if $\lambda = 0$. Otherwise, letting $u = e^\tau x$, we have

$$\int_0^\infty \phi(u; x, \lambda) u^{s-1}\,du = x^s \int_{-\infty}^\infty \Phi(\tau/\lambda) e^{s\tau}\,d\tau. \tag{2.18}$$

Since $\mathrm{Re}(s) > 0$, the bound (2.14) gives

$$\lim_{\tau\to-\infty} \Phi(\tau/\lambda) e^{s\tau} \ll \lim_{\tau\to-\infty} e^{-\,\mathrm{Re}(s)\tau} = 0,$$

while (2.13) gives

$$\lim_{\tau\to\infty} \Phi(\tau/\lambda) e^{s\tau} \ll \lim_{\tau\to\infty} e^{s\tau - \tau^2/(2\lambda^2)} = 0.$$

With these limiting values, integrating the right side of (2.18) by parts, and using (2.12), gives

$$\begin{aligned} x^s \int_{-\infty}^\infty \Phi(\tau/\lambda) e^{s\tau}\,d\tau &= -\frac{x^s}{s} \int_{-\infty}^\infty e^{s\tau} \frac{d}{d\tau}\Phi(\tau/\lambda)\,d\tau \\ &= \frac{x^s}{s} \frac{1}{\sqrt{2\pi}\,\lambda} \int_{-\infty}^\infty e^{-\tau^2/(2\lambda^2)+s\tau}\,d\tau. \end{aligned}$$

After completing the square in $e^{-\tau^2/(2\lambda^2)+s\tau}$, this is

$$= \frac{x^s}{s} e^{\lambda^2 s^2/2} \frac{1}{\sqrt{2\pi}\,\lambda} \int_{-\infty}^\infty e^{-(\tau-\lambda^2 s)^2/(2\lambda^2)}\,d\tau,$$

which is $e^{\lambda^2 s^2/2} x^s/s$ by Equation (2.15). ■

In Lemma 2.7, below, we establish two bounds on $|\widehat{\phi}(s; x, \lambda)|$. The first bound will be used in showing that $\phi(u; x, \lambda)$ is a suitable kernel function for the analytic algorithm in the sense of Definition 1.8. The second bound will be used later—in Chapter 3.

Lemma 2.7 *Let $s = \sigma + it$, $\sigma > 1$, $\lambda \geq 0$. Then*

$$\left|\widehat{\phi}(s;x,\lambda)\right| \leq e^{\lambda^2(\sigma^2-t^2)/2}x^{\sigma}, \tag{2.19}$$

$$\left|\widehat{\phi}(s;x,\lambda)\right| \leq e^{\lambda^2(\sigma^2-t^2)/2}x^{\sigma}/\left|t\right|. \tag{2.20}$$

Proof: Recalling that $\widehat{\phi}(s;x,\lambda) = e^{\lambda^2 s^2/2}x^s/s$, we bound individual factors and then take the product of the bounds. We find that $|x^s| = x^{\sigma}$ and that $\left|e^{\lambda^2 s^2/2}\right| = e^{\lambda^2(\sigma^2-t^2)/2}$. Writing $|s| = |\sigma + it|$, since $\sigma > 1$ we have both $1/|s| \leq 1$, establishing (2.19); and $1/|s| \leq 1/|t|$, establishing (2.20). ■

Theorem 2.8 *Given fixed $x > 0$, $\lambda \geq 0$, we have $\phi(u;x,\lambda)$ is a suitable kernel function for the analytic algorithm in the sense of Definition 1.8 on page 18.*

Proof: As mentioned in Section 1.7, the case where $\lambda = 0$ is a classical result. Considering the case where $\lambda > 0$, Definition 1.8 leads us to first establish that $\phi(u;x,\lambda)$ is a kernel function; evenly-stepped; and of type $(0,\infty)$—and thence to Definitions 1.3 and 1.5 of Section 1.6. We see that $\phi(u;x,\lambda)$ is of bounded variation on finite intervals since it is monotone decreasing, and thus a kernel function. It is evenly-stepped since it is continuous. It easily follows from the bounds (2.13) and (2.14) that $\phi(u;x,\lambda)$ is of type $(0,\infty)$.

The one remaining requirement of Definition 1.8 is that for $\sigma > 1$ Equation (1.10) holds for the Dirichlet series $\sum_{n\geq 1} a_n n^{-s} := \ln\zeta(s)$. This follows from Corollary 1.7, where we again use the bounds (2.13) and (2.14) to establish that $\sum_{n\geq 1} a_n\phi(n)$ converges; and the bound (2.19) to establish that $\int_{\sigma-i\infty}^{\sigma+i\infty}\widehat{\phi}(s)\,ds$ converges absolutely. ■

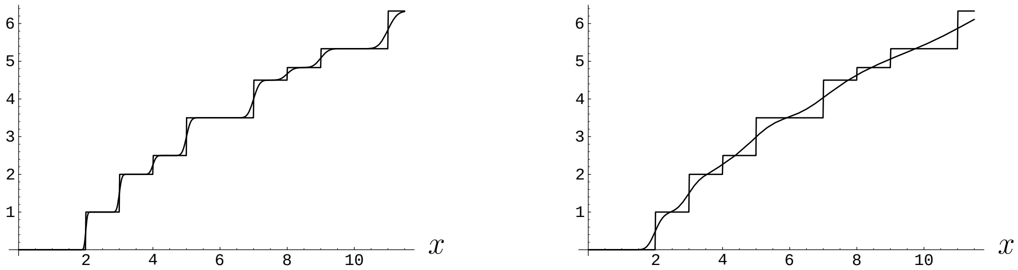
2.3 Restatement of the analytic algorithm

Before restating the analytic algorithm for computing $\pi(x)$ we introduce a "smoothed" version of $\pi^*(x)$:

Definition 2.9 For $\lambda \geq 0$, we let

$$\pi^*(x;\lambda) := \sum_{p^m} \frac{1}{m}\phi(p^m; x, \lambda). \tag{2.21}$$ $\square$

The convergence of (2.21) follows easily from the bound (2.13). Note that $\pi^*(x;0) = \pi^*(x)$ as defined in Section 1.7.



(a) $\lambda = 0.02$

(b) $\lambda = 0.1$

Figure 2.2: $\pi^*(x;\lambda)$ versus $\pi^*(x) = \pi^*(x;0)$, for two values of λ

Remark For $\lambda > 0$, under a change of variables, we can view $\pi^*(x;\lambda)$ as a convolution of $\pi^*(x)$ with an approximation to the Dirac delta function—as follows: Rewriting (2.21) as a Stieltjes integral, and writing $x = e^\alpha$, $u = e^\tau$, and then integrating by parts, we have

$$\begin{aligned}
\pi^*(e^\alpha;\lambda) &= \int_{-\infty}^{\infty} \phi(e^\tau; e^\alpha, \lambda)\, d\pi^*(e^\tau) = -\int_{-\infty}^{\infty} \pi^*(e^\tau)\frac{d}{d\tau}\phi(e^\tau; e^\alpha, \lambda)\, d\tau \\
&= -\int_{-\infty}^{\infty} \pi^*(e^\tau)\frac{d}{d\tau}\Phi((\tau-\alpha)/\lambda)\, d\tau \\
&= \int_{-\infty}^{\infty} \pi^*(e^\tau)\frac{e^{-(\tau-\alpha)^2/(2\lambda^2)}}{\sqrt{2\pi}\lambda}\, d\tau. \qquad (2.22)
\end{aligned}$$

In (2.22) the Dirac delta function, $\delta(\tau)$, is approximated as $\lambda \to 0$ by

$$-\frac{d}{d\tau}\Phi(\tau/\lambda) = \frac{e^{-\tau^2/(2\lambda^2)}}{\sqrt{2\pi}\lambda}.$$

Thus $\pi^*(x;\lambda)$ approximates $\pi^*(x)$ as λ nears zero. Figure 2.2 illustrates $\pi^*(x;\lambda)$ versus the step-function $\pi^*(x)$ for two values of λ. Note how $\pi^*(x;\lambda)$ becomes smoother as λ increases. $\square$

Returning to our restatement of the analytic algorithm, we see that in terms of $\pi^*(x;\lambda)$ Equation (1.22) from page 19 gives, for $x>0$, $\sigma>1$, $\lambda\geq 0$,

$$\pi^*(x;\lambda)=\frac{1}{2\pi i}\int_{\sigma-i\infty}^{\sigma+i\infty}\widehat{\phi}(s;x,\lambda)\ln\zeta(s)\,ds, \tag{2.23}$$

while Equation (1.23) yields the almost trivial statement

$$\pi^*(x)=\pi^*(x;\lambda)+\bigl(\pi^*(x)-\pi^*(x;\lambda)\bigr), \tag{2.24}$$

which becomes less trivial after expanding the first occurrence of $\pi^*(x;\lambda)$ as an integral and the parenthesized term as a sum over prime powers:

$$\begin{aligned}\pi^*(x)=&\frac{1}{2\pi i}\int_{\sigma-i\infty}^{\sigma+i\infty}\widehat{\phi}(s;x,\lambda)\ln\zeta(s)\,ds\\&+\sum_{p^m}\frac{1}{m}\bigl(\phi(p^m;x,0)-\phi(p^m;x,\lambda)\bigr).\end{aligned} \tag{2.25}$$

The analytic algorithm proposed by Lagarias and Odlyzko approximates the sum and the integral in (2.25), giving an approximation to $\pi^*(x)$. In a final stage, their version computes an approximation to $\pi(x)$, via the identity

$$\pi(x)=\pi^*(x)-\sum_{m\geq 2}\frac{1}{m}\pi(x^{1/m}), \tag{2.26}$$

valid for x not a prime power. In Section 3.2 we present a slightly different approach, in which the sums over prime powers of Equations (2.25) and (2.26) are combined. In either case, if all sources of error have been sufficiently bounded, we will get an approximation to $\pi(x)$, say $\pi(x)+\varepsilon\,O(1)$, with ε comfortably less than $1/2$. Rounding this approximation to the nearest integer gives $\pi(x)$ exactly.

2.4 Computing the kernel and its transform

For our kernel to be useful, we must be able to efficiently compute both the kernel, $\phi(u;x,\lambda)$, and its transform, $\widehat{\phi}(s;x,\lambda)$.

We can easily deal with the computation of $\widehat{\phi}(s;x,\lambda)$, which reduces to the computation of elementary transcendental functions. As discussed in Section 1.5, it follows that $\widehat{\phi}(s;x,\lambda)$ can be computed to n-bit accuracy using

$O(\ln(2n)\mathcal{M}(n))$ bit-operations. For our application we can almost certainly reduce the average cost of computing $\widehat{\phi}(s;x,\lambda)$ to $O(\mathcal{M}(n))$ bit-operations. However, we will not explore techniques for this, since each computation of $\widehat{\phi}(s;x,\lambda)$ is paired with a computation of $\zeta(s)$, and we expect the latter computation to have the dominant bit complexity.

The computation of $\phi(u;x,\lambda)$ is not so straightforward, although it reduces to the problem of computing $\operatorname{erfc}(z)$, which is a well-understood function. Furthermore the computation of $\phi(u;x,\lambda)$ is less likely to be dominated by other computations.

As detailed further in Chapter 3, most evaluations of $\phi(u;x,\lambda)$ occur in the context of evaluating $\phi(p;x,\lambda)$ for prime p in an interval $[x_1,x_2]$ with $x_1 < x < x_2$, and with $x_2 - x_1 = x^{1/2+o(1)}$ as $x \to \infty$. If we use the algorithm of Chapter 5, we can enumerate the primes in the interval $[x_1,x_2]$ with an average cost of $O(\ln(x)\,\mathcal{M}(\ln x))$ bit-operations per prime. For this reason, we will give a detailed analysis of a method that can approximate $\phi(u;x,\lambda)$ with a computational cost of the same order of magnitude.

Our approach is based on a suggestion by Richard Crandall, who observed [Cra02] that the following expansion—based on work of Chiarella and Reichel [CR68]—can be used to compute $\operatorname{erfc}(z)$ with error bounded by 2^{-n}, using $O(n)$ arithmetic operations on numbers of $O(n)$ bits:

$$\operatorname{erfc}(z) = \frac{e^{-z^2}hz}{\pi}\sum_{k\in\mathbb{Z}}\frac{e^{-h^2k^2}}{z^2+h^2k^2} + \frac{2}{1-e^{2\pi z/h}} + 3O(e^{-\pi^2/h^2}). \tag{2.27}$$

Here $h > 0$ is a parameter chosen to bound the O-term. (Note that the O-term is dominant for $h \ge \pi/z$.)

The expansion (2.27) has a removable singularity at $z = 0$, which makes the task of approximating it more difficult when $|z|$ is near zero. For this reason, for $|z| \le 1/2$, we prefer to use a classical formula such as the series (1.4) used by Algorithm 1.1 (`ExampleErfc`) on page 10. Since, for $|z| \le 1/2$, Algorithm 1.1 sums $O\big(\ln(2+|z/\varepsilon|)\big)$ terms, Algorithm 1.1 requires

$$\ll \ln(2+|z/\varepsilon|)\,\mathcal{M}(\ln(2+|z/\varepsilon|)) \tag{2.28}$$

bit-operations to find $\operatorname{erfc}(z) + \varepsilon\,O(1)$.

When z is bounded away from zero we can use Equation (2.27). This requires that we choose h and the number of terms to be summed in the series in order to achieve a given error bound. Theorem 2.10, below, gives detailed choices for these parameters:

Theorem 2.10
Given $0 < \varepsilon \leq e^{-1}$, *let* $h = \pi/\sqrt{\ln(6/\varepsilon)}$ *and let* $K = \left\lceil \sqrt{\ln(1/\varepsilon)}/h \right\rceil$. *Then*

$$\operatorname{erfc}(z) = \frac{e^{-z^2} hz}{\pi} \sum_{k=-K}^{K} \frac{e^{-h^2k^2}}{z^2 + h^2k^2} + \frac{2}{1 - e^{2\pi z/h}} + \varepsilon\, O(1). \tag{2.29}$$

Proof: With $h = \pi/\sqrt{\ln(6/\varepsilon)}$ we have $\pi^2/h^2 = -\ln(\varepsilon/6)$ and thus

$$3O(e^{-\pi^2/h^2}) = \frac{\varepsilon}{2} O(1). \tag{2.30}$$

Turning to our choice of K, note that the transition from (2.27) to the truncated sum of (2.29) introduces an error bounded by

$$\frac{2}{\pi} e^{-z^2} h\, |z| \sum_{k>K} \frac{e^{-h^2k^2}}{z^2 + h^2k^2}\,. \tag{2.31}$$

Focusing on the sum over k in (2.31): since $K \in \mathbb{N}$ we can easily justify bounding the sum by an integral to find that

$$\sum_{k>K} \frac{e^{-h^2k^2}}{z^2 + h^2k^2} \leq \int_K^\infty \frac{e^{-h^2k^2}}{z^2 + h^2k^2}\, dk \leq \int_K^\infty \frac{e^{-h^2k^2}}{h^2k^2}\, dk, \tag{2.32}$$

and under the change of variables $r := hk$, the rightmost bound in (2.32) is

$$= \frac{1}{h} \int_{hK}^\infty e^{-r^2}/r^2\, dr. \tag{2.33}$$

Our constraint that $\varepsilon \leq e^{-1}$ ensures that $hK \geq 1$, so the bound (2.33) is

$$\leq \frac{1}{2h} \int_{hK}^\infty 2r\, e^{-r^2}\, dr = \frac{1}{2h} e^{-h^2K^2}.$$

It follows that the expression (2.31) is

$$\leq \frac{1}{\pi} |z|\, e^{-z^2} e^{-h^2K^2} \leq \frac{e^{-h^2K^2}}{\pi\sqrt{2\, e}}\,, \tag{2.34}$$

where the rightmost bound follows from solving for the zeros of $d/dz\ z e^{-z^2}$ to find that $|z|\, e^{-z^2}$ achieves its maximum value at $z = \pm 1/\sqrt{2}$.

Our choice of K ensures that $K \geq \sqrt{\ln(1/\varepsilon)}/h$, so $e^{-h^2K^2} \leq \varepsilon$. With this fact, and noting that $\pi\sqrt{2\,e} > 2$, we can apply the rightmost bound of (2.34) to conclude that (2.31) is bounded by $\varepsilon/2$. Combining this bound with the bound (2.30) finishes the proof of our theorem. ■

Remark In [Cra96, §2.4] Crandall suggests the rule-of-thumb that n *decimal* digits of accuracy in (2.29) may be achieved by setting $h = 1/\sqrt{n}$ and setting $K = \lfloor n\sqrt{\ln 10}\rfloor$. Theorem 2.10 implies that h can be taken roughly twice as large, and K roughly half as large, as in this rule-of-thumb. □

Recall from Section 1.5 that the elementary transcendental functions require $O(\ln(2n)\mathcal{M}(n))$ bit-operations for computation to n-bit accuracy. By Theorem 2.10, we see that Formula (2.29) sums a total of $O(K) = O(\ln(1/\varepsilon))$ terms, and that a straightforward evaluation of each term would require $O(\ln(2n)\mathcal{M}(n))$ bit operations, with $n \ll \ln(2 + |z/\varepsilon|)$. This would imply that we can compute $\operatorname{erfc}(z) + \varepsilon\,O(1)$ using

$$\ll \ln(1/\varepsilon)\ln(2\ln(2 + |z/\varepsilon|))\mathcal{M}(\ln(2 + |z/\varepsilon|))$$

bit-operations. This can be reduced to

$$\ll \ln(1/\varepsilon)\,\mathcal{M}(\ln(2 + |z/\varepsilon|)) \tag{2.35}$$

bit-operations by noting that the factors $e^{-h^2k^2}$ occurring in (2.29) can be computed with one elementary function evaluation of $e^{-h^2} =: R$, followed by repeatedly applying the recursions:

$$\begin{aligned} e^{-(k+1)^2h^2} &= Re^{-2kh^2}e^{-k^2h^2} \\ e^{-2(k+1)h^2} &= R^2e^{-2kh^2}. \end{aligned}$$

Note that the bound (2.35) for the complexity of computing Formula (2.29) dominates the bound (2.28) for the complexity of using Algorithm 1.1 to compute $\operatorname{erfc}(z) + \varepsilon\,O(1)$, since the latter bound assumes that $z \leq 1/2$. Thus, we can (and will) use the bound (2.35) for *all* z.

Turning now from the computation of $\operatorname{erfc}(z)$ to the computation of $\phi(u; x, \lambda) = \operatorname{erfc}(\ln(u/x)/(\sqrt{2}\,\lambda))/2$, we apply a simple analysis like the one given in the **Remarks** on page 11 which follow Algorithm 1.1 (`ExampleErfc`).

This leads us to conclude that

$$\phi(u; x, \lambda) + \varepsilon\, O(1) = \frac{1}{2}\left(\operatorname{erfc}\left(\frac{\ln(u/x) + c_1\lambda\varepsilon\, O(1)}{\sqrt{2}\,\lambda} + c_2\varepsilon\, O(1)\right) + \varepsilon\, O(1)\right).$$

where we can take $c_1 := \sqrt{2\pi}/4$, and $c_2 := \sqrt{\pi}/4$. In other words, the $c_1\lambda\varepsilon\, O(1)$ term indicates the accuracy to which $t := \ln(u/x)$ must be approximated; while the $c_2\varepsilon\, O(1)$ term indicates the accuracy to which $z := t/(\sqrt{2}\,\lambda)$ must be approximated; and the $\varepsilon\, O(1)$ term indicates the accuracy to which $\operatorname{erfc}(z)$ must be approximated.

Clearly the complexity of approximating $\ln(u/x)$ and $\operatorname{erfc}(z)$ dominates the complexity of approximating $t/(\sqrt{2}\,\lambda)$. For our application—as outlined in Theorem 3.13 on page 49 and its proof—we can assume that $\ln(2) \le \ln(u) \ll \ln(x)$ for some fixed O-constant, so $\ln(z) \ll \ln(x)$. We can also assume that $\lambda \gg x^{-1/2}$ and that $\varepsilon \le e^{-1}$. Under these conditions, summing the two dominant complexity bounds and observing that the first term dominates, we find that we need

$$\begin{aligned} &\ll \ln(x/(\lambda\varepsilon))\,\mathcal{M}(\ln(x/(\lambda\varepsilon))) + \ln(1/\varepsilon)\,\mathcal{M}(\ln(2 + \ln(x)/\varepsilon)) \\ &\ll \ln(x/(\lambda\varepsilon))\,\mathcal{M}(\ln(x/(\lambda\varepsilon))) \qquad (2.36) \end{aligned}$$

bit operations to compute $\phi(u; x, \lambda) + \varepsilon\, O(1)$.

2.5 How optimal is our kernel?

$\chi(u; x)$ be the step function defined by Equation (1.14) on page 16. Taking x as fixed, our criterion for optimality of $\phi(u)$, will be that $\phi(u) - \chi(u; x)$ should approach zero as rapidly as possible as u moves away from x, while $\widehat{\phi}(s)$ should approach zero as rapidly as possible as $\operatorname{Im}(s) \to \pm\infty$.

Our argument depends on a theorem due to G. H. Hardy, which places constraints on how rapidly a function and its Fourier transform can both decay to zero. Hardy's theorem can be paraphrased as:

Theorem 2.11 (Theorem 2 of [Har33]) *Let the functions f and g be a Fourier transform pair, i.e., a pair satisfying*

$$f(\rho) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} g(\theta) e^{-i\rho\theta}\, d\theta, \qquad g(\theta) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(\rho) e^{i\rho\theta}\, d\rho.$$

Treating f and g as functions of the variable α, if $f(\alpha)$ and $g(\alpha)$ are both $O(e^{-\alpha^2/2})$ as $|\alpha| \to \infty$, then $f(\alpha) = g(\alpha) = Ae^{-\alpha^2/2}$, where A is a constant; and if moreover one of $f(\alpha)$, $g(\alpha)$ is $o(e^{-\alpha^2/2})$ then $A = 0$.

Thus we see that if one member of a non-zero Fourier pair were chosen to vanish more rapidly than $e^{-\alpha^2/2}$ as $|\alpha| \to \infty$, the other member must vanish less rapidly.

Recognizing that we must balance the rate at which $\phi(u) - \chi(u; x) \to 0$ as u moves away from x against the rate at which $\widehat{\phi}(s) \to 0$ as $\mathrm{Im}(s) \to \pm\infty$, we introduce $\lambda > 0$ as a "balancing parameter". To formulate our goal in terms of Fourier transforms we write $\phi(u) =: \Phi(\ln(u/x)/\lambda)$, where $\Phi(\rho)$ is not necessarily the function $\mathrm{erfc}(\rho/\sqrt{2})/2$ of Definition 2.3.

Our goal that $\phi(u) - \chi(u; x)$ should rapidly approach zero is equivalent to the requirement that $\Phi(\rho)$ closely approximate the step function $\chi(\rho; 0)$ as ρ moves away from zero. Assuming that $\Phi(\rho)$ is differentiable, let $f(\rho) := -\Phi'(\rho)$, so that $\Phi(\rho) = \int_\rho^\infty f(r)\,dr$. To achieve our goal it suffices to require that $f(\rho) \to 0$ rapidly as ρ moves away from zero, with the additional constraint that $\int_{-\infty}^\infty f(\rho)\,d\rho = 1$ so that $\Phi(\rho) \to 1$ as $\rho \to -\infty$.

We now consider the $\widehat{\phi}(s)$ that results from the $\phi(u)$ described above. We have, for $\mathrm{Re}(s) > 0$,

$$\widehat{\phi}(s) = \int_0^\infty \phi(u) u^{s-1}\,du = \lambda x^s \int_{-\infty}^\infty \Phi(\rho) e^{\lambda\rho s}\,d\rho. \tag{2.37}$$

The integrals in (2.37) converge for $\mathrm{Re}(s) > 0$, i.e., $\phi(u)$ is of type $(0, \infty)$, since our assumptions imply that $\Phi(\rho)$ is bounded as $\rho \to -\infty$ and that $\Phi(\rho) \to 0$ rapidly as $\rho \to \infty$. Integration by parts in (2.37) gives

$$\widehat{\phi}(s) = \frac{x^s}{s} \int_{-\infty}^\infty f(\rho) e^{\lambda\rho s}\,d\rho, \tag{2.38}$$

where the integral converges for $s \in \mathbb{C}$ under the assumption that $f(\rho) \to 0$ sufficiently rapidly as $\rho \to \pm\infty$. Letting $s = i\theta/\lambda$, Equation (2.38) can be written as

$$\widehat{\phi}(i\theta/\lambda) = -\sqrt{2\pi} i \frac{\lambda x^{i\theta/\lambda}}{\theta} g(\theta),$$

where

$$g(\theta) := \frac{1}{\sqrt{2\pi}} \int_{-\infty}^\infty f(\rho) e^{i\rho\theta}\,d\rho.$$

Our goal that $\widehat{\phi}(s) \to 0$ rapidly as $\mathrm{Im}(s) \to \pm\infty$ suggests that we require $g(\theta) \to 0$ rapidly as $\mathrm{Re}(\theta) \to \pm\infty$. In particular, setting $\mathrm{Im}(\theta) = 0$ lets us apply Theorem 2.11. Arguing that, as functions of a single variable α, we want both $f(\alpha)$ and $g(\alpha)$ to vanish as rapidly as possible as $\alpha \to \pm\infty$, Theorem 2.11 implies that $f(\alpha) = g(\alpha) = Ae^{-\alpha^2/2}$. Our requirement that $\int_{-\infty}^{\infty} f(\rho)\,d\rho = 1$ gives $A = 1/\sqrt{2\pi}$, from which we find that $\Phi(\rho) = \mathrm{erfc}(\rho/\sqrt{2})/2$ and that $\phi(u) = \phi(u; x, \lambda)$ as defined by Equation (2.9), thus supporting our claim that this kernel is optimal.

3 Bounding Errors and Choosing Parameters

3.1 Overview

In Section 2.3, Equations (2.25) and (2.26), we reduced the computation of $\pi(x)$ to the computation of an integral and of a sum over prime powers. In this chapter we present specific algorithms for accomplishing these tasks. We carefully analyze the approximations we use for the integral and sum over prime powers, analyze the complexity of the resulting algorithms, and discuss the choice of the parameters involved. A concise list of these parameters appears in the **Index and Glossary**, under the entry "Parameters for analytic $\pi(x)$ algorithm".

The choice of parameters is determined by two goals: ensuring that the total error is bounded by ε (say); and minimizing the computation time. In bounding the error, most of our analysis will be concerned with truncation error; while we give a less thorough analysis of roundoff error. One of our tasks will be to subdivide an error bound ε amongst various sources of error. For this reason, our algorithms may perform assignments such as $\varepsilon \leftarrow \varepsilon/2$ when there are two sources of error—and the reader should remain aware that ε may indicate different quantities at different points in our exposition.

Of course, our choice of parameters and our complexity results depend on the complexity of the methods we use for enumerating primes and for computing $\zeta(s)$. Readily available complexity bounds for previously known algorithms for those tasks suffice to show that the algorithms of this chapter can compute $\pi(x)$ using $x^{1/2+\epsilon}$ time and $x^{1/4+\epsilon}$ space. Ignoring the details hidden in the x^ϵ factors, these bounds agree with the bounds which Lagarias and Odlyzko gave for the complexity of their version of the analytic algorithm.

Before presenting Algorithm 3.1 (`APix`), our "top-level" algorithm for computing $\pi(x)$, we begin by introducing a new function, $\Delta(x;\lambda)$:

Definition 3.1 For $x > 0$ and $\lambda > 0$ we let $\Delta(x;\lambda) := \pi(x) - \pi^*(x;\lambda)$. □

(A similar, but different function $\Delta(...)$ is used by Ekkehart Vetter in his Diplomarbeit [Vet91, pg. 33, Eqn. 2.2.2-6].)

By using $\Delta(x;\lambda)$, Algorithm 3.1 (`APix`) avoids a nuisance that was present in our previous formulations of the analytic algorithm as given in Sections 1.7 and 2.3. These formulations where not applicable when x had the form $x = p^m$. Working with $\Delta(x;\lambda)$ also simplifies our sums over prime powers, and combines the computations corresponding to Equations (2.25) and (2.26) from Section 2.3.

We impose a few constraints on the arguments of Algorithm 3.1, for technical reasons which are explained later in this chapter. Note that Algorithm 3.1 subdivides its error bound ε amongst two sources: error in the computation of $\Delta(x;\lambda)$ and error in the computation of $\pi^*(x;\lambda)$.

Algorithm 3.1 (`APix`: Find $\pi(x)$ using analytic algorithm)

Given $x \geq 2$, $\sigma \geq 1.045$, and length parameter $\lambda > 0$, returns $\pi(x)$.

```
APix(x,σ,λ) {
 assert x ≥ 2; assert σ ≥ 1.045;
 ε ← 1/4;
```

`Delta`(x,λ,ε) *returns* $\Delta(x;\lambda) + \varepsilon\, O(1)$*, while* `QuadPiStar`$(x,\sigma,\lambda,\varepsilon)$ *returns* $\pi^*(x;\lambda) + \varepsilon\, O(1)$*.*

```
 pival ← Delta(x,λ,ε/2) + QuadPiStar(x,σ,λ,ε/2);
 assert pival = π(x) + ε O(1);
 return ⌊1/2 + pival⌋;}
```

To summarize the reminder of this chapter: We analyze the computation of $\Delta(x;\lambda)$ as a sum over prime powers in Section 3.2. Algorithm 3.2 (`Delta`) for approximating $\Delta(x;\lambda)$ is presented in that section, on page 47. We analyze the computation of $\pi^*(x;\lambda)$, using numerical quadrature along a path of the form $s = \sigma + it$, $-T \leq t \leq T$, in Sections 3.3–3.5. This culminates in our presentation and analysis of Algorithm 3.3 (`QuadPiStar`) on page 71. The choice of the parameters σ and λ is discussed in Section 3.6.

3.2 Approximating $\Delta(x;\lambda)$ as a sum over primes

To approximate $\Delta(x;\lambda)$, we use the definition of $\pi(x)$ and the definition of $\pi^*(x;\lambda)$ (Definition 2.9 on page 28) to find that

$$\begin{aligned}
\Delta(x;\lambda) &= \sum_{p\le x} 1 - \sum_{p}\sum_{m\ge 1}\frac{1}{m}\phi(p^m;x,\lambda) \\
&= \sum_{p\le x}\big(1-\phi(p;x,\lambda)\big) - \sum_{p>x}\phi(p;x,\lambda) - \sum_{p}\sum_{m\ge 2}\frac{1}{m}\phi(p^m;x,\lambda). \qquad (3.1)
\end{aligned}$$

As illustrated in Figure 3.1 on the next page, given suitable x_1, x_2, with $x_1 < x < x_2$, we will have $\phi(p^m;x,0) - \phi(p^m;x,\lambda) \approx 0$ for p^m outside the interval $[x_1, x_2]$. With this in mind, we carefully rearrange (3.1) and collect terms of the form $\phi(p^m;x,0) - \phi(p^m;x,\lambda)$ with p^m lying outside $[x_1, x_2]$ as error expressions (the last two "tail-sums" in Equation (3.2)), giving

$$\begin{aligned}
\Delta(x;\lambda) = &\sum_{x_1\le p\le x}\big(1-\phi(p;x,\lambda)\big) - \sum_{x<p\le x_2}\phi(p;x,\lambda) \\
&- \sum_{m\ge 2}\frac{1}{m}\left(\sum_{p^m<x_1} 1 + \sum_{x_1\le p^m\le x_2}\phi(p^m;x,\lambda)\right) \\
&+ \sum_{p^m<x_1}\frac{1}{m}\big(\phi(p^m;x,0)-\phi(p^m;x,\lambda)\big) \\
&+ \sum_{p^m>x_2}\frac{1}{m}\big(\phi(p^m;x,0)-\phi(p^m;x,\lambda)\big). \qquad (3.2)
\end{aligned}$$

Dropping the tail-sums over $p^m < x_1$ and $p^m > x_2$ in Equation (3.2) gives the finite sum used by Algorithm 3.2 (`Delta`) to approximate $\Delta(x;\lambda)$. Note that the sum over m in Equation (3.2) can be treated as a finite sum, since for m sufficiently large we have $p^m > x_2$ for all prime p.

The bulk of our analysis in this section is devoted to choosing the truncation points x_1 and x_2; and in presenting and analyzing the complexity of Algorithm 3.2 (`Delta`). Somewhat surprisingly, some of the results of this section are also useful in our analysis of quadrature error in Section 3.3.

In order to choose suitable x_1, x_2, we bound the tail-sum over $p^m < x_1$ by $\mathcal{E}_-(x_1;x,\lambda)$ and the tail-sum over $p^m > x_2$ by $\mathcal{E}_+(x_2;x,\lambda)$, where $\mathcal{E}_\pm(u;x,\lambda)$

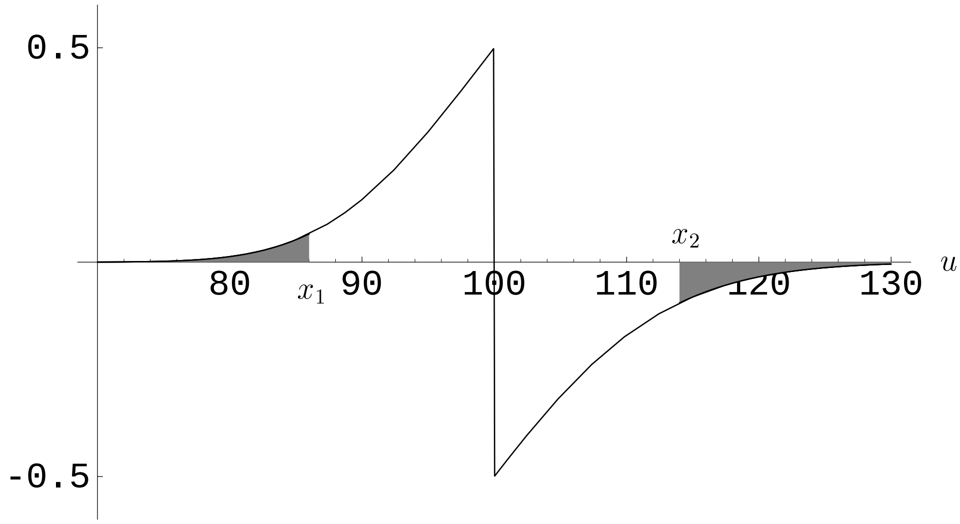


Figure 3.1: $\phi(u;x,0)-\phi(u;x,\lambda)$ showing truncation points x_1, x_2. $x_1 = 86$, $x = 100$, $x_2 = 114$, $\lambda = 0.1$

are defined below. The two tail-sums are of opposite sign, so it follows that

$$\begin{aligned}\Delta(x;\lambda) = &\sum_{x_1\le p\le x}\big(1-\phi(p;x,\lambda)\big) - \sum_{x<p\le x_2}\phi(p;x,\lambda)\\ &-\sum_{m\ge 2}\frac{1}{m}\left(\sum_{p^m<x_1} 1 + \sum_{x_1\le p^m\le x_2}\phi(p^m;x,\lambda)\right) + \varepsilon\, O(1),\end{aligned} \tag{3.3}$$

provided both $\mathcal{E}_-(x_1;x,\lambda)\le\varepsilon$ and $\mathcal{E}_+(x_2;x,\lambda)\le\varepsilon$.

We now define our bounds $\mathcal{E}_\pm(u;x,\lambda)$:

Definition 3.2 Recall that $\Phi(\rho)$ is defined by Equation (2.8) on page 24. For $u>0$, $x>0$, $\lambda>0$, let

$$\mathcal{E}_+(u;x,\lambda) := x\int_{\ln(u/x)}^{\infty}\Phi(\tau/\lambda)e^{\tau}\,d\tau,$$

and let

$$\mathcal{E}_-(u;x,\lambda) := x\int_{\ln(x/u)}^{\infty}\Phi(\tau/\lambda)e^{-\tau}\,d\tau. \qquad \square$$

Although it is not immediately apparent from Definition 3.2, we will see in the proof of Theorem 3.5 that $\mathcal{E}_{\pm}(u; x, \lambda)$ correspond to the shaded areas of Figure 3.1. Note that although $\mathcal{E}_{+}(u; x, \lambda)$ is defined for all $u > 0$, we are primarily interested in its behavior for $u \geq x$. Similarly, our primary interest is in the behavior of $\mathcal{E}_{-}(u; x, \lambda)$ for $u \leq x$.

In order to save space we often state results for both $\mathcal{E}_{+}(u; x, \lambda)$ and for $\mathcal{E}_{-}(u; x, \lambda)$ using the following sign convention:

Definition 3.3 (Sign convention for $\mathcal{E}_{\pm}$) Throughout the rest of this section the upper sign in "$\pm$" and "$\mp$" corresponds to statements about $\mathcal{E}_{+}(u; x, \lambda)$, while the lower sign corresponds to statements about $\mathcal{E}_{-}(u; x, \lambda)$. □

Before showing that $\mathcal{E}_{\pm}(\dots)$ do indeed bound our tail-sums, we begin by establishing some of their simple properties:

Lemma 3.4 *We have $\mathcal{E}_{\pm}(u; x, \lambda) > 0$. For fixed x, λ, and for increasing u, $\mathcal{E}_{+}(u; x, \lambda)$ is strictly decreasing, while $\mathcal{E}_{-}(u; x, \lambda)$ is strictly increasing. For fixed u, x, provided $u^{\pm 1} > x^{\pm 1}$, $\mathcal{E}_{\pm}(u; x, \lambda)$ are strictly increasing in λ.*

Proof: That $\mathcal{E}_{\pm}(u; x, \lambda)$ are positive and that $\mathcal{E}_{+}(u; x, \lambda)$ is decreasing while $\mathcal{E}_{-}(u; x, \lambda)$ is increasing in u follow since both their respective integrands are strictly positive, and since $\ln(u/x)$ is increasing while $\ln(x/u)$ is decreasing.

Differentiating the defining integrals for $\mathcal{E}_{\pm}(u; x, \lambda)$ and taking the derivative under the integral (which is easily justified), we find

$$\frac{\partial}{\partial \lambda}\mathcal{E}_{\pm}(u; x, \lambda) = x \int_{\pm \ln(u/x)}^{\infty} e^{\pm \tau} \frac{\partial}{\partial \lambda}\Phi(\tau/\lambda)\, d\tau.$$

Using Equation (2.12) to find $\partial\, \Phi(\tau/\lambda)/\partial \lambda$ under the integral, we find that

$$\frac{\partial}{\partial \lambda}\mathcal{E}_{\pm}(u; x, \lambda) = \frac{x}{\lambda^2 \sqrt{2\pi}} \int_{\pm \ln(u/x)}^{\infty} e^{\pm \tau} \tau e^{-\tau^2/(2\lambda^2)}\, d\tau. \tag{3.4}$$

Now, $\tau > 0$ over the range of integration since $u^{\pm 1} > x^{\pm 1}$, so the integrand in (3.4) is strictly positive. Since $\lambda > 0$ by assumption in the definition of $\mathcal{E}_{\pm}$, we have $\partial\, \mathcal{E}_{\pm}/\partial \lambda > 0$ and so $\mathcal{E}_{\pm}(u; x, \lambda)$ are strictly increasing in λ. ■

We now establish that $\mathcal{E}_{\pm}(\dots)$ serve to bound our tail-sums:

Theorem 3.5 *Given $x_2 \in \mathbb{Z}_0$, we have*

$$\sum_{p^m > x_2} \frac{1}{m}\phi(p^m; x, \lambda) \le \mathcal{E}_+(x_2; x, \lambda), \tag{3.5}$$

Further, given x_1, $x_2 \in \mathbb{Z}_0$, $x_1 \le x \le x_2$, we have

$$\left| \sum_{p^m > x_2} \frac{1}{m}\big(\phi(p^m; x, 0) - \phi(p^m; x, \lambda)\big) \right| \le \mathcal{E}_+(x_2; x, \lambda), \tag{3.6}$$

and

$$\left| \sum_{p^m < x_1} \frac{1}{m}\big(\phi(p^m; x, 0) - \phi(p^m; x, \lambda)\big) \right| \le \mathcal{E}_-(x_1; x, \lambda). \tag{3.7}$$

Proof: To establish (3.5), we note that

$$\sum_{p^m > x_2} \frac{1}{m}\phi(p^m; x, \lambda) \le \sum_{n > x_2} \phi(n; x, \lambda) \le \int_{x_2}^{\infty} \phi(u; x, \lambda)\, du, \tag{3.8}$$

since $x_2 \in \mathbb{Z}_0$ and $\phi(u; x, \lambda)$ is a decreasing function in u. Setting $u = e^\tau x$ in the rightmost side of (3.8) and integrating with respect to τ gives

$$\int_{x_2}^{\infty} \phi(u; x, \lambda)\, du = x \int_{\ln(x_2/x)}^{\infty} \Phi(\tau/\lambda) e^\tau\, d\tau = \mathcal{E}_+(x_2; x, \lambda),$$

which proves (3.5). The bound (3.6) follows immediately from (3.5), since $\phi(p^m; x, \lambda) \ge 0$ and since $\phi(p^m; x, 0) = 0$ when $p^m > x_2 \ge x$.

Similarly, to establish (3.7), we note that $x_1 \le x$ implies

$$\begin{aligned} \left| \sum_{p^m < x_1} \frac{1}{m}\big(\phi(p^m; x, 0) - \phi(p^m; x, \lambda)\big) \right| &= \sum_{p^m < x_1} \frac{1}{m}\big(1 - \phi(p^m; x, \lambda)\big) \\ &= \sum_{p^m < x_1} \frac{1}{m}\Phi(\ln(x/p^m)/\lambda) \end{aligned} \tag{3.9}$$

by Equation (2.10) on page 25. As before, we bound this by an integral (of an increasing function in u) and then write $u = e^{-\tau}x$ to find that (3.9) is

$$\leq \sum_{1\leq n<x_1} \Phi(\ln(x/n)/\lambda) < \int_0^{x_1} \Phi(\ln(x/u)/\lambda)\, du$$
$$= x\int_{\ln(x/x_1)}^{\infty} \Phi(\tau/\lambda)e^{-\tau}\, d\tau = \mathcal{E}_{-}(x_1; x, \lambda). \qquad \blacksquare$$

Lemma 3.6, below, gives formulae useful both for computing $\mathcal{E}_{\pm}(u; x, \lambda)$ and for finding analytically tractable approximations to $\mathcal{E}_{\pm}(u; x, \lambda)$:

Lemma 3.6

$$\mathcal{E}_{\pm}(u; x, \lambda) = \pm x\, e^{\lambda^2/2}\, \Phi\big(\pm \ln(u/x)/\lambda \mp \lambda\big) \mp u\, \Phi\big(\pm \ln(u/x)/\lambda\big). \tag{3.10}$$

Proof: Letting $\alpha = \pm 1$ as appropriate, we see from Definition 3.2 that both of $\mathcal{E}_{\pm}(u; x, \lambda)$ may be defined as

$$\mathcal{E}_{\pm}(u; x, \lambda) = x\int_{\alpha\ln(u/x)}^{\infty} \Phi(\tau/\lambda)e^{\tau/\alpha}\, d\tau. \tag{3.11}$$

We proceed in much the same spirit as in the proof of Theorem 2.6 on page 26. Integrating (3.11) by parts, with $e^{\tau/\alpha}d\tau = d\,\alpha e^{\tau/\alpha}$, using Equation (2.12) from page 25 and noting that $\alpha^2 = 1$, we find that

$$\mathcal{E}_{\pm}(u; x, \lambda) = \alpha x\left(e^{\tau/\alpha}\Phi(\tau/\lambda)\Big|_{\tau=\alpha\ln(u/x)}^{\tau=\infty} + \frac{1}{\sqrt{2\pi}\,\lambda}\int_{\alpha\ln(u/x)}^{\infty} e^{\tau/\alpha}e^{-\tau^2/(2\lambda^2)}\, d\tau\right)$$
$$= \alpha x\frac{e^{\lambda^2/2}}{\sqrt{2\pi}\,\lambda}\int_{\alpha\ln(u/x)}^{\infty} e^{-((\tau-\lambda^2/\alpha)/(\sqrt{2}\,\lambda))^2}\, d\tau - \alpha u\, \Phi\big(\alpha\ln(u/x)/\lambda\big). \tag{3.12}$$

Letting $r = (\tau - \lambda^2/\alpha)/(\sqrt{2}\,\lambda)$ and $r_0 := (\alpha\ln(u/x) - \lambda^2/\alpha)/(\sqrt{2}\,\lambda)$, we find that (3.12) is

$$= \alpha x\frac{e^{\lambda^2/2}}{\sqrt{\pi}}\int_{r_0}^{\infty} e^{-r^2}\, dr - \alpha u\, \Phi\big(\alpha\ln(u/x)/\lambda\big)$$
$$= \alpha x e^{\lambda^2/2}\Phi\big((\alpha\ln(u/x) - \lambda^2/\alpha)/\lambda\big) - \alpha u\, \Phi\big(\alpha\ln(u/x)/\lambda\big).$$

The desired result follows upon setting $\alpha = \pm 1$. $\blacksquare$

Theorem 3.9, below, gives asymptotic expressions for $\mathcal{E}_{\pm}(u; x, \lambda)$ which will be used in our analysis of Algorithm 3.2 (`Delta`). We first establish some results for a function to be used in the proof of Theorem 3.9.

Definition 3.7 For the remainder of this section we define $F(\rho)$ to be

$$F(\rho) := \sqrt{2\pi}\, e^{\rho^2/2}\, \Phi(\rho). \tag{3.13}$$ □

Lemma 3.8 *For $\rho > 0$ we have*

$$F'(\rho) = -\rho^{-2} + O(\rho^{-4}), \tag{3.14}$$

$$F''(\rho) \ll \rho^{-3}. \tag{3.15}$$

Proof: Differentiation of (3.13), using Equation (2.12) from page 25, gives

$$F'(\rho) = \rho F(\rho) - 1, \tag{3.16}$$

while differentiation of (3.16) gives

$$F''(\rho) = F(\rho) + \rho F'(\rho). \tag{3.17}$$

From Abramowitz and Stegun [AS92, Entry 7.1.23], we have

$$\sqrt{\pi} z e^{z^2} \operatorname{erfc}(z) \sim 1 + \sum_{m \geq 1} (-1)^m \frac{1 \cdot 3 \ldots (2m-1)}{(2z^2)^m},$$

as $z \to \infty$. Truncating this expansion to the first few terms, and rewriting in terms of $F(\rho)$ gives $F(\rho) = \rho^{-1} - \rho^{-3} + O(\rho^{-5})$. Inserting this expansion into (3.16) and (3.17) yields (3.14) and (3.15) respectively. ■

It will become clear in our exposition below that if $\lambda \gg 1$ then our truncation points x_1, x_2 would satisfy $x_2 - x_1 \gg x$ as $x \to \infty$. This would imply a computational cost for approximating $\Delta(x; \lambda)$ that far exceeds our reasonable goal of $x^{1/2+\epsilon}$ operations. For this reason, and to simplify our arguments, we often assume that $\lambda \leq 1/2$, as in Theorem 3.9 below.

Theorem 3.9 *Given $x > 0$, $u > 0$, and using the sign convention of Definition 3.3, let $\tau = \pm \ln(u/x)$. For $0 < \lambda \leq 1/2$ and $\tau \geq \lambda$ we have*

$$\mathcal{E}_{\pm}(u; x, \lambda) = \frac{\lambda u}{\sqrt{2\pi}} e^{-\tau^2/(2\lambda^2)} \left(\lambda^2/\tau^2 + O(\lambda^3/\tau^3)\right). \tag{3.18}$$

Proof: Letting $\rho := \tau/\lambda$ and rewriting Equation (3.10) in terms of $F(\rho)$, we find that

$$\begin{aligned}
\mathcal{E}_{\pm}(u;x,\lambda) &= \pm x e^{\lambda^2/2}\Phi(\rho \mp \lambda) \mp u\Phi(\rho) \\
&= \pm\frac{x}{\sqrt{2\pi}}\left(e^{\lambda^2/2}e^{-(\rho\mp\lambda)^2/2}F(\rho\mp\lambda) - e^{(\pm\tau-\rho^2/2)}F(\rho)\right) \\
&= \pm\frac{u}{\sqrt{2\pi}}e^{-\rho^2/2}\big(F(\rho\mp\lambda) - F(\rho)\big). \qquad (3.19)
\end{aligned}$$

The estimation of (3.19) reduces to an application of Taylor's Theorem and to the use of the expansions (3.14) and (3.15). These give

$$\begin{aligned}
F(\rho\mp\lambda) - F(\rho) &= \mp F'(\rho)\lambda + \frac{F''(\tilde{\rho})}{2}\lambda^2 \\
&= \pm\lambda\left(\rho^2 + O(\rho^{-4}) + O(\lambda\tilde{\rho}^{-3})\right). \qquad (3.20)
\end{aligned}$$

where $\tilde{\rho}$ lies in the interval between ρ and $\rho \mp \lambda$.

Our assumptions that $0 < \lambda \leq 1/2$ ensures that $O(\lambda\tilde{\rho}^{-3}) = O(\tilde{\rho}^{-3})$. Our additional assumption that $\tau \geq \lambda$ gives $\rho \geq 1$, and so $\rho^{-4} \ll \rho^{-3}$ and also $\rho - \lambda \geq \rho/2$. Since $\tilde{\rho} \geq \rho - \lambda$ we conclude $\tilde{\rho}^{-1} \ll \rho^{-1}$. Thus, Equation (3.20) gives

$$F(\rho\mp\lambda) - F(\rho) = \pm\lambda\left(\rho^{-2} + O(\rho^{-3})\right).$$

Substituting this last result into Equation (3.19), with $\rho = \tau/\lambda$, completes the proof. ■

Theorem 3.10, below, gives an analytically tractable estimate for the values of u, expressed in terms of $\tau = \pm\ln(u/x)$, which solve $\mathcal{E}_{\pm}(u;x,\lambda) = \varepsilon$, subject to some restrictions on the parameters.

Theorem 3.10 *There is an absolute constant $\rho_0 \geq 2$ such that given $x > 0$, $0 < \lambda \leq 1/2$, and $0 < \varepsilon \leq \exp(-\rho_0^2/2)\lambda x$, there is a unique τ solving*

$$\mathcal{E}_{+}(e^{\tau}x;x,\lambda) = \varepsilon, \qquad (3.21)$$

and a unique τ solving

$$\mathcal{E}_{-}(e^{-\tau}x;x,\lambda) = \varepsilon. \qquad (3.22)$$

Furthermore, for both choices of sign, τ lies in the interval

$$\lambda(\sqrt{2\ln(\lambda x/\varepsilon)}-1)<\tau<\lambda(\sqrt{2\ln(\lambda x/\varepsilon)}+1). \tag{3.23}$$

I thank Adolf Hildebrand for suggesting the approach used in the following proof.

Proof: First note that Lemma 3.4 gives $\mathcal{E}_\pm(e^{\pm\tau}x;x,\lambda)$ are strictly decreasing as τ increases, so the solutions to (3.21) and (3.22) are unique, if they exist.

By Theorem 3.9, provided $\tau \geq \lambda$ we have

$$\mathcal{E}_\pm(e^{\pm\tau}x;x,\lambda)/\varepsilon = e^{\pm\tau}\frac{\lambda x/\varepsilon}{\sqrt{2\pi}}e^{-\tau^2/(2\lambda^2)}\left(\lambda^2/\tau^2+O(\lambda^3/\tau^3)\right). \tag{3.24}$$

Let $\rho := \sqrt{2\ln(\lambda x/\varepsilon)}$, C be a fixed real number, and $\tau=\lambda(\rho+C)$. We claim that when ρ is large enough then (3.21) and (3.22) have solutions and (3.23) holds.

Note that for $\rho \geq 1-C$ we have $\tau=(\rho+C)\geq\lambda$, and so (3.24) holds. Using $\ln(\lambda x/\varepsilon)=\rho^2/2$, we then rewrite the logarithm of (3.24) in terms of ρ to find that as $\rho\to\infty$

$$\begin{aligned}
&\ln\mathcal{E}_\pm(e^{\pm\tau}x;x,\lambda)-\ln\varepsilon\\
&=\pm\lambda(\rho+C)-\frac{\ln(2\pi)}{2}+\frac{\rho^2}{2}-\frac{(\rho+C)^2}{2}\\
&\qquad-2\ln(\rho+C)-\ln\Big(1+O\big((\rho+C)^{-1}\big)\Big)\\
&=(-C\pm\lambda)\rho\pm\lambda C-C^2/2-\frac{\ln(2\pi)}{2}-2\ln(\rho)+O_C(\rho^{-1}).
\end{aligned} \tag{3.25}$$

Setting $C=-1$, our requirement that $\rho\geq 1-C$ gives $\rho\geq 2$. Furthermore, using our assumption that $\lambda\leq 1/2$, we see that when ρ is large enough in (3.25) the $(-C\pm\lambda)\rho$ term dominates and is positive, which gives $\ln\mathcal{E}_\pm(e^{\pm\lambda(\rho-1)}x;x,\lambda)-\ln\varepsilon>0$. Similarly, if we set $C=1$, then we must have $\rho\geq 0$ and when ρ is large enough we have $\ln\mathcal{E}_\pm(e^{\pm\lambda(\rho+1)}x;x,\lambda)-\ln\varepsilon<0$.

We conclude that there is a constant $\rho_0\geq 2$ such that $\rho\geq\rho_0$ gives

$$\mathcal{E}_\pm(e^{\pm\lambda(\rho-1)}x;x,\lambda)>\varepsilon>\mathcal{E}_\pm(e^{\pm\lambda(\rho+1)}x;x,\lambda).$$

Our condition that $\varepsilon\leq\exp(-\rho_0^2/2)\lambda x$ ensures that $\rho\geq\rho_0$. Since $\mathcal{E}_\pm(e^{\pm\tau}x;x,\lambda)$ are continuous and strictly decreasing in τ; both the existence of the solutions to (3.21) and (3.22), and the bounds (3.23) on their values follow. ■

Theorem 3.11, below, is our main tool in the analysis of Algorithm 3.2 (`Delta`). It gives bounds on the *size* of the interval $[x_1, x_2]$, which we define to be the number of integers within the interval.

Theorem 3.11 *Given $x > 0$, $0 < \lambda \leq 1/2$, and $\varepsilon > 0$, let x_1 be the largest integer $\leq x$ satisfying $\mathcal{E}_-(x_1; x, \lambda) \leq \varepsilon$ and let x_2 be the least integer $\geq x$ satisfying $\mathcal{E}_+(x_2; x, \lambda) \leq \varepsilon$.*

Let ρ_0 denote the absolute constant specified in Theorem 3.10. Then if $\varepsilon \leq \exp(-\rho_0^2/2)\lambda x$ we have

$$1 + x_2 - x_1 \asymp 1 + \big(e^{\lambda\rho} - e^{-\lambda\rho}\big)x, \tag{3.26}$$

where $\rho := \sqrt{2\ln(\lambda x/\varepsilon)}$.

On the other hand, if we do not require $\varepsilon \leq \exp(-\rho_0^2/2)\lambda x$ we still have

$$1 + x_2 - x_1 \ll 1 + \big(e^{\lambda\rho_0} - e^{-\lambda\rho_0}\big)x. \tag{3.27}$$

Proof: Provided $\varepsilon \leq \exp(-\rho_0^2/2)\lambda x$, we can apply Theorem 3.10 and use the upper bound on τ given in (3.23) to conclude that $x_1 \geq -1 + e^{-\lambda(\rho+1)}x$ while $x_2 \leq 1 + e^{\lambda(\rho+1)}x$. (The ∓ 1 arises because x_1 and x_2 are restricted to be integers.) Since $\lambda \leq 1/2$, we can absorb $e^{\pm\lambda}$ into our O-constant, giving

$$1 + x_2 - x_1 \ll 1 + \big(e^{\lambda\rho} - e^{-\lambda\rho}\big)x. \tag{3.28}$$

Similarly, the lower bound on τ in (3.23) gives $x_1 \leq e^{-\lambda(\rho-1)}x$ while $x_2 \geq e^{\lambda(\rho-1)}x$. Again, since $\lambda \leq 1/2$, it follows that

$$1 + x_2 - x_1 \gg 1 + \big(e^{\lambda\rho} - e^{-\lambda\rho}\big)x,$$

and (3.26) follows.

In the case where we drop the condition that $\varepsilon \leq \exp(-\rho_0^2/2)\lambda x$ we can still apply the reasoning which gave the bound (3.28), but now with $\exp(-\rho_0^2/2)\lambda x$ replacing the role of ε in our application of Theorem 3.10. This leads us to conclude that there are $u_1 \leq x$ and $u_2 \geq x$ satisfying $\mathcal{E}_-(u_1; x, \lambda) \leq \exp(-\rho_0^2/2)\lambda x < \varepsilon$ and $\mathcal{E}_+(u_2; x, \lambda) \leq \exp(-\rho_0^2/2)\lambda x < \varepsilon$.

Furthermore, by analogy to (3.28), we have

$$1 + u_2 - u_1 \ll 1 + \big(e^{\lambda\rho_0} - e^{-\lambda\rho_0}\big)x.$$

Since $\mathcal{E}_{\pm}(e^{\pm\tau}x; x, \lambda)$ are continuous and strictly decreasing in τ we must have $x_1 > u_1$ and $x_2 < u_2$, and the bound (3.27) follows. ■

We are now ready to present Algorithm 3.2 (`Delta`) for approximating $\Delta(x;\lambda)$. This algorithm implements Equation (3.3) given on page 39.

To find solutions to equations such as $\mathcal{E}_{-}(x_1; x, \lambda) = \varepsilon$ we introduce the construct

solve_for `var` **in** `eqn`;

to denote the act of finding a solution for the variable `var` in the equation `eqn`. Recognizing that our computations are subject to roundoff error, we use expressions such as

solve_for x **in** $x^2 = 2 + 10^{-12}O(1)$;

to denote the act of setting x to some value for which $|x^2 - 2| \leq 10^{-12}$. This construct is used in lines 7 and 10 of Algorithm 3.2, where we rather arbitrarily specify an error tolerance of $\varepsilon/4$ for our solutions. The equations on those lines may be solved using $O(x^{\epsilon})$ operations on numbers of $O(\ln(2 + |x|))$ bits by means of root-bracketing and bisection methods such as the routines `zbrac` and `rtbis` of [PTVF92, §9.1]. In practice, one would probably instead use Newton's method—although the complexity analysis seems more difficult—since as functions of u both $\mathcal{E}_{+}(u; x, \lambda)$ and $\mathcal{E}_{-}(u; x, \lambda)$ are monotone, with closed forms for their derivatives.

Algorithm 3.2 (`Delta`: Approximate $\Delta(x;\lambda) := \pi(x) - \pi^*(x;\lambda)$)

Given $x > 0$, $\lambda > 0$, $\varepsilon > 0$, return $\Delta(x;\lambda) + \varepsilon\, O(1)$.

1 **Delta**$(x, \lambda, \varepsilon)$ {
2 **assert** $x > 0 \wedge \lambda > 0 \wedge \varepsilon > 0$;
3 *// Subdivide allowable error equally amongst truncation error and roundoff error.*
4 $\varepsilon \leftarrow \varepsilon/2$;
5 $x_1 \leftarrow \lfloor x \rfloor$;
6 **if** $(\mathcal{E}_{-}(x_1; x, \lambda) > \varepsilon)$ {
7 **solve_for** u **in** $\mathcal{E}_{-}(u; x, \lambda) = \frac{3}{4}\varepsilon + \frac{1}{4}\varepsilon\, O(1)$; $x_1 \leftarrow \lfloor u \rfloor$; }
8 $x_2 \leftarrow \lceil x \rceil$;
9 **if** $(\mathcal{E}_{+}(x_2; x, \lambda) > \varepsilon)$ {

10 **solve_for** u **in** $\mathcal{E}_+(u; x, \lambda) = \frac{3}{4}\varepsilon + \frac{1}{4}\varepsilon\, O(1)\,;\qquad x_2 \leftarrow \lceil u \rceil\,;\}$

Set N to the number of integers in the two ranges that p^m may lie in, so that N bounds the total number of terms in both the sums over p in line 13 and the sums over p^m of lines 16–19.

11 $N \leftarrow (1 + x_2 - x_1) + \lceil \sqrt{x_2}\, \rceil\,;$

12 $\varepsilon_1 \leftarrow \varepsilon\, 2^{-\lceil \log_2(N) \rceil}\,;$ // ε_1 *is the maximum allowed error per term.*

13 $\Delta \leftarrow \sum_{x_1 \le p \le x} \big(1 - \phi(p; x, \lambda) + \varepsilon_1 O(1)\big) - \sum_{x < p \le x_2} \big(\phi(p; x, \lambda) + \varepsilon_1 O(1)\big)\,;$

14 **for** $(p \in [2, \sqrt{x_2}\,] \cap \mathcal{P})$

15 **for** $(m \leftarrow 2\,; p^m \le x_2\,; m{+}{+})$

16 **if** $(p^m < x_1)$

17 $\Delta \leftarrow \Delta - \big(1/m + \varepsilon_1 O(1)\big)\,;$

18 **else**

19 $\Delta \leftarrow \Delta - \phi(p^m; x, \lambda)/m + \varepsilon_1 O(1)\,;$

20 **return** $\Delta\,;\}$

Lemma 3.12 *Let x, λ, and ε satisfy the conditions of Algorithm 3.2. Let x_1 and x_2 denote the corresponding quantities calculated in the algorithm and let $L := 1 + x_2 - x_1$. Then Algorithm 3.2 requires $\ll L/\ln(L+1) + \sqrt{x_2}/\ln(x_2)$ computations of $\phi(u; x, \lambda)$.*

Proof: Our proof uses a special case of the Brun-Titchmarsh Theorem in the form proven by Montgomery and Vaughan in [MV73]. To paraphrase the instance of their result which we use: given $x \ge 0$ and $y > 1$, then $\pi(x + y) - \pi(x) \le 2y/\log(y)$. It follows that uniformly for $x \ge 0$ and $y \ge 0$ we have

$$\pi(x + y) - \pi(x) \ll (y + 1)/\log(y + 2). \tag{3.29}$$

We first bound the number of computations of $\phi(p; x, \lambda)$ in the sums of line 13. This is simply the number of primes in the interval $[x_1, x_2]$, which is $\lim_{\delta \to 0+} \pi(x_2 + \delta) - \pi(x_1 - \delta)$. By (3.29), this is $\ll L/\ln(L + 1)$.

In order to bound the number of computations of $\phi(p^m; x, \lambda)$ in the execution of lines 16–19, we use the nearly trivial bound provided by bounding the number of p^m with $m \geq 2$ within the interval $[2, \sqrt{x_2}\,]$. This is

$$\sum_{p \leq \sqrt{x_2}} \sum_{m=2}^{\lfloor \ln(x_2)/\ln(p) \rfloor} 1 \ll \sum_{p \leq x_2{}^{1/4}} \ln(x_2) + \sum_{x_2{}^{1/4} < p \leq \sqrt{x_2}} 1 \ll \sqrt{x_2}/\ln(x_2),$$

where the rightmost bound follows from the Brun-Titchmarsh Theorem (or, more simply, from the prime number theorem).

Summing these bounds, the result follows. ■

In order to continue our analysis of Algorithm 3.2, we must characterize the cost of enumerating primes in an interval $[x_1, x_2]$. To simplify our analysis we will assume that a single fixed method is used, and that the method requires $O(\mathcal{B}_{\mathcal{P}}(x_2))$ bits of memory and

$$\ll \alpha(x_2)(x_2 - x_1) + \beta(x_2) \tag{3.30}$$

arithmetic operations on numbers of $O(\ln(2 + x_2))$ bits. Using these conventions, Table 3.1 summarizes the costs of several methods for enumerating primes.

Method	$\mathcal{B}_{\mathcal{P}}(x)$	$\alpha(x)$	$\beta(x)$
Eratosthenes' sieve	$x^{1/2}$	$\ln\ln(x)$	$x^{1/2}\ln\ln(x)$
Pritchard's "wheel sieve"	$x^{1/2+\epsilon}$	$1/\ln\ln(x)$	$x^{1/2+\epsilon}$
Dissected sieve	$x^{1/3}$	1	$x^{1/3}$
Hybrid sieve	$x^{1/4}$	$\ln\ln(x)$	$x^{1/2}\ln\ln(x)$
APRCL, ECPP	x^{ϵ}	x^{ϵ}	1

Table 3.1: Costs of some methods for enumerating primes. See Chapter 4 for a further discussion of Pritchard's sieve and of the APRCL and ECPP algorithms; Chapter 5 for details of the dissected sieve; and Chapter 6 for details of the hybrid sieve. Note that it is an unproven conjecture that $\mathcal{B}_{\mathcal{P}}(x) = x^{1/4}$ for the hybrid sieve.

To simplify the statement and proof of Theorem 3.13, we assume that $\lambda \gg x^{-1/2}$. This restriction is reasonable, since we will show that the number of quadrature points evaluated by Algorithm 3.3 (`QuadPiStar`) is $\gg 1/\lambda$, and thus the cost of quadrature would dominate the cost of executing Algorithm 3.13 if we were to have $\lambda \ll x^{-1/2}$

Theorem 3.13 *Assume that $x \geq 2$, $x^{-1/2} \ll \lambda \leq 1/2$, and $0 < \varepsilon \leq 1/2$. Assume also that $\alpha(Cx) = O_C(\alpha(x))$; and that $\beta(x)$ is nondecreasing in x*

and $\beta(x) \ll \sqrt{x\ln(x)}$*; where, as explained above, the functions* $\alpha(x)$ *and* $\beta(x)$ *characterize the cost of enumerating primes.*

Then Algorithm 3.2 requires

$$\ll (1+\alpha(x))\sqrt{\ln(\lambda x/\varepsilon)}\,\lambda x\,\mathcal{M}(\ln(x/\varepsilon)) \tag{3.31}$$

bit operations and $\ll x^{\epsilon} + \mathcal{B}_{\mathcal{P}}(x)$ *bits of memory. The O-constant implicit in the bound* (3.31) *depends both on* ε *and on the O-constant implicit in our assumption that* $\lambda \gg x^{-1/2}$*.*

Proof: The bound on the memory requirement of Algorithm 3.2 follows immediately upon noting that all computations other than the enumeration of primes can be accomplished using x^{ϵ} bits of memory.

To establish the bound (3.31), we begin by observing that our assumption that $x^{-1/2} \ll \lambda \le 1/2$ implies that for x sufficiently large we have

$$\varepsilon \le \exp(-\rho_0^2/2)\lambda x, \tag{3.32}$$

where ρ_0 is the absolute constant of Theorem 3.10. Throughout the rest of our proof we will assume that x is so large that (3.32) holds, and then observe that the theorem holds for all $x \ge 2$ after suitably readjusting the O-constant in the bound (3.31).

Under this assumption, it follows from Theorems 3.10 and 3.11 that $x_1 \asymp x \asymp x_2$ and that

$$L \asymp N \asymp \sqrt{\ln(\lambda x/\varepsilon)}\,\lambda x, \tag{3.33}$$

where $L := 1 + x_2 - x_1$ as in Theorem 3.11, and where N is the variable defined in line 11 of Algorithm 3.2. Note that these results hold even though we halved ε in line 4 of Algorithm 3.2, and although we allow some slack in solving the equations that determined x_1 and x_2 in lines 7 and 10. Note also that our assumption that $x^{-1/2} \ll \lambda \le 1/2$ implies that $\lambda x \gg x^{1/2}$.

From (3.33) it follows that ε_1, defined in line 12 of Algorithm 3.2, satisfies

$$\varepsilon_1 \asymp \frac{\varepsilon}{\sqrt{\ln(\lambda x/\varepsilon)}\,\lambda x}\,. \tag{3.34}$$

By the bound (2.36), given on page 33, we know that each computation of $\phi(u; x, \lambda) + \varepsilon_1 O(1)$ in Algorithm 3.2 takes

$$\ll \ln(x/(\lambda\varepsilon_1))\,\mathcal{M}(\ln(x/(\lambda\varepsilon_1))) \ll \ln(x/\varepsilon)\,\mathcal{M}(\ln(x/\varepsilon)) \tag{3.35}$$

bit operations, where the rightmost bound follows from (3.34) and since $\lambda x \gg x^{1/2}$.

From (3.33) we have $\ln(L+1) \asymp \ln(x)$, and clearly $\ln(x_2) \asymp \ln(x)$. Using these two facts, we apply Lemma 3.12 to find that the number of computations of $\phi(u; x, \lambda) + \varepsilon_1 O(1)$ is

$$\ll \frac{1}{\ln(x)}\left(\sqrt{\ln(\lambda x/\varepsilon)}\,\lambda x + \sqrt{x}\right) \ll \frac{1}{\ln(x)}\sqrt{\ln(x/\varepsilon)}\,\lambda x,$$

where, again, the rightmost bound follows since $\lambda x \gg x^{1/2}$.

Combining this last bound with the bound (3.35), we find that the totality of all computations of $\phi(u; x, \lambda) + \varepsilon_1 O(1)$ takes

$$\ll \sqrt{\ln(\lambda x/\varepsilon)}\,\lambda x\,\mathcal{M}(\ln(x/\varepsilon)) \tag{3.36}$$

bit operations, where we have used the fact that $\ln(x/\varepsilon)/\ln(x) = O_\varepsilon(1)$.

By our assumptions about $\alpha(x)$ and $\beta(x)$, the enumeration of the primes in the interval $[x_1, x_2]$ takes

$$\begin{aligned} &\ll \left(\alpha(x)\sqrt{\ln(\lambda x/\varepsilon)}\,\lambda x + \beta(x)\right)\,\mathcal{M}(\ln(x)) \\ &\ll \alpha(x)\sqrt{\ln(\lambda x/\varepsilon)}\,\lambda x\,\mathcal{M}(\ln(x)) \end{aligned} \tag{3.37}$$

bit operations, since, by our assumption that $\beta(x) \ll \sqrt{x\ln(x)}$, the $\beta(x)$ term is dominated by the other term.

Since we assume that $\beta(x)$ is nondecreasing in x, we can easily show that the bound (3.37) dominates the cost of enumerating the primes in the interval $[2, \sqrt{x_2}\,]$. Finally, we note that the costs analyzed above dominate all other costs in Algorithm 3.2, such as the cost of accumulating the partial sum in lines 13, 17, and 19; and the cost of computing p^m after p is given. Recall from our discussion in Section 1.5 that $\mathcal{M}(n)$ is nondecreasing in n, so $\mathcal{M}(x/\varepsilon) \geq \mathcal{M}(x)$. With this in mind, summing the bounds (3.36) and (3.37) the bound claimed in (3.31) follows. ∎

This completes our analysis of Algorithm 3.2 (`Delta`). We conclude this section by finding some upper bounds on $\pi^*(x;\lambda)$ which will be useful in our analysis of quadrature error in Section 3.3. After a preparatory lemma, we give these bounds in Theorem 3.15, below.

Lemma 3.14 *Given $\lambda > 0$, $u \geq e^{\lambda^2}x > 0$, and writing $\tau = \ln(u/x)$, we have*

$$\mathcal{E}_+(u;x,\lambda) \leq \frac{4\sqrt{2}}{\pi\sqrt{\pi}}\lambda u\, e^{-\tau^2/(2\lambda^2)}. \tag{3.38}$$

Proof: We begin by treating τ as a variable of integration, and only later set $\tau = \ln(u/x)$. With this understanding, we begin with the definition of $\mathcal{E}_+(u;x,\lambda)$ and then note that $\ln(u/x) > 0$, so the bound (2.13) from page 25 applies. This gives

$$\mathcal{E}_+(u;x,\lambda) = x\int_{\ln(u/x)}^{\infty} \Phi(\tau/\lambda)e^{\tau}\,d\tau \leq \frac{2}{\pi}x\int_{\ln(u/x)}^{\infty} e^{\tau-\tau^2/(2\lambda^2)}\,d\tau. \tag{3.39}$$

We then complete the square in $\tau - \tau^2/(2\lambda^2)$, set $r = (\tau-\lambda^2)/(\sqrt{2}\,\lambda)$ and $z = (\ln(u/x)-\lambda^2)/(\sqrt{2}\lambda)$, to find that

$$\begin{aligned}\mathcal{E}_+(u;x,\lambda) &\leq \frac{2}{\pi}e^{\lambda^2/2}x\int_{\ln(u/x)}^{\infty} e^{-(\tau-\lambda^2)^2/(2\lambda^2)}\,d\tau\\ &= \frac{2}{\pi}e^{\lambda^2/2}x\sqrt{2}\,\lambda\int_{z}^{\infty} e^{-r^2}\,dr \leq \frac{4\sqrt{2}}{\pi\sqrt{\pi}}e^{\lambda^2/2}\lambda x e^{-z^2}.\end{aligned}$$

The last bound follows from the bound (2.5) on page 23, where we have used $\ln(u/x) \geq \lambda^2$ to ensure $z \geq 0$. The result follows upon rewriting z in terms of $\tau := \ln(u/x)$. ■

Theorem 3.15 *Given $x > 0$, $\lambda > 0$, we have*

$$\pi^*(x;\lambda) \leq e^{\lambda^2/2}x. \tag{3.40}$$

Furthermore, if $x \leq 2e^{-\lambda^2}$, letting $\tau := \ln(2/x)$, we have

$$\pi^*(x;\lambda) \leq 1.7e^{-\tau^2/(2\lambda^2)}. \tag{3.41}$$

Proof: Letting $x_2 = 1$ in the bound (3.5), we have

$$\pi^*(x;\lambda) = \sum_{p^m>1}\frac{1}{m}\phi(p^m;x,\lambda) \leq \mathcal{E}_+(1;x,\lambda) \leq \mathcal{E}_+(0;x,\lambda).$$

Applying Lemma 3.6 and then the bound (2.14) from page 25, we find

$$\mathcal{E}_+(0; x, \lambda) = \lim_{u \to 0} x\, e^{\lambda^2/2}\, \Phi\big(\ln(u/x)/\lambda - \lambda\big) - u\, \Phi\big(\ln(u/x)/\lambda\big) = e^{\lambda^2/2} x,$$

and (3.40) follows.

To establish (3.41), we pull out the first term of the sum defining $\pi^*(x; \lambda)$ and then use the bound (3.5) from page 41, to find

$$\begin{aligned}\pi^*(x; \lambda) &= \phi(2; x, \lambda) + \sum_{p^m > 2} \frac{1}{m} \phi(p^m; x, \lambda) \\ &\leq \phi(2; x, \lambda) + \mathcal{E}_+(2; x, \lambda).\end{aligned}$$

Letting $\tau := \ln(2/x)$, our conditions ensure $\tau > 0$, so we may apply the bound (2.13) from page 25 to find that

$$\phi(2; x, \lambda) = \Phi(\tau/\lambda) \leq \frac{2}{\pi} e^{-\tau^2/(2\lambda^2)}.$$

Again, our condition that $x \leq 2e^{-\lambda^2}$ lets us apply Lemma 3.14, giving

$$\mathcal{E}_+(2; x, \lambda) \leq \frac{4\sqrt{2}}{\pi\sqrt{\pi}}\, e^{-\tau^2/(2\lambda^2)}.$$

These bounds give

$$\pi^*(x; \lambda) \leq \left(\frac{2}{\pi} + \frac{4\sqrt{2}}{\pi\sqrt{\pi}} \right) e^{-\tau^2/(2\lambda^2)} \leq 1.7 e^{-\tau^2/(2\lambda^2)},$$

since a simple calculation establishes that $2/\pi + 4\sqrt{2}/(\pi\sqrt{\pi}) \leq 1.7$. ■

3.3 Approximating the integral by an infinite sum

Throughout the reminder of this chapter, we will let $\Psi(s)$ denote our integrand for the analytic algorithm:

$$\Psi(s) := \Psi(s; x, \lambda) := \widehat{\phi}(s; x, \lambda) \ln \zeta(s) = e^{\lambda^2 s^2/2}\, \frac{x^s}{s} \ln \zeta(s). \tag{3.42}$$

In this section, beginning with Theorem 3.16 below, we will relate the integral representation for $\pi^*(x;\lambda)$ to its approximating Riemann sum:

$$\pi^*(x;\lambda) = \frac{1}{2\pi i}\int_{\sigma-i\infty}^{\sigma+i\infty} \Psi(s)\,ds \approx \frac{h}{2\pi}\sum_{k\in\mathbb{Z}} \Psi(\sigma+ikh).$$

We will continue our quadrature analysis in Section 3.4 starting on page 64, where we analyze the error introduced by truncating this infinite sum to a finite sum. In Section 3.5 we will apply our analysis by presenting Algorithm 3.3 (`QuadPiStar`) for approximating $\pi^*(x;\lambda)$.

Theorem 3.16 *Let*

$$S(x) := S(x;\lambda,\sigma,h) := \frac{h}{2\pi}\sum_{k\in\mathbb{Z}} \Psi(\sigma+ikh;x,\lambda) \tag{3.43}$$

$$I_{\mathcal{L}} := I_{\mathcal{L}}(x) := I_{\mathcal{L}}(x;\lambda,\sigma,h) := \sum_{k\geq 1} e^{-2\pi k\sigma/h}\pi^*(e^{2\pi k/h}x;\lambda) \tag{3.44}$$

$$I_{\mathcal{R}} := I_{\mathcal{R}}(x) := I_{\mathcal{R}}(x;\lambda,\sigma,h) := \sum_{k\geq 1} e^{2\pi k\sigma/h}\pi^*(e^{-2\pi k/h}x;\lambda). \tag{3.45}$$

Then, for fixed $\sigma > 1$ and $h > 0$ we have

$$S(x;\lambda,\sigma,h) = \sum_{k\in\mathbb{Z}} e^{2\pi\sigma k/h}\pi^*(e^{-2\pi k/h}x;\lambda), \tag{3.46}$$

or, equivalently,

$$\pi^*(x;\lambda) = S(x;\lambda,\sigma,h) - I_{\mathcal{L}}(x;\lambda,\sigma,h) - I_{\mathcal{R}}(x;\lambda,\sigma,h). \tag{3.47}$$

Proof: We will use a version of the Poisson summation formula. Recalling from Section 1.2 our convention for writing infinite sums which may be conditionally convergent, the Poisson summation formula may be stated as

$$\sum_{k\in\mathbb{Z}} g(k) = \sum_{k\in\mathbb{Z}} \int_{-\infty}^{\infty} e^{-2\pi ikz} g(z)\,dz, \tag{3.48}$$

provided the following three conditions hold [Hen91, §10.6.IV]:

1. for some $\delta > 0$, $g(z)$ is analytic in the strip $|\operatorname{Im} z| < \delta$;
2. $\sum_{k\in\mathbb{Z}} g(z+k)$ converges uniformly for z in the rectangle $0 \leq \operatorname{Re} z \leq 1$, $|\operatorname{Im} z| < \delta$;

3. $\int_{-\infty}^{\infty} |g(z)|\, dz < \infty$.

Recall that σ is fixed and let $g(z) := \Psi(\sigma + ihz; x, \lambda)$. Since $\sigma > 1$, letting $\delta := (\sigma - 1)/(2h)$ we see that $g(z)$ satisfies condition 1. From Equation (2.17) we see that, uniformly for $|\mathrm{Im}(z)| < \delta$, we have $g(z) \ll \exp\big(-(\lambda h \,\mathrm{Re}(z))^2/2\big)$, from which it follows that conditions 2 and 3 are satisfied. Thus, Equation (3.48) holds. Rewriting (3.48) in terms of s and $\Psi(s)$ and multiplying both sides by $h/(2\pi)$, gives

$$(3.49)\quad \frac{h}{2\pi} \sum_{k \in \mathbb{Z}} \Psi(\sigma + ikh) = \sum_{k \in \mathbb{Z}} \frac{e^{2\pi k \sigma/h}}{2\pi i} \int_{\sigma - i\infty}^{\sigma + i\infty} e^{-2\pi k s/h} \widehat{\phi}(s; x, \lambda) \ln \zeta(s)\, ds.$$

By Equation (2.23) on page 29, and noting that

$$e^{-2\pi k s/h} \widehat{\phi}(s; x, \lambda) = \widehat{\phi}(s; e^{-2\pi k/h} x, \lambda),$$

the right side of (3.49) is $= \sum_{k \in \mathbb{Z}} e^{2\pi \sigma k/h} \pi^*(e^{-2\pi k/h} x; \lambda)$, establishing (3.46). Collecting terms with $k < 0$, $k = 0$, $k > 0$, from (3.46) gives $S(x) = I_{\mathcal{L}}(x) + \pi^*(x; \lambda) + I_{\mathcal{R}}(x)$, and Equation (3.47) follows upon rearrangement. ■

Remark Note that both $I_{\mathcal{L}}$ and $I_{\mathcal{R}}$ are positive. Thus Equation (3.47) implies that $\pi^*(x; \lambda)$ is always overestimated by the approximating Riemann sum $S(x; \lambda, \sigma, h)$. □

Equation (3.47) serves as the basis of our quadrature algorithm for approximating $\pi^*(x; \lambda)$. Treating $I_{\mathcal{L}}$ and $I_{\mathcal{R}}$ as errors to be bounded, if we define $h_{\mathcal{L}} = h_{\mathcal{L}}(x, \lambda, \sigma, \varepsilon)$ and $h_{\mathcal{R}} = h_{\mathcal{R}}(x, \lambda, \sigma, \varepsilon)$ so that $I_{\mathcal{L}}(x; \lambda, \sigma, h_{\mathcal{L}}) \le \varepsilon$ and $I_{\mathcal{R}}(x; \lambda, \sigma, h_{\mathcal{R}}) \le \varepsilon$, it suffices to have $h \le \min(h_{\mathcal{L}}, h_{\mathcal{R}})$ to ensure that

$$(3.50)\qquad \pi^*(x; \lambda) = S(x; \lambda, \sigma, h) + 2\varepsilon\, O(1).$$

However, we also describe and partially analyze a more sophisticated quadrature method that offers the possibility of using a larger value of h. This method incorporates "quadrature correction terms", based on the following expansion for $I_{\mathcal{R}}(x; \lambda, \sigma, h)$, arising from Equation (3.45) and the observation that $\pi^*(x;\lambda) = \pi(x) - \Delta(x;\lambda)$:

$$
\begin{aligned}
I_{\mathcal{R}}(x;\lambda,\sigma,h) &= \sum_{k=1}^{K} e^{2\pi k\sigma/h}\pi^*(e^{-2\pi k/h}x;\lambda) + e^{2\pi K\sigma/h} I_{\mathcal{R}}(e^{-2\pi k/h}x;\lambda,\sigma,h) \\
&= \sum_{k=1}^{K} e^{2\pi k\sigma/h}\big(\pi(e^{-2\pi k/h}x) - \Delta(e^{-2\pi k/h}x;\lambda)\big) \\
&\qquad + e^{2\pi K\sigma/h} I_{\mathcal{R}}(e^{-2\pi k/h}x;\lambda,\sigma,h).
\end{aligned} \tag{3.51}
$$

(Note that when $K = 0$, Equation (3.51) reduces to the trivial statement that $I_{\mathcal{R}}(x;\lambda,\sigma,h) = I_{\mathcal{R}}(x;\lambda,\sigma,h)$, under the convention that $\sum_{k=1}^{0}\ldots$ denotes an empty sum.)

Redefining $h_{\mathcal{R}} = h_{\mathcal{R}}(x,\lambda,\sigma,\varepsilon,K)$ so that it also depends on the parameter $K \in \mathbb{Z}_0$, and requiring that

$$
e^{2\pi K\sigma/h_{\mathcal{R}}} I_{\mathcal{R}}(e^{-2\pi k/h_{\mathcal{R}}}x;\lambda,\sigma,h_{\mathcal{R}}) \le \varepsilon, \tag{3.52}
$$

it suffices to have $h \le \min(h_{\mathcal{L}}, h_{\mathcal{R}})$ to ensure that the following generalization of Equation (3.50) holds:

$$
\begin{aligned}
&\pi^*(x;\lambda) = \\
&S(x;\lambda,\sigma,h) - \sum_{k=1}^{K} e^{2\pi k\sigma/h}\big(\pi(e^{-2\pi k/h}x) - \Delta(e^{-2\pi k/h}x;\lambda)\big) + 2\varepsilon\, O(1).
\end{aligned} \tag{3.53}
$$

Provided that $e^{-2\pi k/h}x$ is significantly smaller than x, each term of the form $\pi(e^{-2\pi k/h}x) - \Delta(e^{-2\pi k/h}x;\lambda)$ in Equation (3.53) can be computed by taking a sum over primes. Thus, each term contributes "number theoretical" information to refine the accuracy of our numerical quadrature.

In Section 3.6, we will find that the optimal choice for K in (3.53) is very dependent on the method used to compute $\zeta(s)$, and that for any reasonable algorithm for $\zeta(s)$ it is doubtful that we would want $K > 2$. Based on our current understanding of the fastest methods of computing $\zeta(s)$, it is likely that we will want $K = 0$, but we spend some effort in analyzing our more sophisticated method both in anticipation that better methods for computing $\zeta(s)$ may be found and because the conditions leading to the choice $K = 0$ lie close to the boundary where we would choose $K = 1$.

Although we will not give a detailed analysis of the computational cost of approximating the $\sum_K \dots$ in Equation (3.53), we believe the cost will be low. In Remark 3.35, on page 74, we discuss some of the issues that should be dealt with to implement Equation (3.53).

In this section, we will further discuss how to choose the parameter K after dealing with defining suitable $h_{\mathcal{L}}$ and $h_{\mathcal{R}}$. Turning now to the definition of $h_{\mathcal{L}}$, we begin by bounding $I_{\mathcal{L}}$ as a function of h:

Lemma 3.17 *Given $\sigma > 1$, $h > 0$, we have*

$$I_{\mathcal{L}}(x; \lambda, \sigma, h) \le e^{\lambda^2/2} x \frac{e^{-2\pi(\sigma-1)/h}}{1 - e^{-2\pi(\sigma-1)/h}}. \tag{3.54}$$

Proof: Applying the bound (3.40) to the right side of (3.44) and summing the resulting geometric series, we find

$$I_{\mathcal{L}}(x; \lambda, \sigma, h) \le \sum_{k \ge 1} e^{-2\pi k\sigma/h} e^{\lambda^2/2} e^{2\pi k/h} x = e^{\lambda^2/2} x \frac{e^{-2\pi(\sigma-1)/h}}{1 - e^{-2\pi(\sigma-1)/h}}.$$

The series converges by our assumptions that $\sigma > 1$, $h > 0$. ■

As in the method used in the previous section to choose x_1, x_2 in Algorithm 3.2 (`Delta`); one way of finding $h_{\mathcal{L}}$ to ensure that $I_{\mathcal{L}}(x; \lambda, \sigma, h_{\mathcal{L}}) \le \varepsilon$ would be to numerically solve for h to make the right side of (3.54) equal to ε, and to then set $h_{\mathcal{L}} = h$. However, we prefer to define $h_{\mathcal{L}}$ by Equation (3.55), below, since this gives an analytically tractable expression that works well.

Theorem 3.18 *Given $x > 0$, $\lambda > 0$, $\sigma > 1$, $0 < \varepsilon \le 1$, let*

$$h_{\mathcal{L}} := h_{\mathcal{L}}(x, \lambda, \sigma, \varepsilon) := \frac{2\pi(\sigma - 1)}{\ln(x/\varepsilon) + \lambda^2/2 + 1/x}. \tag{3.55}$$

Then for $0 < h \le h_{\mathcal{L}}$ we have $I_{\mathcal{L}}(x; \lambda, \sigma, h) \le \varepsilon$.

Proof: Assuming $h \le h_{\mathcal{L}}$, a short calculation gives

$$e^{-2\pi(\sigma-1)/h} \le \varepsilon\, e^{-\lambda^2/2} e^{-1/x}/x. \tag{3.56}$$

Since we assume $\varepsilon \le 1$, and since $e^{-\lambda^2/2} \le 1$, the bound (3.56) gives

$$\frac{1}{1 - e^{-2\pi(\sigma-1)/h}} \le \frac{1}{1 - e^{-1/x}/x}. \tag{3.57}$$

Using the bounds (3.56) and (3.57) in (3.54), we find

$$I_{\mathcal{L}}(x;\lambda,\sigma,h) \le e^{\lambda^2/2}x\frac{e^{-2\pi(\sigma-1)/h}}{1-e^{-2\pi(\sigma-1)/h}} \le \frac{\varepsilon}{e^{1/x}-1/x} \le \varepsilon,$$

where we have used the fact that $e^\alpha \ge 1+\alpha$ for real α. ■

We now turn to choosing $h_{\mathcal{R}}$. As suggested by Figure 3.2(a) on the following page, $I_{\mathcal{R}}(x;\lambda,\sigma,h)$ grows irregularly as a function of h. For this reason, we will not present an analogue to Lemma 3.17 above. However, as illustrated in Figure 3.2(b), $I_{\mathcal{R}}(x;\lambda,\sigma,h)$ behaves nicely when $e^{-2\pi/h}x < 2$, since the $k=1$ term dominates all other terms in Equation (3.45) (the defining sum for $I_{\mathcal{R}}$). Thus, generalizing the above remarks to arbitrary $K \in \mathbb{Z}_0$, we ensure $e^{-2\pi(K+1)/h}x < 2$ by our choice of $h_{\mathcal{R}}$ in Theorem 3.19, below.

Theorem 3.19 *Given $x \ge 2$, $\lambda > 0$, $\sigma > 1$, $0 < \varepsilon \le 1$, and $K \in \mathbb{Z}_0$; let*

$$\begin{aligned} h_{\mathcal{R}} &:= h_{\mathcal{R}}(x,\lambda,\sigma,\varepsilon,K) \\ &:= \frac{2\pi(K+1)}{\ln(x/2)+\sigma\lambda^2+\lambda\sqrt{2(K+1)}\sqrt{\sigma\ln(x/2)+\sigma^2\lambda^2/2+\ln(3.4/\varepsilon)}}. \end{aligned} \tag{3.58}$$

Then for $0 < h \le h_{\mathcal{R}}$ we have $e^{2\pi K\sigma/h}I_{\mathcal{R}}(e^{-2\pi K/h}x;\lambda,\sigma,h) \le \varepsilon$.

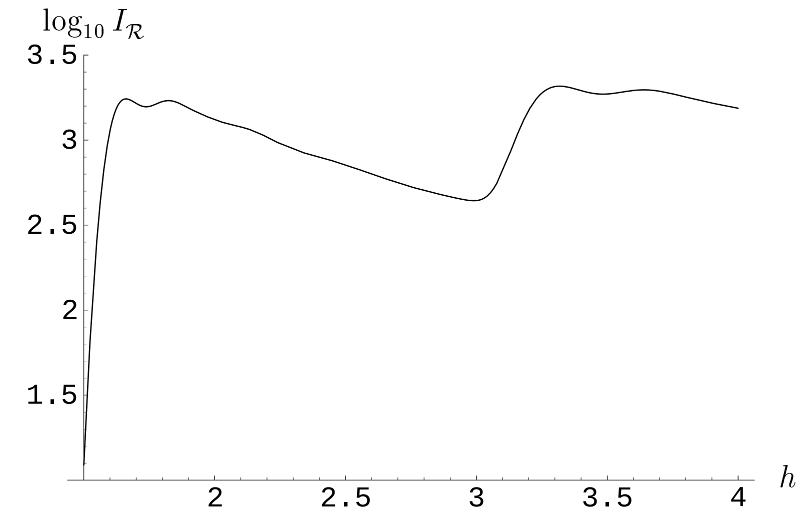
Proof: In this proof we will let $\tau_0 := \ln(2/x)$ and $c := 2\pi/h$. Since we assume $h \le h_{\mathcal{R}}$, Equation (3.58) and $\sigma > 1$ imply that

$$\begin{aligned} c &\ge \frac{\ln(x/2)+\sigma\lambda^2}{K+1} + \lambda\sqrt{2/(K+1)}\sqrt{\sigma\ln(x/2)+\sigma^2\lambda^2/2+\ln(3.4/\varepsilon)} \\ &\ge (\ln(x/2)+\lambda^2)/(K+1), \end{aligned} \tag{3.59}$$

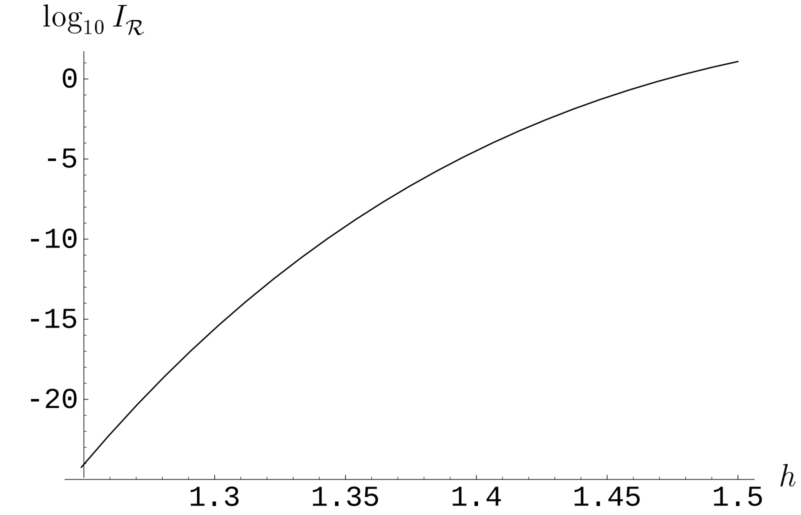
and so $e^{-kc}x \le 2e^{-\lambda^2}$ for $k \ge K+1$. For $e^{-kc}x \le 2e^{-\lambda^2}$ we may apply the bound (3.41) of Theorem 3.15 on page 52 to find that for $k \ge K+1$

$$\pi^*(e^{-kc}x;\lambda) \le 1.7\exp\left(-(\tau_0+kc)^2/(2\lambda^2)\right). \tag{3.60}$$

Rewriting Equation (3.45) from page 54 in terms of c, and then applying



(a) h over a "bad" range



(b) h over a "nice" range where h satisfies $e^{-2\pi/h}x < 2$

Figure 3.2: $\log_{10} I_{\mathcal{R}}(x; \lambda, \sigma, h)$ as a function of h. $x = 100$, $\lambda = 0.1$, $\sigma = 2$. Note $e^{-2\pi/h}x = 2$ near $h = 1.606\ldots$.

the bound (3.60) gives

$$
\begin{aligned}
e^{2\pi K\sigma/h} I_{\mathcal{R}}(e^{-2\pi K/h}x;\lambda,\sigma,h) &= \sum_{k\geq K+1} e^{k\sigma c}\pi^*(e^{-kc}x;\lambda) \\
&\leq 1.7 \sum_{k\geq K+1} \exp\left(\frac{2\lambda^2 k\sigma c-(\tau_0+kc)^2}{2\lambda^2}\right). \qquad (3.61)
\end{aligned}
$$

The numerator of the argument of $\exp(\dots)$ in (3.61) is

$$
\begin{aligned}
2\lambda^2 k\sigma c-(\tau_0+kc)^2 &= -\tau_0^2-2k(\tau_0-\lambda^2\sigma)c-k^2c^2 \\
&= \lambda^4\sigma^2-2\lambda^2\sigma\tau_0-(\tau_0-\lambda^2\sigma+kc)^2, \qquad (3.62)
\end{aligned}
$$

so, substituting (3.62) into the bound (3.61) and noting that $e^{-\sigma\tau_0}=(x/2)^\sigma$, we find that

$$
\begin{aligned}
& e^{2\pi K\sigma/h} I_{\mathcal{R}}(e^{-2\pi K/h}x;\lambda,\sigma,h) \\
& \quad \leq 1.7e^{\lambda^2\sigma^2/2}(x/2)^\sigma \sum_{k\geq K+1} \exp\left(-\frac{(\tau_0-\sigma\lambda^2+kc)^2}{2\lambda^2}\right). \qquad (3.63)
\end{aligned}
$$

Now, the bound (3.59) gives

$$
\begin{aligned}
&\tau_0-\sigma\lambda^2+kc \\
&\geq \frac{k-K-1}{K+1}(-\tau_0+\sigma\lambda^2)+k\lambda\sqrt{\frac{2}{K+1}}\sqrt{\sigma\ln(x/2)+\sigma^2\lambda^2/2+\ln(3.4/\varepsilon)}.
\end{aligned}
$$

Our assumption that $x\geq 2$ implies $-\tau_0=\ln(x/2)\geq 0$, so, for $k\geq K+1$, the preceding bound is

$$
\geq k\lambda\sqrt{2/(K+1)}\sqrt{\sigma\ln(x/2)+\sigma^2\lambda^2/2+\ln(3.4/\varepsilon)}.
$$

Squaring and rearranging this, and then noting that $-k^2\leq -k(K+1)$ for $k\geq K+1$, we find

$$
\begin{aligned}
-\frac{(\tau_0-\sigma\lambda^2+kc)^2}{2\lambda^2} &\leq -k^2\left(\sigma\ln(x/2)+\sigma^2\lambda^2/2+\ln(3.4/\varepsilon)\right)/(K+1) \\
&\leq -k\ln\left(3.4e^{\lambda^2\sigma^2/2}(x/2)^\sigma\varepsilon^{-1}\right). \qquad (3.64)
\end{aligned}
$$

Using the exponentiation of (3.64) to bound each term in (3.63) gives

$$
\begin{aligned}
&e^{2\pi K\sigma/h} I_{\mathcal{R}}(e^{-2\pi K/h}x;\lambda,\sigma,h) \\
&\quad \le 1.7 e^{\lambda^2\sigma^2/2}(x/2)^\sigma \sum_{k\ge K+1} \bigl(3.4e^{\lambda^2\sigma^2/2}(x/2)^\sigma\varepsilon^{-1}\bigr)^{-k} \\
&\quad = \frac{\varepsilon}{2}\bigl(3.4e^{\lambda^2\sigma^2/2}(x/2)^\sigma\varepsilon^{-1}\bigr)^{-K} \sum_{k\ge 0} \bigl(3.4e^{\lambda^2\sigma^2/2}(x/2)^\sigma\varepsilon^{-1}\bigr)^{-k} \\
&\quad = \frac{\varepsilon}{2}\,\frac{\bigl(3.4e^{\lambda^2\sigma^2/2}(x/2)^\sigma\varepsilon^{-1}\bigr)^{-K}}{1-\bigl(3.4e^{\lambda^2\sigma^2/2}(x/2)^\sigma\bigr)^{-1}\varepsilon}. \qquad (3.65)
\end{aligned}
$$

Finally, the conditions of our theorem (including the condition that $\varepsilon \le 1$) imply $\bigl(3.4e^{\lambda^2\sigma^2/2}(x/2)^\sigma\bigr)^{-1}\varepsilon \le 1/3.4 < 1/2$, so from (3.65) it follows that

$$e^{2\pi K\sigma/h} I_{\mathcal{R}}(e^{-2\pi K/h}x;\lambda,\sigma,h) \le \varepsilon. \qquad \blacksquare$$

As we have seen in Figure 3.2(b), as h grows smaller than $h_{\mathcal{R}}$ the value of $I_{\mathcal{R}}$ drops very rapidly. Theorem 3.20, below, quantifies this observation.

Theorem 3.20 *Given $x \ge 2$, $\lambda > 0$, $\sigma > 1$, $0 < \varepsilon \le 1$, and $K \in \mathbb{Z}_0$; if*

$$h \le \frac{2\pi K}{\ln(x/2) + \sigma\lambda^2} \qquad (3.66)$$

then $e^{2\pi K\sigma/h} I_{\mathcal{R}}(e^{-2\pi K/h}x;\lambda,\sigma,h) \ll \exp\bigl(-2\pi^2/(\lambda^2h^2)\bigr)$.

Proof: Much of our proof is similar to the proof of Theorem 3.19. As in that proof, we let $\tau_0 := \ln(2/x)$ and $c := 2\pi/h$.

Now, the bound (3.66) and $\sigma > 1$ imply that

$$c \ge (-\tau_0 - \lambda^2)/K, \qquad (3.67)$$

and it follows that $e^{-kc}x \le 2e^{-\lambda^2}$ for $k \ge K$. Thus, we can apply the bound (3.41) to again verify that Equations (3.60) through (3.63) hold.

From the bound (3.67) we have, for $k \ge K$,

$$\tau_0 - \sigma\lambda^2 + kc = \tau_0 - \sigma\lambda^2 + Kc + (k-K)c \ge (k-K)c. \qquad (3.68)$$

Using the rightmost bound of (3.68) in each term of (3.63), recalling that

$c := 2\pi/h$, and letting $j = k - K$ gives, as claimed,

$$\begin{aligned} e^{2\pi K\sigma/h} I_{\mathcal{R}}(e^{-2\pi K/h}x; \lambda, \sigma, h) &\le 1.7 e^{\lambda^2\sigma^2/2}(x/2)^{\sigma} \sum_{j\ge 1} e^{-2\pi^2 j^2/(\lambda^2 h^2)} \\ &\ll e^{-2\pi^2/(\lambda^2h^2)} + \sum_{j\ge 2} e^{-2j/(\lambda^2h^2)} \ll e^{-2\pi^2/(\lambda^2h^2)}. \end{aligned}$$ ■

Remark Taken together, the bounds of Lemma 3.17 and Theorem 3.20 imply that $I_{\mathcal{L}}$ dominates $I_{\mathcal{R}}$ as $h \to 0$ and that

$$\begin{aligned} &\pi^*(x; \lambda) \\ &= S(x; \lambda, \sigma, h) - \sum_{k=1}^{K} e^{2\pi k\sigma/h}\big(\pi(e^{-2\pi k/h}x) - \Delta(e^{-2\pi k/h}x; \lambda)\big) + O(e^{-2\pi(\sigma-1)/h}). \end{aligned}$$

That is, the trapezoidal rule approximation to the integral representation of $\pi^*(x; \lambda)$ converges exponentially in $1/h$. This rapid convergence as $h \to 0$ applies whenever the trapezoidal rule is used to approximate the integral of a function which is analytic in a strip about the line of integration. In Chapter 7 we again take advantage of this rapid convergence. Further discussion of this phenomenon, and other applications, can be found in [Ste93]. □

To approximate $\pi^*(x; \lambda)$, Algorithm 3.3 (`QuadPiStar`) implements Equation (3.50) (i.e., Equation (3.53) with $K = 0$). Since $K = 0$ is fixed, it first computes $h_{\mathcal{L}}$ and $h_{\mathcal{R}}$ to bound the error term in Equation (3.53), and then sets $h = \min(h_{\mathcal{L}}, h_{\mathcal{R}})$.

To implement the more sophisticated method implied by the general case of Equation (3.53) where $K > 0$, we need a method for choosing K. To do this, we propose to first use Equation (3.55) to find $h_{\mathcal{L}}(x, \lambda, \sigma, \varepsilon)$. We then set $h = h_{\mathcal{L}}(x, \lambda, \sigma, \varepsilon)$ and use Equation (3.58) to find an integer K such that $h \le h_{\mathcal{R}}(x, \lambda, \sigma, \varepsilon, K)$. Theorem 3.21, below, gives the details.

Theorem 3.21 *Let $h_{\mathcal{R}}(x, \lambda, \sigma, \varepsilon, K)$ be defined by Equation* (3.58)*. Given $h > 0$, $x \ge 2$, $\lambda > 0$, $\sigma > 1$, and $0 < \varepsilon \le 1$; K satisfies $h \le h_{\mathcal{R}}(x, \lambda, \sigma, \varepsilon, K)$ provided*

$$K \ge -1 + \frac{C_1 h}{2\pi} + \frac{C_2 h^2}{8\pi^2}\left(1 + \sqrt{1 + 8\pi C_1/(C_2 h)}\right), \tag{3.69}$$

where

$$C_1 := \ln(x/2) + \sigma\lambda^2, \quad \textit{and} \tag{3.70}$$

$$C_2 := 2\lambda^2 \left(\sigma \ln(x/2) + \sigma^2\lambda^2/2 + \ln(3.4/\varepsilon)\right). \tag{3.71}$$

Proof: Within the body of this proof we treat K as an element of $\mathbb{R}$, and let $\alpha := K + 1$. Having used C_1 and C_2 to abbreviate the ponderous expressions in Equation (3.58), which defines $h_{\mathcal{R}}(x, \lambda, \sigma, \varepsilon, K)$, we find that $h \leq h_{\mathcal{R}}(x, \lambda, \sigma, \varepsilon, K)$ is equivalent to $h \leq 2\pi\alpha/(C_1 + \sqrt{C_2\alpha})$, which holds provided

$$\sqrt{C_2\alpha} \leq 2\pi\alpha/h - C_1. \tag{3.72}$$

Since $h > 0$, $C_1 > 0$, and $C_2 > 0$, it is clear that the inequality (3.72) holds for large enough α. Focusing on the value of α which gives equality in (3.72), and then squaring both sides, gives a quadratic equation which the right side of (3.69) solves when rewritten in terms of α. It is also clear that the value of α solving $-\sqrt{C_2\alpha} = 2\pi\alpha/h - C_1$ is less than the value solving $+\sqrt{C_2\alpha} = 2\pi\alpha/h - C_1$, which confirms that we chose the correct quadratic root for (3.69). ■

With C_1 and C_2 defined by (3.70) and (3.71), it follows from (3.69) that we get an error term of $2\varepsilon\, O(1)$ in Equation (3.53) by setting $h = h_{\mathcal{L}}(x, \lambda, \sigma, \varepsilon)$ and then setting

$$K = -1 + \left\lceil \frac{C_1 h}{2\pi} + \frac{C_2 h^2}{8\pi^2}\left(1 + \sqrt{1 + 8\pi C_1/(C_2 h)}\right)\right\rceil. \tag{3.73}$$

This expression for K becomes less opaque if we recall that we will choose $\lambda = x^{-1/2+o(1)}$ as $x \to \infty$. Without giving a careful analysis, if we once again treat K as a real variable and treat all terms as negligible if the contain a factor of λ^2 or x^{-1}, we find from Equation (3.55) (defining $h_{\mathcal{L}}$) and from (3.73) that as $x \to \infty$ we have

$$K \approx -1 + (\sigma - 1)\frac{\ln(x/2)}{\ln(x/\varepsilon)} \approx \sigma - 2. \tag{3.74}$$

If we hold x, λ, and ε fixed and treat K as a function of σ, then K as defined in Equation (3.73) changes value at points where $h_{\mathcal{L}}(x, \lambda, \sigma, \varepsilon)$ crosses

$h_{\mathcal{R}}(x, \lambda, \sigma, \varepsilon, K)$. These intersection points are illustrated in Figure 3.3 below. Note that the points at which $h_{\mathcal{L}}$ and $h_{\mathcal{R}}(\ldots, K)$ intersect are well-estimated by the middle expression in the approximation (3.74).

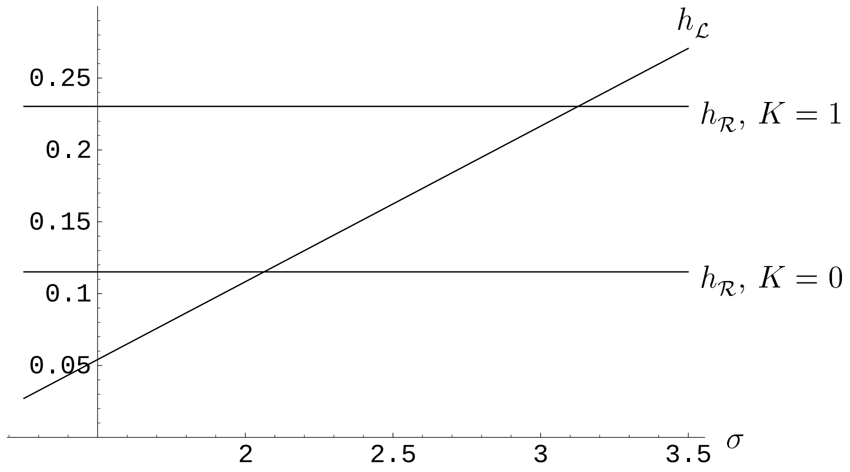


Figure 3.3: $h_{\mathcal{L}}(x, \lambda, \sigma, \varepsilon)$ and $h_{\mathcal{R}}(x, \lambda, \sigma, \varepsilon, K)$, $K = 0, 1$, as functions of σ. $x = 10^{24}$, $\lambda = 10^{-12}$, $\varepsilon = 1/16$. For $K = 0$, $h_{\mathcal{L}} = h_{\mathcal{R}} \approx 0.1151$ near $\sigma = 2.0635$. For $K = 1$, $h_{\mathcal{L}} = h_{\mathcal{R}} \approx 0.2303$ near $\sigma = 3.127$.

3.4 Approximating the integral by a finite sum

Recall that

$$\Psi(s) := \Psi(s; x, \lambda) := \widehat{\phi}(s; x, \lambda) \ln \zeta(s) = e^{\lambda^2 s^2/2} \, \frac{x^s}{s} \ln \zeta(s),$$

where the parameters x, λ are assumed to be fixed when not explicitly given.

We now turn to the analysis of the error introduced by approximating the infinite sum

$$S(x; \lambda, \sigma, h) := \frac{h}{2\pi} \sum_{k \in \mathbb{Z}} \Psi(\sigma + ikh) \tag{3.75}$$

with the finite sum

$$\frac{h}{2\pi} \sum_{|kh| \le T} \Psi(\sigma + ikh). \tag{3.76}$$

We halve the number of quadrature points in (3.76), and simplify some of our analysis, by noting that since $\Psi(\overline{s};x,\lambda)=\overline{\Psi(s;x,\lambda)}$, the sum (3.76) is

$$= \frac{h}{\pi}\left(\frac{1}{2}\Psi(\sigma;x,\lambda)+\sum_{k=1}^{\lfloor T/h\rfloor}\operatorname{Re}\Psi(\sigma+ikh;x,\lambda)\right). \tag{3.77}$$

After an initial lemma, we will bound $|\operatorname{Re}\Psi(s;x,\lambda)|$ in Lemma 3.23.

Lemma 3.22 *Let $s=\sigma+it$, $\sigma>1$. Then*

$$|\ln\zeta(s)|\le\ln\zeta(\sigma), \tag{3.78}$$

and

$$\left|\frac{d}{ds}\ln\zeta(s)\right|\le|\zeta'(\sigma)/\zeta(\sigma)|, \tag{3.79}$$

where $\zeta'(s)$ denotes $d/ds\,\zeta(s)$.

Proof: Since $\sigma>1$, Euler's product formula gives

$$|\ln\zeta(s)|=\left|-\sum_p\ln(1-p^{-s})\right|=\left|\sum_p\sum_{m\ge1}\frac{p^{-ms}}{m}\right|\le\sum_p\sum_{m\ge1}\frac{p^{-m\sigma}}{m}=\ln\zeta(\sigma),$$

establishing the bound (3.78).

Similarly, we establish the bound (3.79) by noting that

$$\begin{aligned}\left|\frac{d}{ds}\ln\zeta(s)\right|&=\left|-\frac{d}{ds}\sum_p\ln(1-p^{-s})\right|=\left|\sum_p\ln(p)p^{-s}\sum_{m\ge1}p^{-ms}\right|\\&\le\sum_p\ln(p)p^{-\sigma}\sum_{m\ge1}p^{-m\sigma}=\left|\frac{d}{d\sigma}\ln\zeta(\sigma)\right|=|\zeta'(\sigma)/\zeta(\sigma)|.\end{aligned}$$ ■

Lemma 3.23 establishes two, slightly different, bounds to be used below.

Lemma 3.23 *Let $s=\sigma+it$, $\sigma>1$, $t>0$, $\lambda\ge0$. Then*

$$|\operatorname{Re}\Psi(s;x,\lambda)|\le|\Psi(s;x,\lambda)|\le e^{\lambda^2\sigma^2/2}x^\sigma\ln\zeta(\sigma)e^{-\lambda^2t^2/2}, \tag{3.80}$$

and

$$|\operatorname{Re}\Psi(s;x,\lambda)|\le|\Psi(s;x,\lambda)|\le e^{\lambda^2\sigma^2/2}x^\sigma\ln\zeta(\sigma)e^{-\lambda^2t^2/2}/t. \tag{3.81}$$

Proof: It is obvious that $|\mathrm{Re}\,\Psi(s)| \le |\Psi(s)|$.

Recalling that $\Psi(s;x,\lambda) = \widehat{\phi}(s;x,\lambda)\ln\zeta(s)$, we have $|\ln\zeta(s)| \le \ln\zeta(\sigma)$ by (3.78). Furthermore, $|\widehat{\phi}(s;x,\lambda)| \le e^{\lambda^2(\sigma^2-t^2)/2}x^\sigma$ by the bound (2.19) on page 27; while $|\widehat{\phi}(s;x,\lambda)| \le e^{\lambda^2(\sigma^2-t^2)/2}x^\sigma/t$ by the bound (2.20), and using $t > 0$. The results follow upon taking the product of the bound on $\ln\zeta(s)$ with the two bounds on $|\widehat{\phi}(s;x,\lambda)|$. ■

In analogy to our error bounds $\mathcal{E}_-(u;x,\lambda)$ and $\mathcal{E}_+(u;x,\lambda)$ of Section 3.2, we define an error bound $\mathcal{E}_\Sigma(T;x,\lambda,\sigma)$, for the tail of the sum (3.75):

Definition 3.24 Given $T > 0$, $x > 0$, $\lambda > 0$, $\sigma > 1$, let

$$\mathcal{E}_\Sigma(T) := \mathcal{E}_\Sigma(T;x,\lambda,\sigma) := \frac{1}{2\pi}e^{\lambda^2\sigma^2/2}x^\sigma\ln\zeta(\sigma)E_1(\lambda^2T^2/2), \tag{3.82}$$

where $E_1(z)$ is the exponential integral, $E_1(z) := \int_z^\infty e^{-r}/r\,dr$. □

Remark Formulae for computing $E_1(z)$ are given in [AS92, Chapter 5] and in [PTVF92, §6.3]. □

Theorem 3.25 *Given $x > 0$, $\lambda > 0$, $\sigma > 1$, $h > 0$, and $T > 0$ with the restriction that T be a positive integer multiple of h, then we have*

$$\frac{h}{\pi}\sum_{kh>T}|\mathrm{Re}\,\Psi(\sigma+ikh;x,\lambda)| \le \mathcal{E}_\Sigma(T;x,\lambda,\sigma). \tag{3.83}$$

Proof: Bounding the summand by the right side of (3.81), noting that this bound is decreasing in t, and since we assume $T/h \in \mathbb{N}$, we have

$$\begin{aligned}
&\frac{h}{\pi}\sum_{kh>T}|\mathrm{Re}\,\Psi(\sigma+ikh;x,\lambda)| \\
&\le \frac{1}{\pi}e^{\lambda^2\sigma^2/2}x^\sigma\ln\zeta(\sigma)\int_T^\infty e^{-\lambda^2t^2/2}\frac{dt}{t} = \frac{1}{2\pi}e^{\lambda^2\sigma^2/2}x^\sigma\ln\zeta(\sigma)E_1(\lambda^2T^2/2) \\
&= \mathcal{E}_\Sigma(T;x,\lambda,\sigma).
\end{aligned}$$ ■

We now present two corollaries of Theorem 3.25. The first: Corollary 3.26, justifies the choice of T which we will use in Algorithm 3.3.

Corollary 3.26 *Given $T > 0$ with $\mathcal{E}_\Sigma(T;x,\lambda,\sigma) \le \varepsilon$ we have*

$$S(x;\lambda,\sigma,h) = \frac{h}{\pi}\left(\frac{1}{2}\Psi(\sigma;x,\lambda) + \sum_{k=1}^{\lceil T/h\rceil}\mathrm{Re}\,\Psi(\sigma+ikh;x,\lambda)\right) + \varepsilon\,O(1). \tag{3.84}$$

Proof: As z increases $E_1(z)$ decreases, so $\mathcal{E}_\Sigma(h\lceil T/h\rceil; x, \lambda) \le \mathcal{E}_\Sigma(T; x, \lambda, \sigma)$. Since $h\lceil T/h\rceil$ is an integer multiple of h, Equation (3.84) follows immediately from Theorem 3.25. ■

As an aid to the error analysis of Algorithm 3.3, our second corollary of Theorem 3.25: Corollary 3.27, gives a bound for the partial sums in Formula (3.84).

Corollary 3.27 *Given $0 \le \lambda \le 1/2$, $\sigma > 1$, and $h > 0$, for any $k_1 > 0$ we have*

$$\begin{aligned} &\frac{1}{2}\Psi(\sigma; x, \lambda) + \sum_{k=1}^{k_1} \operatorname{Re}\Psi(\sigma + ikh; x, \lambda) \\ &\qquad \le \left(\frac{3}{2} + \frac{1}{2h}\left(17/8 + \ln(2/\lambda^2)\right)\right) e^{\lambda^2\sigma^2/2} x^\sigma \ln\zeta(\sigma). \end{aligned} \tag{3.85}$$

Proof: First focusing on the sum over k, we have

$$\begin{aligned} \sum_{k=1}^{k_1} \operatorname{Re}\Psi(\sigma + ikh; x, \lambda) &\le \sum_{k\ge 1} |\Psi(\sigma + ikh; x, \lambda)| \\ &= \sum_{k=1}^{\lceil 1/h\rceil} |\Psi(\sigma + ikh; x, \lambda)| + \sum_{k>\lceil 1/h\rceil} |\Psi(\sigma + ikh; x, \lambda)|. \end{aligned} \tag{3.86}$$

By the bound (3.80), the first subsum in (3.86) is

$$\le \sum_{k=1}^{\lceil 1/h\rceil} e^{\lambda^2\sigma^2/2} x^\sigma \ln\zeta(\sigma) e^{-\lambda^2k^2h^2/2} \le (1 + 1/h) e^{\lambda^2\sigma^2/2} x^\sigma \ln\zeta(\sigma). \tag{3.87}$$

Exactly as in the proof of Theorem 3.25, we find that the second subsum of Equation (3.86) is

$$\le \frac{\pi}{h}\mathcal{E}_\Sigma(1; x, \lambda, \sigma) = \frac{1}{2h} e^{\lambda^2\sigma^2/2} x^\sigma \ln\zeta(\sigma) E_1(\lambda^2/2). \tag{3.88}$$

From [AS92, Entry 5.1.20] we have $E_1(z) < \ln(1 + 1/z)$ for $z > 0$. Now, $\ln(1 + 1/z) = \ln(1/z) + \ln(1 + z)$ and $\ln(1 + z) \le z$. Furthermore, we may assume that $z \le 1/8$ since $\lambda^2 \le 1/2$. It follows that (3.88) is

$$\le \frac{1}{2h}(1/8 + \ln(2/\lambda^2)) e^{\lambda^2\sigma^2/2} x^\sigma \ln\zeta(\sigma). \tag{3.89}$$

The bound (3.85) follows upon summing the two bounds (3.87) and (3.89), and also adding in the initial term: $\Psi(\sigma; x, \lambda)/2 = e^{\lambda^2\sigma^2/2} x^\sigma \ln\zeta(\sigma)/2$. ∎

Theorem 3.29, below, clarifies how the solution for T in $\mathcal{E}_\Sigma(T; x, \lambda, \sigma) = \varepsilon$ varies as a function of x and other parameters. We begin with a lemma giving an asymptotic expansion for the inverse function of $E_1(z)$.

Lemma 3.28 *Given $\delta = E_1(z)$, $z > 0$, the inverse function that gives z as a function of positive real δ is well defined, and*

$$z = -\ln\delta + O(\ln|\ln\delta|) \quad \textit{as } \delta \to 0+.$$

Proof: The inverse function is well defined since $E_1(z)$ is continuous and strictly decreasing from ∞ to 0 as z ranges from 0 to ∞. Using [AS92, Entry 5.1.51] we find that

$$E_1(z) = \frac{e^{-z}}{z}\left(1 + O(z^{-1})\right) \quad \text{as } z \to \infty. \tag{3.90}$$

Setting $\delta = E_1(z)$, treating z as a function of δ, and taking logarithms in (3.90), we find that for δ sufficiently small (and thus z sufficiently large) we have $\ln(\delta) = -z - \ln(z) + O(z^{-1})$. It follows that

$$z = -\ln(\delta) - \ln(z) + O(z^{-1}). \tag{3.91}$$

From [AS92, Entry 5.1.19] we have $e^{-z}/(z+1) < E_1(z) \le e^{-z}/z$. Taking logarithms and writing this in terms of δ gives

$$z + \ln(z) \le -\ln(\delta) < z + \ln(z+1). \tag{3.92}$$

Since $z \le z + \ln(z)$ for $z \ge 1$, the left inequality in (3.92) gives $z \le -\ln(\delta)$, and thus $\ln(z) \ll \ln(-\ln(\delta)) = \ln(|\ln(\delta)|)$ as $\delta \to 0+$. Since $\ln(z+1) \le z$ for $z > 0$, the right inequality in (3.92) gives $2z > -\ln(\delta)$. Thus $z^{-1} \ll 1/|\ln(\delta)|$ as $\delta \to 0+$. Inserting these bounds on $\ln(z)$ and on z^{-1} into (3.91) completes the proof. ∎

In our proof of Theorem 3.29, below, note that the result depends on restricting σ to lie within an interval $[\sigma_{\min}, \sigma_{\max}]$, with $\sigma_{\min} > 1$. We will justify those restrictions in Lemma 3.30 and in Section 3.6, below.

Theorem 3.29 *Given $x > 0$, $0 < \lambda \le 1/2$, $1 < \sigma_{\min} \le \sigma \le \sigma_{\max}$, and $\varepsilon > 0$, the solution for T in $\mathcal{E}_\Sigma(T; x, \lambda, \sigma) = \varepsilon$ satisfies*

$$T = \frac{\sqrt{2}}{\lambda}\left(\sqrt{\ln(x^\sigma/\varepsilon)} + O\left(\frac{\ln\ln(x^\sigma/\varepsilon)}{\ln(x^\sigma/\varepsilon)}\right)\right) \quad \textit{as } x^\sigma/\varepsilon \to \infty \ , \tag{3.93}$$

where the O-constant depends only on $\sigma_{\min}$ and $\sigma_{\max}$.

Proof: Letting $z = \lambda^2 T^2/2$, the condition $\varepsilon = \mathcal{E}_\Sigma(T)$ is equivalent to the condition $E_1(z) = \delta$, where we define δ by

$$1/\delta = \frac{1}{2\pi\varepsilon} e^{\lambda^2\sigma^2/2} x^\sigma \ln\zeta(\sigma). \tag{3.94}$$

Note that $x^\sigma/\varepsilon \to \infty$ suffices to ensure $\delta \to 0+$. Taking logarithms in (3.94), we find that as $x^\sigma/\varepsilon \to \infty$ we have

$$\begin{aligned} -\ln\delta &= \ln(x^\sigma/\varepsilon) - \ln(2\pi) + \ln\ln\zeta(\sigma) + \lambda^2\sigma^2/2 \\ &= \ln(x^\sigma/\varepsilon) + O(1), \qquad (3.95) \\ \ln|\ln\delta| &= O(\ln\ln(x^\sigma/\varepsilon)). \qquad (3.96) \end{aligned}$$

Note that the O-constants in (3.95) and (3.96) depend only on $\sigma_{\min}$ and $\sigma_{\max}$, since λ is bounded.

Now, recalling that $z = \lambda^2 T^2/2$ we apply Lemma 3.28 and use (3.95) and (3.96) to find that

$$\lambda^2 T^2/2 = \ln(x^\sigma/\varepsilon) + O(\ln\ln(x^\sigma/\varepsilon)).$$

Finally, solving for T gives

$$\begin{aligned} T &= \frac{\sqrt{2}}{\lambda}\sqrt{\ln(x^\sigma/\varepsilon) + O(\ln\ln(x^\sigma/\varepsilon))} \\ &= \frac{\sqrt{2}}{\lambda}\left(\sqrt{\ln(x^\sigma/\varepsilon)} + O\left(\frac{\ln\ln(x^\sigma/\varepsilon)}{\ln(x^\sigma/\varepsilon)}\right)\right). \end{aligned} \qquad \blacksquare$$

We now deal with a technical issue that would otherwise complicate the computation of $\Psi(s; x, \lambda)$. More specifically, we treat the computation of $\ln\zeta(s)$, which is a factor of $\Psi(s; x, \lambda)$. Given $\zeta(s)$ the computationally simple way to find $\ln\zeta(s)$ would be to take the principal branch of its logarithm, $\operatorname{Ln}\zeta(s)$, for which $-\pi < \operatorname{Im}\operatorname{Ln}\zeta(s) \le \pi$. Although it could well be that

$\ln\zeta(s) = \operatorname{Ln}\zeta(s)$ whenever $\operatorname{Re}(s) > 1$, neither our definition of $\ln\zeta(s)$ given by Equation (1.16) on page 17, nor the expansion for $\ln\zeta(s)$ given by Equation (1.17), obviously satisfies this condition. Lemma 3.30 gives conditions under which the principal branch *certainly* gives the desired value:

Lemma 3.30 *If* $\sigma \geq \sigma_{\min} := 1.045$ *then* $-\pi < \operatorname{Im}\ln\zeta(\sigma+it) \leq \pi$, *where* $\ln\zeta(\sigma+it)$ *is defined by Equation* (1.16).

Proof: Writing $\ln\zeta(\sigma+it) = \rho + i\theta$, we have $|\theta| \leq \sqrt{\rho^2+\theta^2} = |\ln\zeta(s)|$. Since $\sigma > 1$, we can apply the bound (3.78) to find that $\ln\zeta(s) \leq \ln\zeta(\sigma)$. The result then follows since $\zeta(\sigma)$ is a decreasing function of σ when $\sigma > 1$, and since a numerical computation yields $\ln\zeta(1.045) = 3.126879\cdots < \pi$. ■

Remark For the proof of Lemma 3.30 we computed $\ln\zeta(1.045)$, working with a minimum of 30 digits of precision, using two different methods. In *Mathematica* version 4.2 we used the command

```
N[Log[Zeta[1045/1000]],30]
```

and in PARI/GP version 2.1.4 we used the commands

```
default(realprecision,30); log(zeta(1045/1000))
```

□

3.5 Quadrature algorithm for $\pi^*(x;\lambda)$

In this section we summarize the results of the last few sections by presenting Algorithm 3.3 (`QuadPiStar`) for computing $\pi^*(x;\lambda)$. Roughly speaking, this algorithm implements Equation (3.84) from page 66 to approximate $S(x;\lambda,\sigma,h)$ with an explicitly bounded error term, and applies Equation (3.50) on page 55 to find $\pi^*(x;\lambda)$ from $S(x;\lambda,\sigma,h)$, again with an explicitly bounded error term.

To ensure that $\mathcal{E}_\Sigma(T;x,\lambda,\sigma) \leq \varepsilon$, in line 3 of Algorithm 3.3 we again use our **solve_for** construct. As before, it is clear that the equation of line 3 may be solved using $O(x^\epsilon)$ operations on numbers of $O(\ln(2+|x|))$ bits by root-bracketing and bisection methods, but that in practice one would probably instead use Newton's method, since $\mathcal{E}_\Sigma(T;x,\lambda,\sigma)$ is monotone as a function of T and has a closed form for its derivative.

To avoid spending too much time on error analysis, in Algorithm 3.3 we often use the construct $x := y$ in preference to $x \leftarrow y$. In programming terms, $x := y$ may be thought of as a "macro definition"—in which the right-hand side of the definition specifies the expansion to be used in replacing the left-hand side.

Algorithm 3.3 (`QuadPiStar`: Approximate $\pi^*(x;\lambda)$ by quadrature)
Given $x \geq 2$, $\sigma \geq \sigma_{\min} := 1.045$, $\lambda > 0$, $\varepsilon > 0$, return $\pi^*(x;\lambda) + \varepsilon\, O(1)$. (The requirement that $\sigma \geq \sigma_{\min}$ is a technical condition discussed just before Lemma 3.30 on the preceding page.)

1 **QuadPiStar**$(x, \sigma, \lambda, \varepsilon)$ {

Subdivide allowable error amongst four sources: roundoff error, $I_{\mathcal{L}}$, $I_{\mathcal{R}}$, and $\sum_{kh>T}(\ldots)$.

2 $\varepsilon \leftarrow \varepsilon/4$;

See Equation (3.84) *on page 66 on the choice of T. For the choice of h see Theorem 3.18 on page 57, and Theorem 3.19 on page 58. Recall that in choosing $h_{\mathcal{R}}$ we are setting $K = 0$ in Theorem 3.19. Note that at this point ε is a fraction of its original value.*

3 **solve_for** T **in** $\mathcal{E}_{\Sigma}(T; x, \lambda, \sigma) = \frac{3}{4}\varepsilon + \frac{1}{4}\varepsilon\, O(1)$;

4 $h_{\mathcal{L}} := \dfrac{2\pi(\sigma - 1)}{\ln(x/\varepsilon) + \lambda^2/2 + 1/x}$;

5 $h_{\mathcal{R}} := \dfrac{2\pi}{\ln(x/2) + \sigma\lambda^2 + \lambda\sqrt{2}\sqrt{\sigma \ln(x/2) + \sigma^2\lambda^2/2 + \ln(3.4/\varepsilon)}}$;

6 $h := \min(h_{\mathcal{L}}, h_{\mathcal{R}})$;

7 // *Equation* (3.42) *on page 53 defines $\Psi(s; x, \lambda)$.*

8 **return** $\varepsilon\, O(1) + \dfrac{h}{\pi}\left(\dfrac{1}{2}\Psi(\sigma; x, \lambda) + \displaystyle\sum_{k=1}^{\lceil T/h \rceil} + \operatorname{Re}\Psi(\sigma + ikh; x, \lambda)\right)$; }

Remark We have glossed over a few computational issues in our presentation of Algorithm 3.3. In particular, the computation of $\lceil T/h \rceil$, the uppper bound for the sum of line 8, might require computation to an arbitrarily high accuracy if T/h were an integer. Speaking practically, this is not a difficulty since, instead of $\lceil T/h \rceil$, it would suffice to use any integer that was certainly larger than T/h. □

We now establish a bound on h which will be used below in our complexity analysis of Algorithm 3.3:

Lemma 3.31 *Let x, σ, λ, and ε satisfy the conditions of Algorithm 3.3 and of Theorem 3.29 as stated on page 69. Assume further that $\varepsilon \leq 1/2$. Let h be the value defined in line 6 of Algorithm 3.3. Then there is an $h_{\mathcal{LR}}$ for which $h \leq h_{\mathcal{LR}}$. Subject to the constraints on our parameters, $h_{\mathcal{LR}}$ is an absolute constant.*

Proof: As shown in Section 3.3, as functions of σ, $h_{\mathcal{L}}$ grows in proportion to $\sigma - 1$ while $h_{\mathcal{R}}$ is roughly constant (but slowly decreasing) as σ increases. Thus, there is a unique $\sigma_{\mathcal{LR}}$ for which $h_{\mathcal{L}} = h_{\mathcal{R}}$ at $\sigma = \sigma_{\mathcal{LR}}$.

Let $h_{\mathcal{LR}} := h_{\mathcal{L}}(x, \lambda, \sigma_{\mathcal{LR}}, \varepsilon)$. Since $h = \min(h_{\mathcal{L}}, h_{\mathcal{R}})$, we conclude that $h \leq h_{\mathcal{LR}}$. That $h_{\mathcal{LR}}$ is an absolute constant follows from Equation (3.55), which defines $h_{\mathcal{L}}$, and from our assumptions that $x \geq 2$, $\lambda \leq 1/2$, $\varepsilon \leq 1/2$, and $1 < \sigma_{\min} \leq \sigma \leq \sigma_{\max}$. ■

Definition 3.32, below, introduces our notation for characterizing the cost of computing $\zeta(s)$ in Algorithm 3.3. Note that we characterize our precision requirement for the computation of $\zeta(s)$ in terms of a fixed number of bits of *relative* precision, instead of an absolute precision.

Definition 3.32 Given $n > 0$, let $\varepsilon := 2^{-n}$ and let $\mathcal{K}_\zeta(n)$ denote the number of bit operations required to compute $\big(1 + \varepsilon\, O(1)\big)\zeta(\sigma + it)$, averaged over all values of $\sigma + it$ used in line 8 of Algorithm 3.3. Let $\mathcal{B}_\zeta$ denote the memory requirements for those calculations. □

Remark Note that $\mathcal{K}_\zeta(n)$ and $\mathcal{B}_\zeta$ are implicitly functions of σ, and of other parameters than n, which are used in Algorithm 3.3. □

In our statement of Theorem 3.33, below, we once again assume that $\lambda \gg x^{-1/2}$. Our justification for this assumption is explained just before Theorem 3.13 on page 49.

Theorem 3.33 *Let x, σ, λ, and ε satisfy the conditions of Lemma 3.31. Assume further that $\lambda \gg x^{-1/2}$. Let h be the value defined in line 6 of the algorithm. Then Algorithm 3.3 requires*

$$\ll \frac{\sqrt{\sigma \ln(x)}}{\lambda h}\big(\mathcal{K}_\zeta(\sigma \ln(x)) + \ln(2\sigma \ln(x))\mathcal{M}(\sigma \ln(x))\big) \tag{3.97}$$

bit operations and $\ll x^{\epsilon} + \mathcal{B}_{\zeta}$ *bits of memory. The O-constant implicit in the bound* (3.97) *depends on* ε, *and on the O-constant implicit in our assumption that* $\lambda \gg x^{-1/2}$.

Proof: The bound on the memory requirement of Algorithm 3.33 follows immediately upon noting that all computations other than the computation of values of $\zeta(s)$ can be accomplished using x^{ϵ} bits of memory.

Clearly, the complexity of Algorithm 3.3 is dominated by the complexity of the computation of the main sum in line 8. Applying Theorem 3.29, we find that there are

$$\ll 1 + T/h \ll \frac{\sqrt{\sigma \ln(x)}}{\lambda h} \tag{3.98}$$

terms summed in that line, where we have absorbed a dependency on ε into the O-constant.

To determine the bit complexity of computing each term in the main sum, it suffices to estimate to within an order of magnitude the number of bits of precision needed in each term. Note that although we subdivided ε in line 2, switching between either of these two values for ε in our arguments below will only perturb our results by an O-constant.

Now, assume that we approximate $\Psi(s; x, \lambda)$ to a fixed number of bits of precision in each term of the main sum, which implies that we approximate it to within a fixed relative precision. (See Remark 3.34, below, for further comments on this assumption.)

More precisely, write $(1 + \varepsilon_0 O(1))\Psi(s; x, \lambda)$ for our approximation of $\Psi(s; x, \lambda)$, and write our main sum as

$$U := \frac{1}{2}\Psi(\sigma; x, \lambda) + \sum_{k=1}^{\lceil T/h \rceil} + \operatorname{Re}\Psi(\sigma + ikh; x, \lambda).$$

To compute $\varepsilon\, O(1) + (h/\pi)U$, as we do in line 8, it suffices to choose ε_0 so that $h\varepsilon_0\,|U| \ll \varepsilon$, for some appropriately chosen O-constant. In other words, it suffices to have $\varepsilon_0 \ll \varepsilon/(h|U|)$.

Now, applying Corollary 3.27, we find that

$$h|U| \ll (1 + h + \ln(1/\lambda))e^{\lambda^2\sigma^2/2}x^{\sigma} \ll \ln(x)x^{\sigma}. \tag{3.99}$$

The rightmost side of (3.99), and the fact that the O-constant is absolute, follow from the result of Lemma 3.31 that $h \le h_{\mathcal{LR}} \ll 1$; and because we have assumed that σ lies within a bounded range, that $x \ge 2$, and that $x^{-1/2} \ll \lambda \le 1/2$.

It follows from the rightmost bound of (3.99) that $\varepsilon_0 \ll \varepsilon/(h|U|)$ provided $\varepsilon_0 \ll \varepsilon\, x^{-\sigma}/\ln(x)$. Thus,

$$\ll \ln\left(x^{\sigma}\ln(x)/\varepsilon\right) \ll \sigma\ln(x) \tag{3.100}$$

bits suffice to compute $\Psi(s; x, \lambda)$ to the necessary precision. (The O-constant in the rightmost bound of (3.100) depends on ε.) The computation of $\Psi(s; x, \lambda)$ reduces to the computation of $e^{\lambda^2 s^2/2}x^s/s$ and of $\ln\zeta(s)$, both to $O(\sigma\ln(x))$ bits of precision. A short argument, that depends on our assumption that $1 < \sigma_{\min} \le \sigma \le \sigma_{\max}$, shows that the computation of $\ln\zeta(s)$ reduces to the computation of $\zeta(s)$ to $O(\sigma\ln(x))$ bits of precision. The result (3.97) follows upon summing the complexity bounds for the computation of $e^{\lambda^2 s^2/2}x^s/s$ and of $\zeta(s)$, and then taking the product of the resulting bound on bit complexity with the bound (3.98) on the number of terms computed. ■

Remark 3.34 In our analysis of Algorithm 3.3 we assumed that $\Psi(s; x, \lambda)$, and thus $\zeta(s)$, were both approximated to within a fixed relative precision.

This is not the optimal choice for precision, since Algorithm 3.3 would be more efficient if we approximated $\Psi(s; x, \lambda)$ to within a fixed *absolute* precision. This would imply that the number of bits required in our computation of $\Psi(s; x, \lambda)$ would grow smaller as $\Psi(s; x, \lambda)$ grows smaller. Although it might lower our complexity bound for Algorithm 3.3, we have not treated the possibility of working to within a fixed absolute precision here, since it would complicate our complexity analysis. □

Remark 3.35 Several changes would be required in order to have Algorithm 3.3 apply the more sophisticated quadrature method implied by Equation (3.53) on page 56. We would further subdivide ε to account for one further source of error. We would not compute $h_{\mathcal{R}}$, and instead set $h \leftarrow h_{\mathcal{L}}$. We would then choose K as suggested by Equation (3.73), and set $R := e^{2\pi/h}$. Instead of returning the sum computed in line 8 we would instead assign the result to `pistar`, say, and then compute the value to be returned by adding in our "quadrature correction terms" as follows:

$$\texttt{pistar} \leftarrow \texttt{pistar} - \sum_{k=1}^{K} R^{k\sigma}\big(\pi(R^{-k}x) - \texttt{Delta}(R^{-k}x, \lambda, R^{-k\sigma}\varepsilon)\big)\,;$$

Note that in this sum we have $R^{-k}x$ replacing the role of x, and $R^{-k\sigma}\varepsilon$ replacing the role of ε as arguments passed to $\texttt{Delta}(\dots)$. These values invalidate several assumptions that we made about the arguments passed to $\texttt{Delta}(x, \lambda, \varepsilon)$ (Algorithm 3.2) in our analysis given in Theorem 3.13 on page 49, and that analysis would need to be extended to cover a more diverse range of values. □

3.6 Choosing the parameters σ and λ

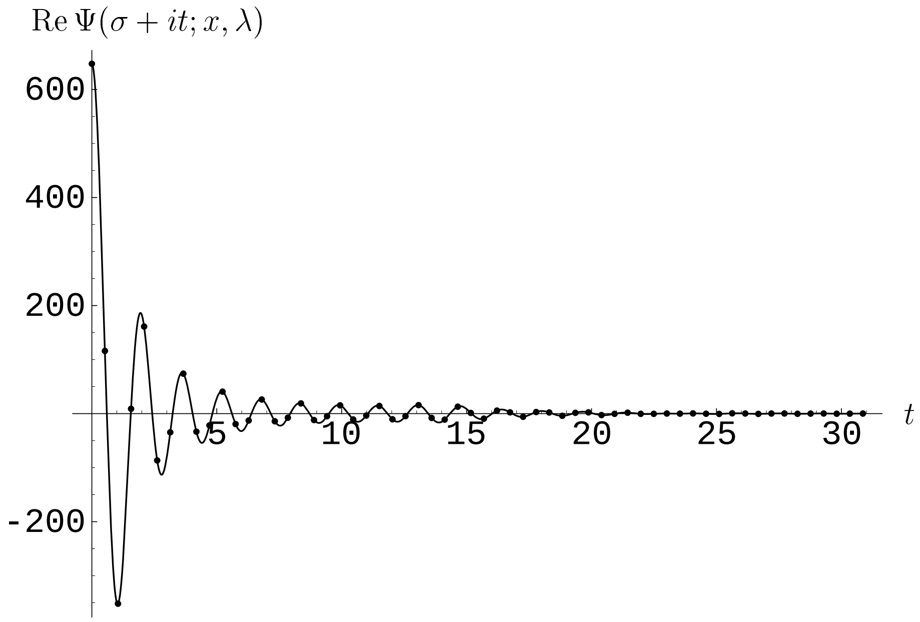
Recall that in this dissertation we only consider paths of integration for the analytic algorithm of the form $s = \sigma + it$, parameterized by t, $-\infty < t < \infty$, with fixed $\sigma > 1$. There may be advantages to choosing more general paths, in particular, paths that pass to the left of $\sigma = 1$, but the apparent complexity of analyzing such paths has prevented their consideration here.

We now consider the choice of σ that minimizes the running time of Algorithm 3.3. We give only a rough analysis, which has not been tested in an implementation. In practice, σ would best be chosen based on experience with an implementation—since the running time will be sensitive to details of the machine arithmetic and of the software used to implement multiprecision arithmetic.

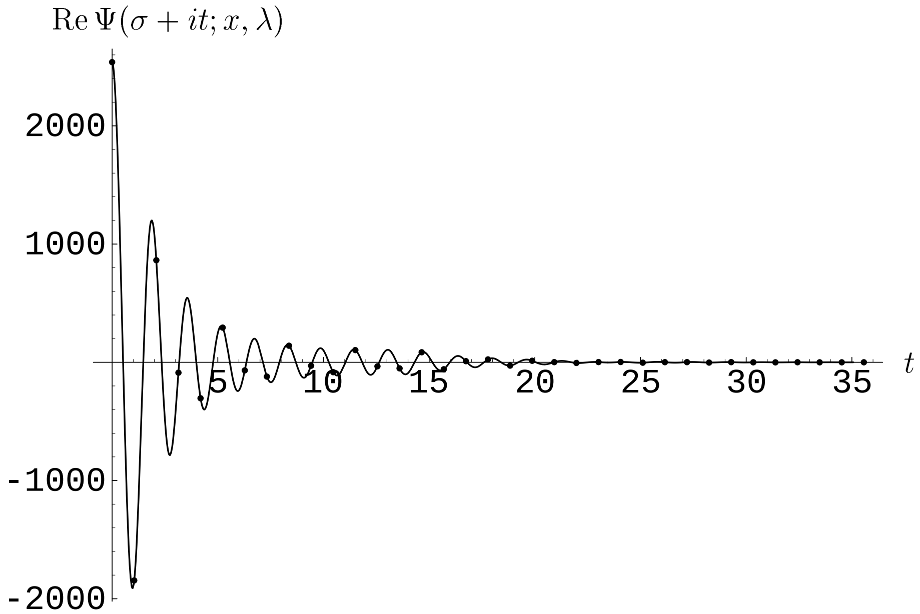
In our analysis, we need only consider how the running time of Algorithm 3.3 depends on σ. Figure 3.4, on the next page, illustrates how the choice of σ affects the quantities that determine the running time. These quantities are the magnitude of the integrand, which is the main factor determining the precision required for our arithmetic; and the number of quadrature points used, which depends on T and h.

As shown in the proof of Theorem 3.33, the required number of bits of accuracy grows in proportion to σ. By Theorem 3.29, T grows roughly in proportion to $\sqrt{\sigma}$.

From the proof of Lemma 3.31, we know that there is a $\sigma_{\mathcal{LR}}$ for which h as a function of σ reaches its maximum value at $\sigma = \sigma_{\mathcal{LR}}$, and that h grows in proportion to $\sigma - 1$ for $\sigma \le \sigma_{\mathcal{LR}}$, while h is roughly constant (but slowly



(a) $\sigma = 1.5$, $h \approx 0.52$, $T \approx 30.72$, 60 sample points.



(b) $\sigma = 2.0$, $h \approx 1.05$, $T \approx 35.07$, 35 sample points.

Figure 3.4: Integrand and quadrature sample points for two choices of σ. In both cases we have $x = 100$, $\lambda = 0.1$, $\varepsilon = 1/4$, we let $h = h_{\mathcal{L}}(x, \lambda, \sigma, \varepsilon)$ and choose T to solve $\mathcal{E}_\Sigma(T; x, \lambda, \sigma) = \varepsilon$.

decreasing in σ) for $\sigma \geq \sigma_{\mathcal{LR}}$.

The analysis above leads us to conclude that the number of quadrature points grows roughly in proportion to $\sqrt{\sigma}/(\sigma - 1)$ for $\sigma \leq \sigma_{\mathcal{LR}}$ and in proportion to $\sqrt{\sigma}$ for $\sigma \geq \sigma_{\mathcal{LR}}$. We note that the same analysis that led to the estimate (3.74) on page 63 shows that, as a rule of thumb,

$$\sigma_{\mathcal{LR}} \approx 1 + \frac{\ln(x/\varepsilon)}{\ln(x/2)} \approx 2. \tag{3.101}$$

On the assumption that it exists and is well defined, let σ_{opt} denote the optimal choice for σ. Clearly we must have $\sigma_{\text{opt}} \leq \sigma_{\mathcal{LR}}$. This argument helps justify the assumption of Theorem 3.29 that $\sigma \leq \sigma_{\max}$ for some $\sigma_{\max}$, since it would be inefficient to choose $\sigma > \sigma_{\text{opt}}$ and we can reasonably let $\sigma_{\max} := \sigma_{\text{opt}} \leq \sigma_{\mathcal{LR}}$.

To continue with our analysis we let $\mathcal{K}(\sigma)$ denote the running time of Algorithm 3.3 as a function of σ. Let $f(\sigma) \propto g(\sigma)$ denote the property that $f(\sigma)/g(\sigma)$ is constant as a function of σ, i.e., that $f(\sigma)$ grows in proportion to $g(\sigma)$. Working from the bound (3.97) on page 72, and dropping all factors which change slowly with σ, we use a rough and simple model in which we assume that

$$\mathcal{K}(\sigma) \propto \frac{\sigma^{\mu_1+\mu_2+1/2}}{\sigma - 1} \propto \frac{\sigma^{\mu+1/2}}{\sigma - 1} \tag{3.102}$$

where the meaning of μ_1 and μ_2 are explained below, and where, of course, $\mu := \mu_1 + \mu_2$. Assuming that $\mu > 1/2$, we can easily show that $\mathcal{K}(\sigma)$ is minimized at

$$\sigma = \sigma_0 := \frac{\mu + 1/2}{\mu - 1/2}. \tag{3.103}$$

The model for $\mathcal{K}(\sigma)$ in Equation (3.102) is based on three assumptions: First, that σ is restricted to the range $\sigma_{\min} \leq \sigma \leq \sigma_{\mathcal{LR}}$, so that $h = h_{\mathcal{L}}$ in that range. Second, that for some choice of μ_1, we have $\mathcal{M}(\sigma \ln(x)) \propto \sigma^{\mu_1}$. Third, that on average in Algorithm 3.3, and for some choice of μ_2, the computation of $\Psi(s; x, \lambda)$ to an accuracy of $O(\sigma \ln(x))$ bits requires $\propto \sigma^{\mu_2}$ arithmetic operations on numbers of $O(\sigma \ln(x))$ bits.

Our first assumption will be violated if Equation (3.103) gives $\sigma_0 > \sigma_{\mathcal{LR}}$, but that simply means that we should set $\sigma_{\text{opt}} = \min(\sigma_0, \sigma_{\mathcal{LR}})$.

Our second and third assumption are plausible if we ignore factors of σ^ϵ and assume that $\sigma \ln(x)$ is so large that theoretical complexity estimates apply, and for the third assumption, perhaps only for σ in a limited range.

Considering the choice of μ_1, we should have $1 \le \mu_1 \le 2$, depending on the algorithm used for multiplication. The lower limit of $\mu_1 = 1$ applies for an FFT-based algorithm [Knu81, §4.3.3], while the uppper limit of $\mu_1 = 2$ applies for the classical multiplication algorithm.

Similar remarks apply to μ_2. Recall from Definition 3.32 on page 72 that $\mathcal{K}_\zeta(n)$ denotes the bit complexity of computing $\zeta(s)$ to n-bit accuracy, and that $\mathcal{K}_\zeta(n)$ is implicitly a function of σ. Since μ_1 already accounts for the cost of basic operations on numbers of n bits, and since the dominant complexity in computing $\Psi(s; x, \lambda)$ will be the complexity of computing $\zeta(s)$, we see that μ_2 will be determined by the behavior of $\mathcal{K}_\zeta(\sigma \ln(x))/\mathcal{M}(\sigma \ln(x))$ as a function of σ.

Assuming that we use the method of Chapter 7 to compute $\zeta(s)$, the number of operations required to approximate $\zeta(s)$ is determined by two parameters: N and M, which bound the number of terms in the first two sums of Formula (7.9) on page 143. There are two other sums in Formula (7.9), but our analysis in Section 7.6, starting on page 149, indicates that the first two have the dominant complexity.

The first sum of Formula (7.9) is the truncated Dirichlet series: $\sum_{n=1}^{N} n^{-s}$. Writing $s = \sigma + it$, our analysis in Section 7.6 shows that $N \asymp t^{1/2}$, with little dependence on σ provided t is large compared to σ. The analysis also suggests that M is much smaller than N unless σ is quite large. If we compute the truncated Dirichlet series in a straightforward way, it follows that we can assume $\mu_2 = 0$.

On the other hand, if we use the Odlyzko-Schönhage algorithm to compute the truncated Dirichlet series, the complexity of computing the series is relatively insensitive to N and should grow roughly in proportion to σ. Furthermore, our analysis indicates that, roughly, $M \asymp \sigma$ provided σ is not too large, and $M \asymp \sigma^{3/2}$ for larger values of σ. (See the discussion immediately below Equation (7.22) on page 149, and the discussion which follows—which describes of a choice of parameters for Formula (7.9) which are suitable for very precise computation of $\zeta(s)$.) It follows that, when using the Odlyzko-Schönhage algorithm, we can assume $1 \le \mu_2 \le 3/2$.

To summarize, depending on our multiplication algorithm and our algorithm for computing $\zeta(s)$, we will have $\mu = \mu_1 + \mu_2$ in the range $1 \le \mu \le 7/2$. From Equation (3.103), we conclude that σ_0 a decreasing function of μ in the range $3 \ge \sigma_0 \ge 4/3$. Since our rule of thumb given by Equation (3.101) suggests that $\sigma_{\mathcal{LR}} \approx 2$, we expect that σ_{opt} will lie in the range $2 \ge \sigma_{\mathrm{opt}} \ge 4/3$, depending on our choice of algorithms. If we assume that $\mu_1 = 1$ (fast multiplication) and that $\mu_2 = 1$ (best case when using the Odlyzko-Schönhage algorithm) then we find that, by this model, $\sigma_{\mathrm{opt}} = 5/3 \approx 1.67$.

We now turn to the question of choosing λ, our *length parameter*. As in the case of choosing σ, in practice λ would best be chosen based on experience with an implementation. Here we use a simple model where we assume that the complexity of Algorithm 3.2 (`Delta`) is characterized by the function $\mathcal{K}_1(x, \lambda)$ and the complexity of Algorithm 3.3 (`QuadPiStar`) is given by $\mathcal{K}_2(x, \lambda)$. Of course $\mathcal{K}_1$ and $\mathcal{K}_2$ depend on other parameters such as ε and σ, but we assume that those parameters are fixed.

Although Theorem 3.13 gives an upper bound on $\mathcal{K}_1(x, \lambda)$ and Theorem 3.33 gives an upper bound on $\mathcal{K}_2(x, \lambda)$, we assume that the true costs have the same form as these upper bounds. Also, recall that we have assumed that $x^{-1/2} \ll \lambda \le 1/2$ in our analyses above. With this understanding, we find that

$$\mathcal{K}_1(x, \lambda) = \kappa_1(x)\sqrt{\ln(\lambda x)}\lambda x, \tag{3.104}$$

$$\mathcal{K}_2(x, \lambda) = \kappa_2(x)/\lambda, \tag{3.105}$$

where $\kappa_1(x)$ depends on our choice of method for enumerating primes, and $\kappa_2(x)$ depends on our choice of method for computing $\zeta(s)$.

Rather than solving for λ that minimizes $\mathcal{K}_1(x, \lambda) + \mathcal{K}_2(x, \lambda)$ as a function of x, we simply solve for λ to make $\mathcal{K}_1(x, \lambda) = \mathcal{K}_2(x, \lambda)$ as a function of x, since the latter solution serves nearly the same purpose. This gives

$$\lambda = x^{-1/2}\sqrt{\frac{\kappa_2(x)}{\kappa_1(x)\sqrt{\ln(\lambda x)}}}, \tag{3.106}$$

and our restriction that $x^{-1/2} \ll \lambda \le 1/2$ implies that

$$\lambda \asymp x^{-1/2}\sqrt{\frac{\kappa_2(x)}{\kappa_1(x)\sqrt{\ln(x)}}}. \tag{3.107}$$

Substituting the solution (3.107) back into Equations (3.104) and (3.105), we find that for this choice of λ we have

$$\mathcal{K}_1(x,\lambda) \asymp \mathcal{K}_2(x,\lambda) \asymp \ln^{1/4}(x)\sqrt{\kappa_1(x)\kappa_2(x)x}.$$

4 Survey of Methods for Enumerating Primes

> **e·nu·mer·ate** *v.* **1.** To name one by one; list. **2.** To determine the number of; count. [Lat. *ēnumerāre*, count out.]
>
> *The American Heritage Dictionary, 3rd Ed.*

4.1 Preliminaries

To implement Algorithm 3.2 (`Delta`) we need an efficient method for enumerating primes, where by *enumerate* we mean the first sense of the dictionary definition quoted above. In this chapter we introduce terminology and summarize some of the issues involved in enumerating primes, while the following two chapters give new efficient methods for achieving this goal.

Throughout this chapter, and the following two chapters, we will analyze the task of enumerating primes in an interval $[x_1, x_2]$, $0 < x_1 \leq x_2$. In the terminology of Section 1.5, all our estimates of computational complexity will be given in terms of the number of arithmetic operations performed on numbers of $O(\ln(2 + x_2))$ bits. Following the convention of Definition 1.2 on page 13, the unqualified term *operation* refers to such an arithmetic operation.

4.2 Enumeration by sieving

The sieve of Eratosthenes, or one of its many variants developed over the last few decades, has been the method of choice for enumerating primes in an interval $[x_1, x_2]$, provided the interval is sufficiently long. However, as we will explain below, sieves previously described in the literature become inefficient unless $x_2 - x_1 \geq x_2^{1/2+o(1)}$ as $x_2 \to \infty$. For these methods, this implies a lower bound of $x_2^{1/2+o(1)}$ bits of memory required for efficient sieving. A survey of many sieving algorithms and their memory requirements is given in a recent paper by Sorenson [Sor98].

In this section we give an overview of sieving algorithms, and then illustrate this overview with Algorithm 4.1 (`DemoSieve`) on page 85. We begin by introducing some convenient notation:

Definition 4.1 We define the *size* of an interval $[x_1, x_2]$ to be the number of integers in that interval, i.e., $1 + \lfloor x_2 \rfloor - \lceil x_1 \rceil$.

Given a sieving algorithm, and an interval $[x_1, x_2]$, let $\mathcal{K}$ denote the number of arithmetic operations required to sieve the interval and let B denote the size of the interval. We define the *cost per unit subinterval* for the sieve to be $\mathcal{K}/B$. □

Remarks We will often assume that $x_1, x_2 \in \mathbb{Z}$, in which case the size of the interval $[x_1, x_2]$ is $1 + x_2 - x_1$. The distinction between the length of an interval (i.e., $x_2 - x_1$) and the size of an interval becomes important for very short intervals, especially in the case $x_1 = x_2$. □

All of the sieves that we consider will use a "bit vector" to represent a set $\mathcal{S}$ of numbers within an interval $[x_1, x_2]$. Usually we will be interested in $\mathcal{S} = \mathcal{P} \cap [x_1, x_2]$, but in Chapter 6 we will also consider the set of numbers in $[x_1, x_2]$ which are free of "small" prime divisors.

In imitation of C-notation, writing `S` for the bit vector representing the set $\mathcal{S}$, we associate three quantities with `S`: the endpoints of the interval, which are denoted by `S.x1` and `S.x2`; and the actual vector of bits, which is denoted as `S.data`. We represent the binary values in `S.data` as $\mathtt{S}[n] \in \{0, 1\}$, where $n \in \mathcal{S} \Leftrightarrow \mathtt{S}[n] = 1$ and where $\mathtt{S.x1} \leq n \leq \mathtt{S.x2}$.

In other words, the elements of $\mathcal{S}$ can be enumerated by testing whether $\mathtt{S}[n] = 1$ as n runs through the values $\mathtt{S.x1} \leq n \leq \mathtt{S.x2}$, although we will normally consider the elements to be enumerated once the computation of `S` is completed. For convenience, we require $\mathtt{S.x1} \in \mathbb{N}$, $\mathtt{S.x2} \in \mathbb{N}$. By the "size" of the bit vector `S` we mean `BitSizeOf(S.data)`. The expression $\mathtt{S} \leftarrow \mathtt{AllocateBitVector}(B)$ denotes the creation of (storage allocation for) a bit vector `S` of size B.

After creating the bit vector, we then assign values to `S.x1` and `S.x2` to specify the endpoints of the interval being treated—under the restriction that $\mathtt{BitSizeOf(S.data)} \geq 1 + \mathtt{S.x2} - \mathtt{S.x1}$. This convention lets us reuse the space allocated for `S.data` when we sieve over consecutive intervals of size bounded by B. Note that we allow for the case that `S.data` is larger than strictly necessary. Also note that since it takes $O(\ln(\texttt{S.x2}))$ bits to represent both `S.x1` and `S.x2`, a bit vector `S` of size B requires a total of $B + O(\ln(\texttt{S.x2}))$ bits of storage, i.e.,

$$\texttt{BitSizeOf(S)} = \texttt{BitSizeOf(S.data)} + O(\ln(\texttt{S.x2})).$$

Of course, a practical implementation of a sieve would not allocate storage for, nor waste computation on, the even numbers in an interval. More generally, it is possible to exclude from consideration all numbers divisible by primes below some bound. By letting the bound grow with x_2, as in Pritchard's "wheel sieve" [Pri81, Pri82], it is possible to sieve with a cost of $O(1/\ln\ln(x_2))$ arithmetic operations per unit subinterval provided the interval $[x_1, x_2]$ is long enough. However, attempting to apply these techniques would complicate our exposition and analysis, and except for a few comments in Section 5.7 we will not explore them further in this dissertation.

When discussing any sieving algorithm, we will usually be concerned with a family of closely related algorithms for sieving an interval $[x_1, x_2]$. By convention, the *unsegmented* version of a sieve will work with a preallocated bit vector of size $1 + \lfloor x_2 \rfloor - \lceil x_1 \rceil$, i.e., a bit vector of sufficient size to represent the entire interval $[x_1, x_2]$. We also consider the *segmented* version of the sieve. This version, in order to save storage, breaks $[x_1, x_2]$ into subintervals whose disjoint union is $[x_1, x_2]$—where each subinterval is as large as possible subject to having a size bounded by B. The segmented sieve enumerates the primes in each subinterval using the unsegmented version, working with a bit vector of size B. (The idea of a segmented sieve was mentioned in print as early as 1973 by Richard Brent [Bre73], although the idea is often attributed to a later paper by Bays and Hudson [BH77].)

We can characterize the operation count for the unsegmented version of a sieve by a bound of the form $O\big(\alpha(x_2)(1 + \lfloor x_2 \rfloor - \lceil x_1 \rceil) + \beta(x_2)\big)$. Here, $\alpha(x_2)$ measures the sieve's cost per unit subinterval, while $\beta(x_2)$ measures a fixed overhead independent of the size. When, as is often the case, we have $\beta(x_2) \gg \alpha(x_2)$ then the bound on the operation count can be written more simply as $O\big(\alpha(x_2)(x_2 - x_1) + \beta(x_2)\big)$.

In practice, $\beta(x_2)$ increases as x_2 increases, and a sieve becomes inefficient if the interval $[x_1, x_2]$ fails to grow longer as x_2 increases. More precisely, a sieve becomes inefficient when the interval is so short that the sieve's overhead dominates its cost per unit subinterval.

In terms of $\alpha(x_2)$, $\beta(x_2)$, we see that the cost of a sieve in arithmetic operations per unit subinterval is $\ll \alpha(x_2) + \beta(x_2)/(1 + \lfloor x_2 \rfloor - \lceil x_1 \rceil)$. Thus, provided $x_2 - x_1 \gg \beta(x_2)/\alpha(x_2)$ a sieve requires $O(\alpha(x_2))$ operations per unit subinterval. Although we have been discussing upper bounds, in practice the operation count is also $\gg \alpha(x_2)(x_2 - x_1) + \beta(x_2)$, so a sieve becomes inefficient if the size of $[x_1, x_2]$ grows more slowly than $\beta(x_2)/\alpha(x_2)$, i.e., if $(x_2 - x_1)\alpha(x_2)/\beta(x_2) \to 0$ as $x_2 \to \infty$.

Suppose $\alpha(x_2)$, $\beta(x_2)$ characterize the operation count for the unsegmented version of a sieve, $\beta(x_2) \gg \alpha(x_2)$, and that $\alpha(x_2)$ is nondecreasing. The unsegmented sieve will require $O(\alpha(x_2)B + \beta(x_2))$ operations to sieve a subinterval of $[x_1, x_2]$, where B bounds the size of the subinterval. (We allow B to vary with x_2.) The segmented version will require an additional $O(1)$ operations per subinterval to partition $[x_1, x_2]$ and to invoke the unsegmented version on each subinterval, but this $O(1)$ cost will be dominated by $\beta(x_2)$.

Thus, summing over subintervals, we find that the segmented version of the sieve requires

$$\begin{aligned} &\ll \left\lfloor \frac{1 + \lfloor x_2 \rfloor - \lceil x_1 \rceil}{B} \right\rfloor (\alpha(x_2)B + \beta(x_2)) \\ &\qquad + \alpha(x_2)\min(B, 1 + \lfloor x_2 \rfloor - \lceil x_1 \rceil) + \beta(x_2) \\ &\ll (\alpha(x_2) + \beta(x_2)/B)(x_2 - x_1) + \beta(x_2) \qquad (4.1) \end{aligned}$$

operations to sieve the interval $[x_1, x_2]$. In other words, the segmented sieve requires

$$\ll \alpha(x_2) + \beta(x_2)\left(\frac{1}{B} + \frac{1}{1 + \lfloor x_2 \rfloor - \lceil x_1 \rceil}\right) \qquad (4.2)$$

operations per unit subinterval, which works out to $O(\alpha(x_2))$ operations per unit subinterval provided both $B \gg \beta(x_2)/\alpha(x_2)$ and $x_2 - x_1 \gg \beta(x_2)/\alpha(x_2)$.

This analysis shows that the segmented version becomes inefficient when either $B\,\alpha(x_2)/\beta(x_2) \to 0$ or $(x_2 - x_1)\alpha(x_2)/\beta(x_2) \to 0$ as $x_2 \to \infty$. Similar remarks apply to the "wheel sieve", even though $\alpha(x_2) = 1/\ln\ln(x_2)$ *decreases* slowly as x_2 increases.

We illustrate our discussion above by analyzing the sieve of Eratosthenes. The unsegmented version is given by Algorithm 4.1 (`DemoSieve`), below.

Algorithm 4.1 (`DemoSieve`: **Sieve of Eratosthenes**)
Given a preallocated bit vector `Pset` *with* `Pset.x1` $= x_1$, `Pset.x2` $= x_2$, $1 \le x_1 \le x_2$, *this algorithm sets* `Pset`$[n]$ *such that upon completion we have* `Pset`$[n] = 1$ *if and only if* n *is prime.*

`DemoSieve(Pset)` $\{$
 $x_1 \leftarrow$ `Pset.x1`; $x_2 \leftarrow$ `Pset.x2`;
 assert $x_1 \in \mathbb{N} \wedge x_2 \in \mathbb{N}$; **assert** $1 \le x_1 \le x_2$;
 // *Initialize* `Pset` *to have all* 1 *entries.*
 for $(n \leftarrow x_1; n \le x_2; n{+}{+})$ `Pset`$[n] \leftarrow 1$;
 // *Now zero out (cross out) entries at nontrivial multiples of primes* $\ell \le \sqrt{x_2}$.
 for $(\ell \in [2, \sqrt{x_2}] \cap \mathcal{P})$
  for $(m \leftarrow \max(2, \lceil x_1/\ell \rceil); m \le x_2/\ell; m{+}{+})$
   `Pset`$[m\ell] \leftarrow 0$;
$\}$

To simplify our analysis, we will assume that the primes ℓ in Algorithm 4.1 have been precomputed. Letting $x_1 =$ `Pset.x1`, $x_2 =$ `Pset.x2`, we see that the initialization phase takes $O(1 + x_2 - x_1)$ operations. This is dominated by the number of operations needed to cross out composites, which is

$$\begin{aligned} &\ll \sum_{\ell \le \sqrt{x_2}} \left(\frac{1 + x_2 - x_1}{\ell} + 1 \right) \ll (1 + x_2 - x_1) \sum_{\ell \le \sqrt{x_2}} \frac{1}{\ell} + \pi(\sqrt{x_2}) \\ &\ll \ln\ln(x_2)(x_2 - x_1) + \sqrt{x_2}/\ln(x_2). \end{aligned} \tag{4.3}$$

From (4.3) we see that for Algorithm 4.1 we have $\alpha(x_2) = \ln\ln(x_2)$ and $\beta(x_2) = \sqrt{x_2}/\ln(x_2)$. (In a more realistic implementation of the sieve of Eratosthenes, where we would also need to sieve to enumerate the primes $\ell \le \sqrt{x_2}$, we would have $\beta(x_2) = \ln\ln(x_2)\sqrt{x_2}$.)

Algorithm 4.2 (`SegmentedSieve`), below, gives the segmented version of Algorithm 4.1.

Algorithm 4.2 (`SegmentedSieve`: **Segmented sieve of Eratosthenes**)
Enumerate the primes in the interval $[x_1, x_2]$, $1 \le x_1 \le x_2$.

`SegmentedSieve`$(B \in \mathbb{N}, x_1, x_2)$ $\{$
 `Pset` $\leftarrow$ `AllocateBitVector`(B);
 for (`Pset.x1` $\leftarrow \lceil x_1 \rceil$; `Pset.x1` $\le x_2$; `Pset.x1` $\leftarrow$ `Pset.x2` $+ 1$) $\{$
  `Pset.x2` $\leftarrow \min(x_2,$ `Pset.x1` $+ B - 1)$;

```
  DemoSieve(Pset);
  for (n ← Pset.x1; n ≤ Pset.x2; n++)
    if (Pset[n] = 1)      Enumerate(n);  // Enumerate n as a prime
}}
```

In Algorithm 4.2, the construct `Enumerate`(n) denotes the act of passing n to another computation or activity. We assume that the activity denoted by `Enumerate`(n) requires $O(1)$ operations, so the bound (4.1) implies that Algorithm 4.2 requires

$$\left(\ln\ln(x_2) + \frac{\sqrt{x_2}}{B\ln(x_2)}\right)(x_2 - x_1) + \frac{\sqrt{x_2}}{\ln(x_2)}$$

operations. Provided that $x_2 - x_1 \gg \sqrt{x_2}/(\ln\ln(x_2)\ln(x_2))$ and also that $B \gg \sqrt{x_2}/(\ln\ln(x_2)\ln(x_2))$, the bound (4.2) implies that Algorithm 4.2 requires $O(\ln\ln(x_2))$ operations per unit subinterval. On the other hand, when $B = 1$ then Algorithm 4.2 is, in effect, using trial division to test for primality, and the bound (4.2) implies that the algorithm then requires $O(\sqrt{x_2}/\ln(x_2))$ operations per unit subinterval.

In the language of this section, all sieves previously described in the literature have $\alpha(x_2) = x_2^{o(1)}$ while $\beta(x_2) = x_2^{1/2+o(1)}$ as $x_2 \to \infty$. Thus, for these sieves we also have $\beta(x_2)/\alpha(x_2) = x_2^{1/2+o(1)}$. Again, the bound (4.2) implies that the segmented versions of these sieves require only $O(x_2^{o(1)})$ operations per unit subinterval to sieve $[x_1, x_2]$, provided both $x_2 - x_1 \gg \beta(x_2)/\alpha(x_2)$ and $B \gg \beta(x_2)/\alpha(x_2)$. However, we see that these sieves become much slower if B grows more slowly than $\beta(x_2)/\alpha(x_2)$ as x_2 increases.

For the analytic algorithm we are interested in $x_2 \geq 4 \cdot 10^{22}$—which corresponds to the current "record computation" of $\pi(x)$. The analysis above shows we would need on the rough order of $B = 10^{11}$ bits of memory for one of the classical sieves to operate efficiently near $x_2 = 4 \cdot 10^{22}$, but this amount of fast computer memory is unavailable on contemporary machines.

In Chapter 5 we develop a new sieve—the "dissected sieve"—with much lower overhead, for which $\beta(x_2) = x_2^{1/3}$ while $\alpha(x_2) = 1$. Thus, the segmented version of this sieve operates efficiently provided both $x_2 - x_1 \gg x_2^{1/3}$ and $B \gg x_2^{1/3}$.

4.3 Enumeration by primality testing

In order to bound the memory requirements for the analytic algorithm to $O(x^{1/4+o(1)})$ bits, Lagarias and Odlyzko [LO87] proposed to enumerate primes without sieving by using the "APR" algorithm of Adleman, Pomerance and Rumely [APR83].

The APR algorithm was the first to be developed of several *primality tests* which test the primality of a single n using $O(n^\epsilon)$ bits and $O(n^\epsilon)$ arithmetic operations. All of these primality tests can be thought of as generalizations of *probable primality* tests such as the test that $2^{n-1} \equiv 1 \pmod{n}$. This particular probable primality test requires $O(\ln n)$ arithmetic operations on numbers of $O(\ln n)$ bits, and is satisfied by all odd prime n and by a relatively sparse set of composite n.

Primality tests differ from *probable* primality tests in that the former determine the primality of n with complete certainty, although at greater computational cost. The APR algorithm has since been refined to give the "APRCL" test [CL84]. Both tests use properties of cyclotomic number fields. They are probabilistic algorithms which require $O(n^\epsilon)$ bits and (at least if we assume the Extended Riemann Hypothesis) $O((\ln n)^{C \ln\ln\ln n})$ arithmetic operations to determine the primality of n, where $C > 0$ is a constant.

In contrast, the "ECPP" primality test [AM93, Mor98] uses properties of elliptic curves. This probabilistic algorithm determines the primality of n using $O(n^\epsilon)$ bits and a number of operations which is polynomial in $\ln(n)$, with an expected running time of $O(\ln^6 n)$. (However, see the discussion of Figure 4.1 on the following page.) Although the ECPP test requires less time than the APRCL test as $n \to \infty$, current refinements of the APRCL test require less time for n of reasonable magnitude (say $\ln(n) < 10^9$, as claimed in [Mih98, §6]).

A third kind of primality test is that of Agrawal, Kayal and Saxena [AKS02, Bor03], which is a *deterministic* algorithm which runs in time which is polynomial in $\ln(n)$. However, in its current form the algorithm is of only theoretical interest, since, in practice, its running time exceeds that of the APRCL and ECPP tests.

To get a sense of the speed of a primality testing algorithm, we can refer to Figure 4.1 on the next page, which presents timing data for an implementation of the ECPP primality test, Version 6.4.5, available at `http:`

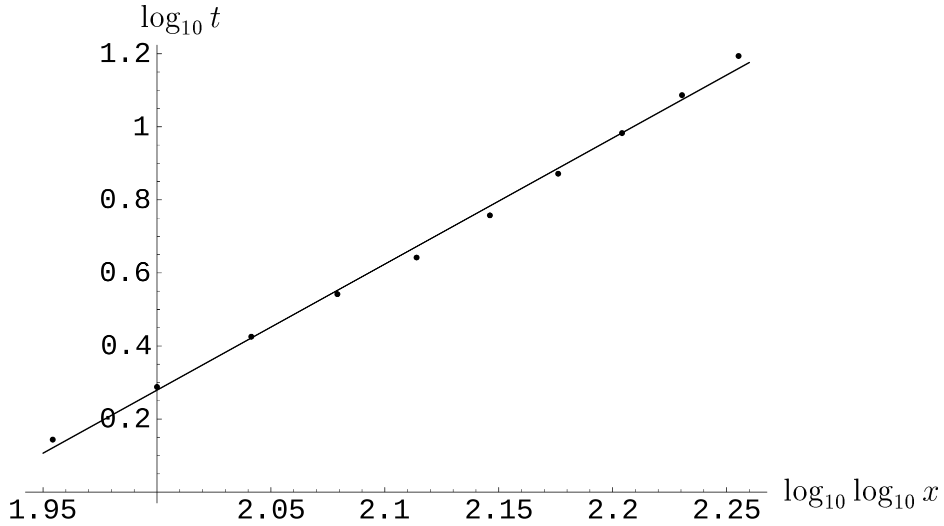


Figure 4.1: Average time, t, in seconds, required by ECPP to test primality of the 500 primes following x, with $x = 10^{90}, 10^{100}, \ldots, 10^{180}$. A linear fit of $\log_{10}(t)$ to $\log_{10}\log_{10}(x)$ at these sample points gives $t \approx 2.4\cdot 10^{-7}(\log_{10} n)^{3.45}$.

`//www.lix.polytechnique.fr/~morain/Prgms/ecpp.english.html`. The timing data was collected on a 500 MHz SUN SPARCv9 processor. ECPP uses trial division and probable primality testing to detect composite numbers relatively quickly. For this reason we restricted our test data to prime numbers. Because this implementation of ECPP is designed to work slowly with "small" numbers (e.g., numbers on the order of 10^{24}) we collected data for primes in the range $10^{90} \ldots 10^{180}$.

Note that $\log_{10}(t)$ is well approximated by a linear expression of the form $\log_{10}(t) \approx \kappa_0 + \kappa_1 \log_{10}\log_{10}(x)$. Recall that $\mathcal{M}(b)$ denotes the number of bit-operations required to multiply two numbers of b bits. Without knowing the specific multiplication algorithm used by ECPP, we can reasonably assume that $b \ll \mathcal{M}(b) \ll b^2$. These bounds on $\mathcal{M}(b)$, and the complexity bound of $O(\ln^6(n))$ arithmetic operations for ECPP, lead us to expect $7 \le \kappa_1 \le 8$. Since $\kappa_1 \approx 3.45$ in our fitted data it appears that the actual complexity of ECPP is significantly less than the upper bound of $O(\ln^6(n))$ operations—at least for the range of numbers used in our benchmark.

Extrapolating the data presented in Figure 4.1 suggests that it would take roughly 0.0138 seconds/prime (72.5 primes/second) for ECPP to test primality near $x = 10^{24}$. In contrast, on the same processor it takes roughly $2.06 \cdot 10^{-4}$ seconds (4863 tests/second) to perform a probable primality test on a number near 10^{24}. (Our implementation of the probable primality test used the GNU Multiple Precision Arithmetic Library (GMP), see Section 6.4 for further information on that library.)

By applying a primality test to each $n \in [x_1, x_2]$ we may enumerate primes in arbitrarily short intervals $[x_1, x_2]$ using $O(x_2^{\epsilon}(1+x_2-x_1))$ operations and $O(x_2^{\epsilon})$ bits. We can enumerate primes more rapidly by reducing the number of primality tests with a "partial sieve"—i.e., by first sieving out those composite numbers with at least one divisor below y, for some fixed y in the range $2 \leq y \leq \sqrt{x_2}$. By letting y and $x_2 - x_1$ both grow with x_2, with $x_2 - x_1 \gg y \asymp x_2^{\vartheta}$ for some fixed ϑ, $0 < \vartheta \leq 1/2$, we can reduce the number of primality tests by a factor of $\vartheta \ln(x_2)$. (To prove this, see Corollary 6.19 on page 135.)

In Chapter 6, beginning on page 122, we develop a "hybrid sieve" which uses this technique of partial sieving, but which replaces strict primality tests with faster probable primality tests. Provided $x_2 - x_1 \geq \sqrt{x_2}$ this sieve requires $O(\ln\ln(x_2)(x_2 - x_1))$ operations to enumerate the primes in the interval $[x_1, x_2]$. Also, we conjecture that this hybrid sieve requires only $O(x_2^{1/4})$ bits of memory. Although we have not proven this conjecture, we do provide some arguments and computational evidence to support it.

5 Enumerating Primes with a Dissected Sieve

5.1 Introduction

In this chapter we describe a sieve that operates efficiently while using notably less memory than sieves previously described in the literature. We make use of the terminology and analyses given in Section 4.2.

Our new sieving algorithm uses ideas from a recently developed sieve of Atkin and Bernstein [AB02]. We modify their sieve by using ideas developed by Voronoï for analyzing the Dirichlet divisor problem [Vor03]—arriving at a "dissected sieve" that enumerates primes in the interval $[x_1, x_2]$ using $O(x_2^{1/3})$ bits and $O(x_2 - x_1 + x_2^{1/3})$ arithmetic operations on numbers of $O(\ln x_2)$ bits. Thus, provided $x_2 - x_1 \gg x_2^{1/3}$, the dissected sieve requires $O(1)$ arithmetic operations per unit subinterval to sieve the interval $[x_1, x_2]$.

5.2 The Atkin-Bernstein sieve

Atkin and Bernstein base their sieve on classical theorems that relate primality to properties of binary quadratic forms [AB02]. Theorem 5.1, below, paraphrases their formulation. (It uses a different but equivalent condition for case (a), and is stated so that there is no overlap between the congruence classes considered.)

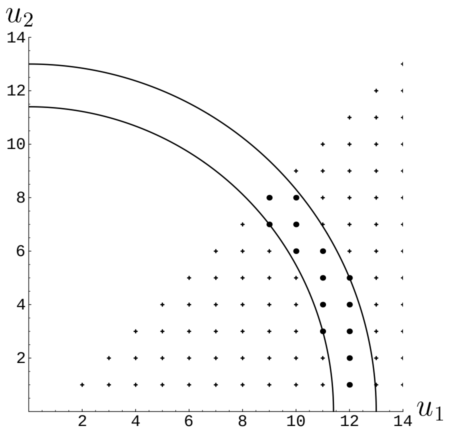
Theorem 5.1 *Let n be a positive integer,* $\gcd(n, 6) = 1$, *and*

(a) if $n \equiv 1 \pmod 4$ let $\mathcal{R} = \{(u_1, u_2) : \ u_1 > u_2 > 0\}$ and let $Q(u_1, u_2) = u_1^2 + u_2^2$;

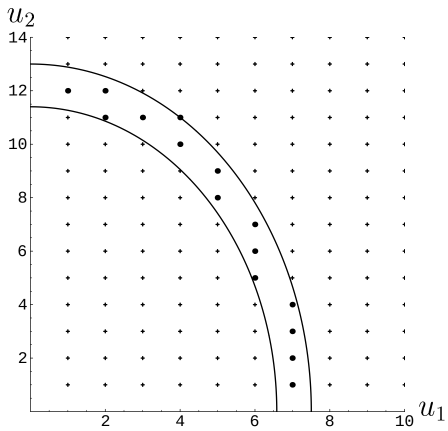
(b) if $n \equiv 7 \pmod{12}$ let $\mathcal{R} = \{(u_1, u_2) : \ u_1 > 0, u_2 > 0\}$ and let $Q(u_1, u_2) = 3u_1^2 + u_2^2$;

(c) if $n \equiv 11 \pmod{12}$ let $\mathcal{R} = \{(u_1, u_2) : \ u_1 > u_2 > 0\}$ and let $Q(u_1, u_2) = 3u_1^2 - u_2^2$.

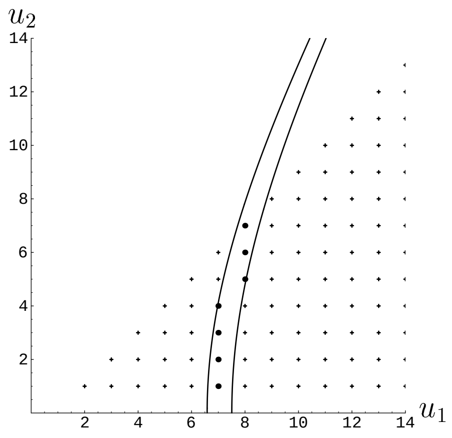
Let $P(n) = |\{(u_1, u_2) \in \mathbb{Z}^2 \cap \mathcal{R} : \ Q(u_1, u_2) = n\}|$. Then n is prime if and only if n is squarefree and $P(n)$ is odd.



(a) $n \bmod 4 = 1$, $n = u_1^2 + u_2^2$



(b) $n \bmod 12 = 7$, $n = 3u_1^2 + u_2^2$



(c) $n \bmod 12 = 11$, $n = 3u_1^2 - u_2^2$

Figure 5.1: The three cases of Theorem 5.1 and of Algorithms 5.1 and 5.2. Darkened points lie within a "swath", which is the set of points bounded by the conditions $x_1 \leq Q(u_1, u_2) \leq x_2$, $(u_1, u_2) \in \mathcal{R}$, where $Q(u_1, u_2)$ and $\mathcal{R}$ are defined case-by-case in Theorem 5.1.

For any n not divisible by 2 or 3, Theorem 5.1 gives a method for determining the primality of a single n in $O(\sqrt{n})$ operations. Atkin and Bernstein observed that Theorem 5.1 is much more efficient when used to determine the primality of all n, $x_1 \le n \le x_2$, provided $x_2 - x_1$ and the available memory are large enough. (Just as the $O(\sqrt{n})$ method of trial division for determining the primality of a single n leads to the much more efficient sieve of Eratosthenes.) We present an unsegmented version of the Atkin-Bernstein sieve in Algorithm 5.1, below.

The purpose of Algorithm 5.1 is to set the bits in a bit vector `Pset` so that $\texttt{Pset}[n] = 1$ if and only if n is prime. In the following discussion we let $x_1 = \texttt{Pset.x1}$, $x_2 = \texttt{Pset.x2}$. After initializing $\texttt{Pset}[n]$ to zero, $x_1 \le n \le x_2$, Algorithm 5.1 enumerates lattice points $(u_1, u_2) \in \mathcal{R}$ bounded between (or lying on) the conics $Q(u_1, u_2) = x_1$ and $Q(u_1, u_2) = x_2$, where $\mathcal{R}$ and the matching $Q(u_1, u_2)$ range through the three cases of Theorem 5.1. For each such point, the corresponding n is complemented when n is in the congruence class appropriate for the quadratic form. Having computed $\texttt{Pset}[n] = P(n) \bmod 2$, the algorithm makes a final pass to sieve out numbers with square factors.

For a given quadratic form and associated congruence class, the lattice points within the swath $x_1 \le Q(u_1, u_2) \le x_2$, $(u_1, u_2) \in \mathcal{R}$, are enumerated by varying u_1, and then for fixed u_1 incrementing u_2 along a vertical scanline, choosing the starting and ending values of u_2 to avoid points outside the swath. Enumerated points are illustrated in Figure 5.1 and correspond to the darkened points within a swath. (The figure shows all lattice points within a swath, not just those satisfying the corresponding congruence condition.)

Algorithm 5.1 (`AtkinBernsteinSieve`: Sieve of Atkin & Bernstein)

Given a preallocated bit vector `Pset` *with* $\texttt{Pset.x1} = x_1$, $\texttt{Pset.x2} = x_2$, $3 < x_1 \le x_2$, *this algorithm sets* $\texttt{Pset}[n]$ *such that upon completion we have* $\texttt{Pset}[n] = 1$ *if and only if* n *is prime.*

```
AtkinBernsteinSieve(Pset) {
 x₁ ←Pset.x1; x₂ ←Pset.x2;
 assert x₁ ∈ ℕ ∧ x₂ ∈ ℕ; assert 3 < x₁ ≤ x₂;
 for (n ← x₁; n ≤ x₂; n++)    Pset[n] ← 0;
 // Case (a) n ≡ 1 (mod 4), handles n mod 12 ∈ {1, 5, 9}.
```

6 **for** $(u_1 \leftarrow \lceil (x_1/2)^{1/2} \rceil\, ; u_1^2 \le x_2\, ; u_1{+}{+})$

7   **for** $(u_2 \leftarrow \lceil \max(0,\ x_1 - u_1^2)^{1/2} \rceil\, ; u_2 < u_1\ \&\&\ (n \leftarrow u_1^2 + u_2^2) \le x_2\, ; u_2{+}{+})$

8   **if** $(n \bmod 4 = 1)$ `Pset`$[n] \leftarrow$`Pset`$[n] + 1 \bmod 2$;

9 *// Case (b)* $n \equiv 7 \pmod{12}$.

10 **for** $(u_1 \leftarrow 1\, ; 3u_1^2 \le x_2\, ; u_1{+}{+})$

11   **for** $(u_2 \leftarrow \lceil \max(0,\ x_1 - 3u_1^2)^{1/2} \rceil\, ;\ \ (n \leftarrow 3u_1^2 + u_2^2) \le x_2\, ; u_2{+}{+})$

12   **if** $(n \bmod 12 = 7)$ `Pset`$[n] \leftarrow$`Pset`$[n] + 1 \bmod 2$;

13 *// Case (c)* $n \equiv 11 \pmod{12}$.

14 **for** $(u_1 \leftarrow \lceil (x_1/3)^{1/2} \rceil\, ; 2u_1^2 \le x_2\, ; u_1{+}{+})$

15   **for** $(u_2 \leftarrow \lceil \max(0,\ 3u_1^2 - x_2)^{1/2} \rceil\, ; u_2 < u_1\ \&\&\ (n \leftarrow 3u_1^2 - u_2^2) \ge x_1\, ; u_2{+}{+})$

16   **if** $(n \bmod 12 = 11)$ `Pset`$[n] \leftarrow$`Pset`$[n] + 1 \bmod 2$;

17 *// Sieve out numbers with square factors.*

18 **for** $(q \leftarrow 3\, ; q^2 \le x_2\, ; q{+}{+})$

19   **for** $(m \leftarrow \lceil x_1/q^2 \rceil\, ; mq^2 \le x_2\, ; m{+}{+})$

20   `Pset`$[mq^2] \leftarrow 0\, ; \}$

The presentation of Algorithm 5.1 given here is quite different from that of Atkin and Bernstein. They avoid enumerating points and allocating storage for n divisible by small primes, and they use difference equations satisfied by the quadratic forms to reduce the number of multiplications and square roots required. However, both versions have the same basic complexity. (Atkin and Bernstein also mention a modification which reduces the operation count of their sieve by a factor of $\ln\ln x_2$, at the cost of slightly increased memory requirements.)

We can bound the operation count $\mathcal{K}$ for Algorithm 5.1 by

$$\mathcal{K} \ll \mathcal{K}_1 + \mathcal{K}_2 + \mathcal{K}_3 + \mathcal{K}_4$$

where $\mathcal{K}_1$ is the size of the interval $[x_1, x_2]$, $\mathcal{K}_2$ the number of lattice points enumerated, $\mathcal{K}_3$ the number of scanlines (values of u_1) enumerated, and $\mathcal{K}_4$ the number of operations performed by the squarefree sieve of lines 18–20.

Trivially, $\mathcal{K}_1 = 1 + x_2 - x_1$, and it is easy to show that $\mathcal{K}_3 \ll x_2^{1/2}$ and that $\mathcal{K}_4 \ll x_2 - x_1 + x_2^{1/2}$. The number of lattice points, $\mathcal{K}_4$, is closely related to the area of the swath they lie in—the number differing from the area by an amount which is $O(x_2^{1/2})$. For each of the three cases, the area of the swath is $O(x_2 - x_1)$. We conclude that the number of lattice points, and the total operation count for Algorithm 5.1, is $O(x_2 - x_1 + x_2^{1/2})$. The corresponding segmented sieve, working with subintervals of size bounded by B, requires $O((1 + x_2^{1/2}/B)(x_2 - x_1) + x_2^{1/2})$ operations. Thus, the segmented version is inefficient if B grows more slowly than $x_2^{1/2}$.

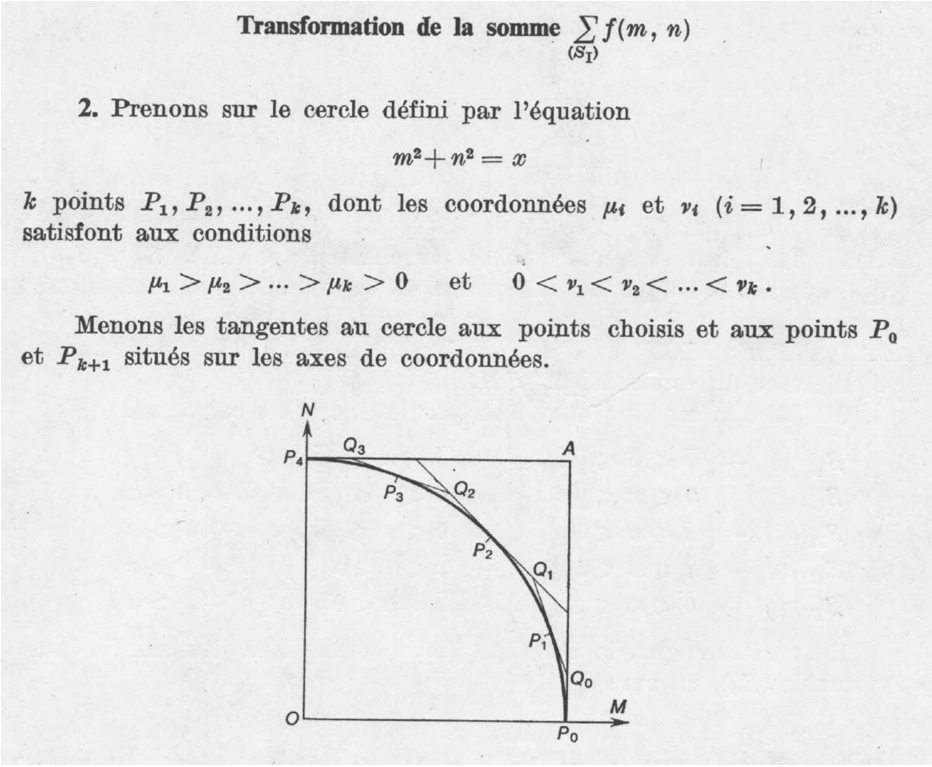
We may refine the bound of $O(x_2^{1/2})$ given above for the difference between the area of a swath and the number of points within it. The problem of estimating this difference is closely related to the *circle problem*, which is concerned with estimating the difference between the number of lattice points within a circle of radius $\sqrt{x}$ and its area. By a result of van der Corput [vdC20] it follows that for each case of Algorithm 5.1 the number of points enumerated is $O(x_2 - x_1 + x_2^{1/3})$. Van der Corput's result generalizes earlier work of Voronoï on the Dirichlet divisor problem [Vor03] and of Sierpiński on the circle problem [Sie74]. (See Section 5.9 on page 120 of this dissertation, and the discussion following Theorem 2.4.2 in [Hux96].)

Van der Corput's result suggests our dissected sieve. This sieve reduces the number of scanlines needed to enumerate points, and thus reduces the fixed overhead of $O(x_2^{1/2})$ operations for Algorithm 5.1 to $O(x_2^{1/3})$ operations for our sieve. The key idea behind van der Corput's result is to dissect the region into pieces, and to estimate the number of points within each piece by scanning roughly tangent to the boundary curve (see Figure 5.2 on the following page for the case of the circle). In our dissected sieve we dissect the swath into pieces, and then scan each piece in a direction roughly tangent to the boundary curves. (See Figure 5.3 on the next page, and also Figure 5.4 on page 100.) We choose tangents with slopes defined by a Farey sequence of order r, and then use corresponding "cuts" to separate the pieces. The optimal choice for r is discussed in Section 5.5 beginning on page 103, where we analyze the number of operations performed by the dissected sieve.

5.3 Notation and background material

Before giving details of the dissected sieve, we introduce some notation and state, often without proof, some properties of quadratic forms and of Farey sequences.

We use vector notation, with vectors denoted by lowercase boldface letters, and matrices by uppercase boldface letters. The transpose of $\boldsymbol{A}$ is written $\boldsymbol{A}^t$. Vectors are always treated as column vectors, although often written as the transpose of a row vector as in "$\boldsymbol{u} = [u_1\ u_2]^t$". We often write



Transformation de la somme $\sum_{(S_1)} f(m, n)$

2. Prenons sur le cercle défini par l'équation

$$m^2 + n^2 = x$$

k points $P_1, P_2, \ldots, P_k$, dont les coordonnées μ_i et ν_i $(i = 1, 2, \ldots, k)$ satisfont aux conditions

$$\mu_1 > \mu_2 > \ldots > \mu_k > 0 \quad \text{et} \quad 0 < \nu_1 < \nu_2 < \ldots < \nu_k .$$

Menons les tangentes au cercle aux points choisis et aux points P_0 et P_{k+1} situés sur les axes de coordonnées.

Figure 5.2: Sierpiński's dissection of the circle [Sie74, pg. 76]

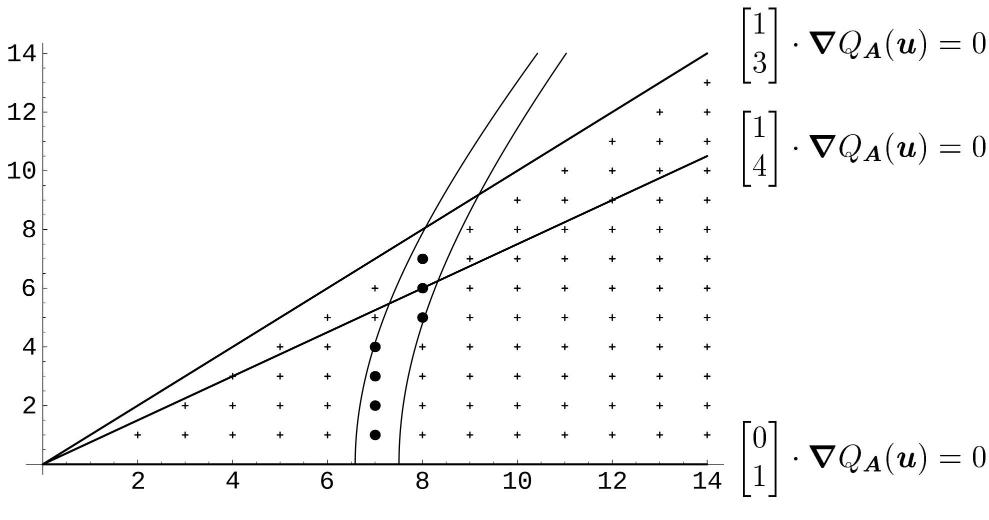


Figure 5.3: A dissection for $Q_{\boldsymbol{A}}(\boldsymbol{u}) = 3u_1^2 - u_2^2$, using three cuts

a matrix as two column vectors, as in "$\boldsymbol{T} := [\boldsymbol{\tau} \ \ \boldsymbol{\tau}']$".

Definition 5.2 Given a symmetric matrix $\boldsymbol{A}$, we define the associated *inner product* to be $\langle \boldsymbol{u}, \boldsymbol{v} \rangle_{\boldsymbol{A}} := \boldsymbol{u}^t \boldsymbol{A} \boldsymbol{v}$. □

Remark Our inner product is a symmetric bilinear form [Coh93, §2.5.1]. □

Definition 5.3 Given $\boldsymbol{u} = [u_1 \ u_2]^t$ and a symmetric matrix $\boldsymbol{A}$, define the quadratic form $Q_{\boldsymbol{A}}(\boldsymbol{u})$ as

$$Q_{\boldsymbol{A}}(\boldsymbol{u}) := \langle u, u \rangle_{\boldsymbol{A}} = a_1 u_1^2 + a_2 u_1 u_2 + a_3 u_2^2,$$

where $\boldsymbol{A}$ and the coefficients a_j are related by

$$\boldsymbol{A} = \begin{bmatrix} a_1 & a_2/2 \\ a_2/2 & a_3 \end{bmatrix}.$$ □

Note: In many cases we will require that $Q_{\boldsymbol{A}}(\boldsymbol{u})$ be a diagonal form, i.e., that $a_2 = 0$.

Lemma 5.4 *Given a symmetric matrix $\boldsymbol{A}$, we have*

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle_{\boldsymbol{A}} = \langle \boldsymbol{v}, \boldsymbol{u} \rangle_{\boldsymbol{A}} \tag{5.1}$$

$$\langle \alpha \, \boldsymbol{u} + \beta \, \boldsymbol{v}, \boldsymbol{w} \rangle_{\boldsymbol{A}} = \alpha \langle \boldsymbol{u}, \boldsymbol{w} \rangle_{\boldsymbol{A}} + \beta \langle \boldsymbol{v}, \boldsymbol{w} \rangle_{\boldsymbol{A}}, \tag{5.2}$$

$$Q_{\boldsymbol{A}}(\alpha \, \boldsymbol{u}) = \alpha^2 Q_{\boldsymbol{A}}(\boldsymbol{u}), \tag{5.3}$$

$$Q_{\boldsymbol{A}}(\boldsymbol{u} + \boldsymbol{v}) = Q_{\boldsymbol{A}}(\boldsymbol{u}) + Q_{\boldsymbol{A}}(\boldsymbol{v}) + 2\langle \boldsymbol{u}, \boldsymbol{v} \rangle_{\boldsymbol{A}}, \tag{5.4}$$

$$\boldsymbol{\nabla} Q_{\boldsymbol{A}}(\boldsymbol{u}) = 2\boldsymbol{A}\boldsymbol{u}. \tag{5.5}$$

Definition 5.5 Given a quadratic form $Q_{\boldsymbol{A}}(\boldsymbol{u})$ and vector $\boldsymbol{\tau}$, let the *cutting line for $\boldsymbol{\tau}$*, or *$\boldsymbol{\tau}$-cut*, be the set of points $\boldsymbol{u}$ at which the gradient $\boldsymbol{\nabla} Q_{\boldsymbol{A}}(\boldsymbol{u})$ is perpendicular to $\boldsymbol{\tau}$, i.e.,

$$\{\boldsymbol{u} : \ \boldsymbol{\tau} \cdot \boldsymbol{\nabla} Q_{\boldsymbol{A}}(\boldsymbol{u}) = 0\} = \{\boldsymbol{u} : \ \langle \boldsymbol{\tau}, \boldsymbol{u} \rangle_{\boldsymbol{A}} = 0\} = \{\boldsymbol{u} : \ \boldsymbol{\kappa} \cdot \boldsymbol{u} = 0\},$$

where $\boldsymbol{\kappa} = \boldsymbol{A}\boldsymbol{\tau}$. □

The $\boldsymbol{\tau}$-cut is a line passing through the origin. It depends on the quadratic form $Q_{\boldsymbol{A}}(\boldsymbol{u})$ as well as on $\boldsymbol{\tau}$; the quadratic form should be clear from context. For a given x, the curve $Q_{\boldsymbol{A}}(\boldsymbol{u}) = x$ is parallel to $\boldsymbol{\tau}$ where the curve intersects the $\boldsymbol{\tau}$-cut. In other words, given a line $\boldsymbol{\kappa} \cdot \boldsymbol{u} = 0$ the vector $\boldsymbol{\tau} = \boldsymbol{A}^{-1}\boldsymbol{\kappa}$ is the tangent vector at the intersection of $\boldsymbol{\kappa} \cdot \boldsymbol{u} = 0$ and the curve $Q_{\boldsymbol{A}}(\boldsymbol{u}) = x$.

We define $\mathcal{F}(r)$, the Farey sequence of order r, to be the ascending sequence of reduced fractions β/α between 0 and 1, with denominators bounded by r: $\mathcal{F}(r) := \{\beta/\alpha : \ \gcd(\alpha, \beta) = 1, \ 0 \leq \beta \leq \alpha \leq r\}$.

Lemma 5.6 (Properties of the Farey sequence)
If $\beta/\alpha < \beta'/\alpha'$ are consecutive elements of $\mathcal{F}(r)$, then

$$\alpha + \alpha' > r \qquad \textit{(Theorem 30 in Hardy and Wright [HW79])}$$

$$\alpha\beta' - \alpha'\beta = 1 \qquad \textit{(Theorem 28 in Hardy and Wright,)} \tag{5.6}$$

or, equivalently, $\dfrac{\beta'}{\alpha'} - \dfrac{\beta}{\alpha} = \dfrac{1}{\alpha\alpha'}$. *Also, for $r \geq 2$, we have*

$$|\mathcal{F}(r)| = \frac{3}{\pi^2}r^2 + O(r \ln r) \qquad \textit{(Theorem 330 in Hardy and Wright).}$$

Lemma 5.7 (Recursion for the Farey sequence) *Let $\beta_0/\alpha_0, \beta_1/\alpha_1, \ldots$ be the Farey sequence of order r. Then,*

$$\beta_0 = 0, \ \alpha_0 = 1; \qquad \beta_1 = 1, \ \alpha_1 = r;$$
$$\beta_{k+2} = \lfloor (r + \alpha_k)/\alpha_{k+1} \rfloor \ \beta_{k+1} - \beta_k;$$
$$\alpha_{k+2} = \lfloor (r + \alpha_k)/\alpha_{k+1} \rfloor \ \alpha_{k+1} - \alpha_k.$$

Proof: See [Knu75, Exercises 1.3.2.18 and 1.3.2.19] and [HW79, §3.4]. ■

5.4 Dissecting the Atkin-Bernstein sieve

Algorithm 5.2 on the following page, and the component algorithms that follow, give an unsegmented version of the dissected sieve. Although we give a dissection which appears to be applicable to general quadratic forms, it should be noted that we have only analyzed and demonstrated the validity of the following algorithms for the particular quadratic forms of Theorem 5.1. Also, many decisions concerning details of the algorithms are rather arbitrary. We discuss possible improvements in Section 5.7.

To dissect a swath, Algorithm 5.2 uses a sequence of tangent vectors related to $\beta/\alpha \in \mathcal{F}(r)$, of the form $\boldsymbol{\tau} = [-\beta \ \alpha]^t$ in cases (a) and (b) of

Theorem 5.1, and of the form $\boldsymbol{\tau} = [\beta\ \alpha]^t$ in case (c), so that in all cases the corresponding cutting lines run through the upper-right quadrant. In case (c) we must terminate the sequence upon reaching a cut of slope 1, which occurs when $\beta/\alpha = 1/3$, so we must choose $r \geq 3$. To dissect the swath over the entire quadrant in case (b), we swap the roles of u_1 and u_2 and then perform a similar dissection for $Q_{\boldsymbol{A}}(\boldsymbol{u}) = u_1^2 + 3u_2^2$

We now give a formal definition of a "piece" of our dissection:

Definition 5.8 Given $x_1 \leq x_2$, quadratic form $Q_{\boldsymbol{A}}(\boldsymbol{u})$, and consecutive tangents $\boldsymbol{\tau}$ and $\boldsymbol{\tau}'$, the corresponding *piece* is the set of points $\boldsymbol{u}$ with $x_1 \leq Q_{\boldsymbol{A}}(\boldsymbol{u}) \leq x_2$ and with $\boldsymbol{u}$ between the $\boldsymbol{\tau}$-cut and the $\boldsymbol{\tau}'$-cut. We exclude points on the $\boldsymbol{\tau}$-cut and include points on the $\boldsymbol{\tau}'$-cut. □

Remark In cases (a) and (c) of Theorem 5.1 the included points lying on the very last $\boldsymbol{\tau}'$-cut, i.e., on the line $u_2 = u_1$, lie outside the corresponding region $\mathcal{R}$ (defined Theorem 5.1). Similarly, in case (b) the points on the line $u_2 = 3u_1$ are counted twice. Although this gives an incorrect calculation of $P(n) \bmod 2$ for n corresponding to these points, the end result is still correct because such n have square factors and are removed in the final pass. □

Algorithm 5.2 controls initialization, dissection into pieces, and calling of the squarefree sieve routine; while Algorithm 5.3 (`ScanPiece`) scans a single piece, and Algorithm 5.4 (`SquareFreeSieve`) does the final sieving to eliminate square factors. We present each algorithm in turn. It should be noted that these algorithms only use diagonal forms.

Algorithm 5.2 (`DissectedSieve`: Dissected sieve of order r)
Given $r \geq 3$ (the order of dissection), and a preallocated bit vector `Pset` *with* `Pset.x1` $= x_1$, `Pset.x2` $= x_2$, $3 < x_1 \leq x_2$, *this algorithm sets* `Pset`$[n]$ *such that upon completion we have* `Pset`$[n] = 1$ *if and only if n is prime.*

```
DissectedSieve(r, Pset) {
  assert r ≥ 3;
  assert Pset.x1 ∈ ℕ ∧ Pset.x2 ∈ ℕ;
  assert 3 < Pset.x1 ≤ Pset.x2;
  β ← 0; α ← 1;     // β/α, β'/α' denote successive Farey fractions of order r.
  β' ← 1; α' ← r;
  while(TRUE) {
    // Case (a) n ≡ 1 (mod 4), handles n mod 12 ∈ {1,5,9}.
```

```
    ScanPiece(1,4,[1 0; 0 1],[−β α]^t,[−β′ α′]^t,Pset);
    // Case (b) n ≡ 7 (mod 12).
    ScanPiece(7,12,[3 0; 0 1],[−β α]^t,[−β′ α′]^t,Pset);
    ScanPiece(7,12,[1 0; 0 3],[−β α]^t,[−β′ α′]^t,Pset);
    if(3β′ ≤ α′) {  // Case (c) n ≡ 11 (mod 12).
     ScanPiece(11,12,[3 0; 0 −1],[β α]^t,[β′ α′]^t,Pset);}
    if(β′ = α′)   break;
    // Advance to next Farey fraction of order r, β/α, using Lemma 5.7.
    k ← ⌊(r + α)/α′⌋;
    {β, β′} ← {β′, kβ′ − β};
    {α, α′} ← {α′, kα′ − α};
  }
  // Sieve out square factors, using Algorithm 5.4, presented on page 102.
  SquareFreeSieve(Pset);}
```

To scan a piece bounded by the cuts for tangents $\boldsymbol{\tau}$ and $\boldsymbol{\tau}'$, Algorithm 5.3 on page 101 transforms to another coordinate system, or "$\boldsymbol{v}$-space". The coordinates are related by $\boldsymbol{u} = \boldsymbol{T}\boldsymbol{v}$, with $\boldsymbol{T} := [\boldsymbol{\tau} \;\; \boldsymbol{\tau}']$, so the map $\boldsymbol{v} = \boldsymbol{T}^{-1}\boldsymbol{u}$ sends $\boldsymbol{\tau}$ to the unit horizontal vector and $\boldsymbol{\tau}'$ to the unit vertical vector. By the method used to construct $\boldsymbol{\tau}$ and $\boldsymbol{\tau}'$ from Farey fractions, Equation (5.6) implies $\det(\boldsymbol{T}) = \pm 1$, so the mapping is area-preserving and gives a one-one map between points in $\mathbb{Z}^2$ (see Figure 5.4 on the next page).

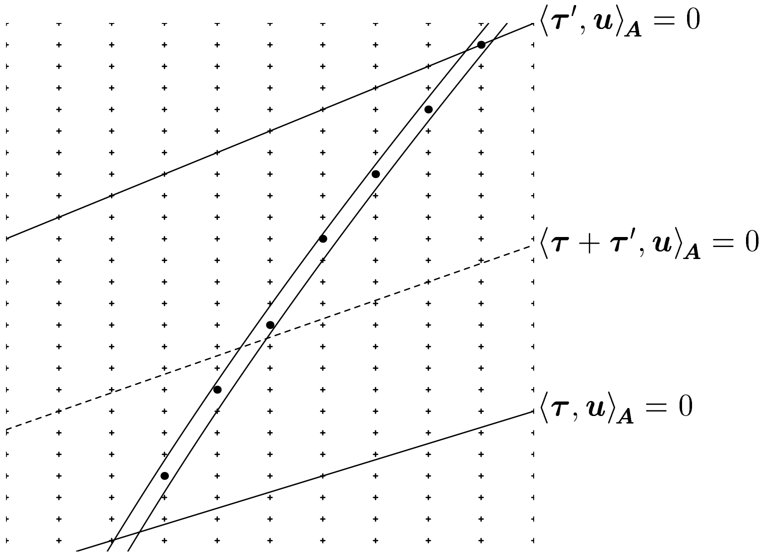
Working in $\boldsymbol{v}$-space, Algorithm 5.3 scans both horizontally and vertically, shifting between the horizontal and vertical directions at the image of the mediant-line, illustrated in Figure 5.4(a) and defined as follows:

Definition 5.9 Given a symmetric matrix $\boldsymbol{A}$ and vectors $\boldsymbol{\tau}$ and $\boldsymbol{\tau}'$, the associated *mediant-line* is the set of points $\boldsymbol{u}$ satisfying $\langle \boldsymbol{\tau} + \boldsymbol{\tau}', \boldsymbol{u} \rangle_{\boldsymbol{A}} = 0$. $\square$

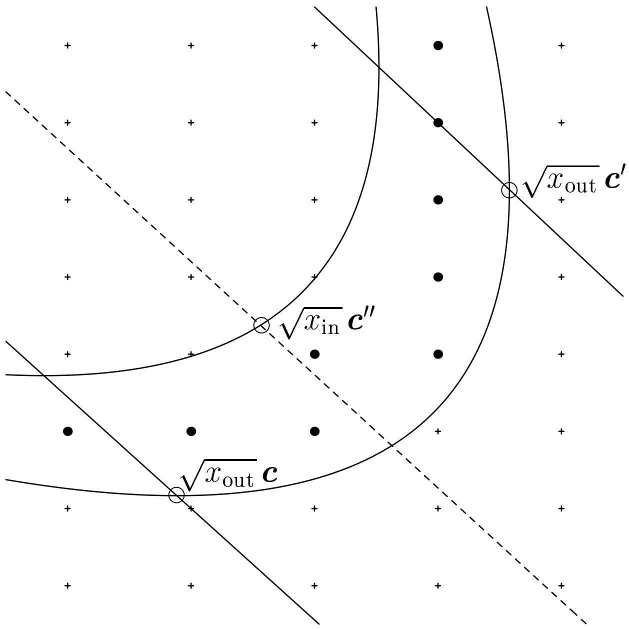
Algorithm 5.3 computes $Q_{\boldsymbol{A}}(\boldsymbol{u})$ using the identity $Q_{\boldsymbol{A}}(\boldsymbol{u}) = Q_{\boldsymbol{B}}(\boldsymbol{v}) = b_1 v_1^2 + b_2 v_1 v_2 + b_3 v_2^2$, with

$$\boldsymbol{B} := \boldsymbol{T}^t \boldsymbol{A} \boldsymbol{T} = \begin{bmatrix} \langle \boldsymbol{\tau}, \boldsymbol{\tau} \rangle_{\boldsymbol{A}} & \langle \boldsymbol{\tau}, \boldsymbol{\tau}' \rangle_{\boldsymbol{A}} \\ \langle \boldsymbol{\tau}', \boldsymbol{\tau} \rangle_{\boldsymbol{A}} & \langle \boldsymbol{\tau}', \boldsymbol{\tau}' \rangle_{\boldsymbol{A}} \end{bmatrix} = \begin{bmatrix} b_1 & b_2/2 \\ b_2/2 & b_3 \end{bmatrix}. \tag{5.7}$$

Writing $[\boldsymbol{b} \;\; \boldsymbol{b}'] := \boldsymbol{B}$, it follows that the cutting lines and their mediant-line



(a) Original coordinate system ($\boldsymbol{u}$-space), showing the two cutting lines (solid) and the mediant-line (dashed)



(b) Transformed coordinate system ($\boldsymbol{v}$-space), showing crossing points (circled) used by Algorithm 5.3

Figure 5.4: A piece of a dissection, in two coordinate systems. (Note that the horizontal axis and vertical axis are at different scales.)

have images in $\boldsymbol{v}$-space satisfying

$$\begin{aligned} \boldsymbol{b} \cdot \boldsymbol{v} &= 0 && \text{for the } \boldsymbol{\tau}\text{-cut} \\ \boldsymbol{b}' \cdot \boldsymbol{v} &= 0 && \text{for the } \boldsymbol{\tau}'\text{-cut} \\ (\boldsymbol{b} + \boldsymbol{b}') \cdot \boldsymbol{v} &= 0 && \text{for the mediant-line.} \end{aligned}$$

These equations follow easily from the definitions above. For example, for the $\boldsymbol{\tau}$-cut the following equations are equivalent:

$$\boldsymbol{\tau}^t \boldsymbol{A} \boldsymbol{u} = 0 \iff \boldsymbol{\tau}^t \boldsymbol{A} [\boldsymbol{\tau} \quad \boldsymbol{\tau}'] \boldsymbol{v} = 0 \iff \boldsymbol{b}^t \boldsymbol{v} = 0.$$

To bound the range of the scanlines, we use three *crossing points* illustrated in Figure 5.4(b), and defined as scalar multiples of "normalized" crossing points $\boldsymbol{c}$, $\boldsymbol{c}'$, $\boldsymbol{c}''$ given by

$$\boldsymbol{c} = \left|\det^{-1}(\boldsymbol{T})\right| (a_1 a_3 b_1)^{-1/2} \, [b_2/2 \quad - b_1]^t \tag{5.8}$$

$$\boldsymbol{c}' = \left|\det^{-1}(\boldsymbol{T})\right| (a_1 a_3 b_3)^{-1/2} \, [b_3 \quad - b_2/2]^t \tag{5.9}$$

$$\boldsymbol{c}'' = \left|\det^{-1}(\boldsymbol{T})\right| (a_1 a_3 (b_1 + b_2 + b_3))^{-1/2} \, [(b_2/2 + b_3) \quad (-b_1 - b_2/2)]^t. \tag{5.10}$$

(We allow for the possibility that $|\det(\boldsymbol{T})| \neq 1$ in future implementations of Algorithm 5.3.) For diagonal forms $Q_{\boldsymbol{A}}(\boldsymbol{u})$; which are the only type of form used in Algorithm 5.2; $\boldsymbol{c}$, $\boldsymbol{c}'$, $\boldsymbol{c}''$ lie at intersections between the curve $Q_{\boldsymbol{B}}(\boldsymbol{v}) = 1$ and the images in $\boldsymbol{v}$-space of the $\boldsymbol{\tau}$-cut, $\boldsymbol{\tau}'$-cut, and mediant-line, respectively.

Given $x \in \mathbb{R}$, Equation (5.3) implies that $\sqrt{x}\,\boldsymbol{c}$ lies at an intersection between $Q_{\boldsymbol{B}}(\boldsymbol{v}) = x$ and the image of the $\boldsymbol{\tau}$-cut, and similarly for $\boldsymbol{c}'$, $\boldsymbol{c}''$. Algorithm 5.3 sets $x_{\text{in}} = x_1$ and $x_{\text{out}} = x_2$, or $x_{\text{in}} = x_2$ and $x_{\text{out}} = x_1$, so as to make $\det(\boldsymbol{T})(x_{\text{out}} - x_{\text{in}}) > 0$. This is necessary to make the point $\sqrt{x_{\text{in}}}\,\boldsymbol{c}''$ lie above the point $\sqrt{x_{\text{out}}}\,\boldsymbol{c}$ and to the left of $\sqrt{x_{\text{out}}}\,\boldsymbol{c}'$.

Making use of the material above, Algorithm 5.3 proceeds in much the same way as Algorithm 5.1, although finding the endpoints for a scanline becomes more complicated because of the greater generality of the quadratics to be solved, and because each end of the scanline may be bounded by either a non-degenerate conic or by a line.

Algorithm 5.3 (`ScanPiece`: Process lattice points within a piece) *This routine enumerates all $\boldsymbol{u} \in \mathbb{Z} \times \mathbb{Z}$ lying within the piece corresponding to $\boldsymbol{\tau}$ and $\boldsymbol{\tau}'$ (in the sense of Definition 5.8 on page 98). Letting $n = Q_{\boldsymbol{A}}(\boldsymbol{u})$ for each $\boldsymbol{u}$ enumerated,* `Pset`$[n]$ *is complemented if $n \equiv k \pmod{m}$.*

```
ScanPiece(k, m, A, τ, τ', Pset) {
 T ← [τ  τ'];
  if(det(T) > 0)
   x_in ← Pset.x1;  x_out ← Pset.x2;
  else
   x_in ← Pset.x2;  x_out ← Pset.x1;
 b1 ← Q_A(τ);  b2 ← τ'^t A τ;  b3 ← Q_A(τ');  B ← [ b1  b2/2 ; b2/2  b3 ];
 c ← |det^{-1}(T)| (a1 a3 b1)^{-1/2} [b2/2   − b1]^t;
 c' ← |det^{-1}(T)| (a1 a3 b3)^{-1/2} [b3   − b2/2]^t;
 c'' ← |det^{-1}(T)| (a1 a3 (b1 + b2 + b3))^{-1/2} [(b2/2 + b3)   (−b1 − b2/2)]^t;
 // Scan the half-piece below, or on, the mediant-line (horizontal scanlines).
 for(v2 ← ⌈[0 1] √x_out c⌉; v2 ≤ ⌊[0 1] √x_in c''⌋;  v2++) {
  d ← max(0, b2² v2² − 4 b1 (b3 v2² − x_in));  // d = discriminant of quadratic, or zero.
  // v_start ∈ ℝ, v_stop ∈ ℝ give limits for the scanline.
  v_start ← (−b2 v2 + √d)/(2 b1);
  d ← b2² v2² − 4 b1 (b3 v2² − x_out);   assert d ≥ 0;
  v_stop ← min(−(2 b3 + b2) v2/(2 b1 + b2), (−b2 v2 + √d)/(2 b1));
  for(v1 ← 1 + ⌊v_start⌋;  v1 ≤ v_stop;  v1++) {
   n ← Q_B([v1 v2]^t);
    if(n mod m = k)   Pset[n] ← Pset[n] + 1 mod 2;}}
 // Scan the half-piece above, and off, the mediant-line (vertical scanlines).
 for(v1 ← ⌈[1 0] √x_in c''⌉; v1 ≤ ⌊[1 0] √x_out c'⌋;  v1++) {
  d ← b2² v1² − 4 b3 (b1 v1² − x_out);   assert d ≥ 0;
  v_start ← max(−(2 b1 + b2) v1/(2 b3 + b2), (−b2 v1 − √d)/(2 b3));
  d ← max(0, b2² v1² − 4 b3 (b2 v1² − x_in));
  v_stop ← (−b2 v1 − √d)/(2 b3);
  for(v2 ← 1 + ⌊v_start⌋;  v2 ≤ v_stop;  v2++) {
   n ← Q_B([v1 v2]^t);
    if(n mod m = k)   Pset[n] ← Pset[n] + 1 mod 2;}}}
```

Algorithm 5.4 is similar to the corresponding code of lines 18–20 in Algorithm 5.1, but uses "Dirichlet's hyperbola method" [Ten95, Part I, §3.2] to reduce the operation count from $O(x_2 - x_1 + x_2^{1/2})$ to $O(x_2 - x_1 + x_2^{1/3})$.

Algorithm 5.4 (`SquareFreeSieve`: **Sieve out n with square factors**)
This algorithm sets `Pset`$[n] = 0$ *for each n of the form $n = mq^2$, $q > 1$.*

`SquareFreeSieve(Pset) {`
 $x_1 \leftarrow$`Pset.x1;` $x_2 \leftarrow$`Pset.x2;`
  for $(q \leftarrow 3;\ q \le x_2^{1/3}; q++)$
    for $(m \leftarrow \lceil x_1/q^2 \rceil;\ mq^2 \le x_2;\ m{+}{+})$
      `Pset`$[mq^2] \leftarrow 0;$
  for $(m \leftarrow 1;\ m \le x_2^{1/3};\ m{+}{+})$
    for $(q \leftarrow \max(3, \lceil (x_1/m)^{1/2} \rceil);\ mq^2 \le x_2;\ q++)$
      `Pset`$[mq^2] \leftarrow 0;\}$

5.5 Choosing the order of dissection

We begin by presenting a lemma to be used several times in our analysis:

Lemma 5.10 *Given $x_2 \ge x_1 > 0$, we have*

$$\sqrt{x_2} - \sqrt{x_1} \le (x_2 - x_1)/\sqrt{x_2}. \tag{5.11}$$

Proof: Noting that $(x_2/x_1)^{1/2} \ge 1$, we have

$$x_2^{1/2} - x_1^{1/2} = x_2^{-1/2}(x_2 - (x_2/x_1)^{1/2} x_1) \le x_2^{-1/2}(x_2 - x_1). \qquad \blacksquare$$

The heart of our analysis will be to put a bound on the operation count for Algorithm 5.2 (`DissectedSieve`). Writing $\mathcal{K}$ for this count, we have

$$\mathcal{K} \ll \mathcal{K}_1 + \mathcal{K}_2 + \mathcal{K}_3 + \mathcal{K}_4 + \mathcal{K}_5,$$

where $\mathcal{K}_1$ is the size of the interval $[x_1, x_2]$, $\mathcal{K}_2$ the number of lattice points enumerated, $\mathcal{K}_3$ the number of scanlines enumerated, $\mathcal{K}_4$ the number of pieces enumerated, and $\mathcal{K}_5$ the number of operations required by Algorithm 5.4.

We proceed to bound each $\mathcal{K}_j$, and then $\mathcal{K}$. After completing the analysis, we will be able to prove Theorem 5.20 on page 115, which treats the choice of the order r which minimizes $\mathcal{K}$.

Trivially, $\mathcal{K}_1 = 1 + x_2 - x_1$. We next bound the other components which are independent of r, namely $\mathcal{K}_2$ and $\mathcal{K}_5$. To bound the number of lattice points, $\mathcal{K}_2$, we use material of van der Corput [vdC20]. Definition 5.11 and Theorem 5.12 below paraphrase his results.

Definition 5.11 We define a *Voronoï-Pfeiffer region* to be a subset of the plane bounded by n lines, λ_ν ($n \geq 0$; $\nu = 1, 2, \ldots, n$) and m convex[1] Jordan curves σ_μ ($m \geq 0$; $\mu = 1, 2, \ldots, m$), which together form a simple closed curve satisfying the conditions **A** and **B** below:

A. Each line λ_ν is either vertical or has a rational slope. The equation of the line may then be written uniquely as $a_\nu u_1 + b_\nu u_2 = c_\nu$, where a_ν, $b_\nu \in \mathbb{Z}$, $\gcd(a_\nu, b_\nu) = 1$, and where $a_\nu u_1 + b_\nu u_2 < c_\nu$ for all interior points of the region which are sufficiently near any point on the line.

B. Each Jordan curve σ_μ has an orientation and at every point P has a unique tangent (that is, at each point P of the Jordan curve, excluding the endpoints, the tangents from the two sides coincide), where the angle θ that this tangent makes with the positive u_1-axis varies monotonically and continuously with P over σ_μ. Furthermore, the coordinates of P are well-defined differentiable[2] functions $u_1(\theta)$ and $u_2(\theta)$ of θ (in the interval traversed by θ) with continuous derivatives $du/d\theta$, $dv/d\theta$. Consequently, the curve σ_μ has a length s, which we regard as positive when moving in the same sense as σ and negative when moving in the opposite direction.

In condition **B** the angle θ is only defined modulo 2π, since negative values and values $\geq 2\pi$ are not excluded. However, it follows from the convexity of σ_μ that θ ranges over an interval of length at most 2π. Let the point P vary continuously and monotonically over the convex Jordan curve σ_μ. The monotone and continuous variable θ is then determined at each point of the curve σ_μ by the given sense of direction of the curve, when it is given at a single point. Thus it suffices to fix the value of θ at a single point, e.g., at an endpoint. Because the coordinates of P are to be well defined functions of θ we exclude the possibility of θ remaining constant over an interval; thus θ is strictly increasing or strictly decreasing with P.

At each point P of σ_μ we define the radius of curvature, $\varrho(\theta)$, as $ds/d\theta$ (using the notation of case **B**). Since

$$u_1'(\theta) = \varrho(\theta)\cos(\theta), \qquad u_2'(\theta) = \varrho(\theta)\sin(\theta)$$

[1] A Jordan curve (that is, a continuous, non-self-intersecting curve in the plane) is called convex if it intersects every straight line in at most two distinct points.

[2] Only differentiable from one side at the endpoints.

it follows that $\varrho(\theta)$ is continuous over the interval traversed by θ. □

Theorem 5.12 *Let G be a Voronoï-Pfeiffer region parameterized by $x > 0$. Let the associated variables m, n, a_ν, b_ν which occurred in Definition 5.11 be bounded above, independent of x. Let the greatest radius of curvature along any Jordan curve σ_μ be $O(\sqrt{x})$. Then the number of lattice points inside and on the boundary of the region G is*

$$J(G) - \sum_{\nu=1}^{n} \frac{\psi(c_\nu) l_\nu}{\sqrt{a_\nu^2 + b_\nu^2}} + O(x^{1/3}).$$

In this formula $J(G)$ denotes the area of G, l_ν the length of the line λ_ν, and we let $\psi(c_\nu) := c_\nu - \lfloor c_\nu \rfloor - 1/2$.

Proof: See [vdC20]. ■

Recall from page 98 that Algorithm 5.2 runs through four quadratic forms. For each quadratic form, $Q_{\boldsymbol{A}}(\boldsymbol{u})$, it enumerates lattice points $\boldsymbol{u} = [u_1\ u_2]^t$ satisfying $x_1 \leq Q_{\boldsymbol{A}}(\boldsymbol{u}) \leq x_2$ which lie strictly above the horizontal axis and below (or on) a line with a slope which we will denote as $M_{\boldsymbol{A}}$. (Specifically, $M_{\boldsymbol{A}} = 1$ when $Q_{\boldsymbol{A}}(\boldsymbol{u}) = u_1^2 + u_2^2$ and when $Q_{\boldsymbol{A}}(\boldsymbol{u}) = 3u_1^2 - u_2^2$; $M_{\boldsymbol{A}} = 3$ when $Q_{\boldsymbol{A}}(\boldsymbol{u}) = 3u_1^2 + u_2^2$, and $M_{\boldsymbol{A}} = 1/3$ when $Q_{\boldsymbol{A}}(\boldsymbol{u}) = u_1^2 + 3u_2^2$.)

Theorem 5.14 on the following page bounds the number of points enumerated in this region. We begin by expressing the region in terms of a slightly simpler region, $G_{\boldsymbol{A}}(x)$, defined in the following lemma.

Lemma 5.13 *Let $Q_{\boldsymbol{A}}(\boldsymbol{u})$ be one of the quadratic forms used in Algorithm 5.2, and let $M_{\boldsymbol{A}}$ be the associated slope defined above. Writing $\boldsymbol{u} = [u_1\ u_2]^t$, define the associated region $G_{\boldsymbol{A}}(x)$ to be*

$$G_{\boldsymbol{A}}(x) := \{\boldsymbol{u} :\ Q_{\boldsymbol{A}}(\boldsymbol{u}) \leq x, 0 \leq u_2 \leq M_{\boldsymbol{A}} u_1\}.$$

Then $G_{\boldsymbol{A}}(x)$ is a Voronoï-Pfeiffer region parameterized by x, satisfying the conditions of Theorem 5.12.

Proof: For fixed $Q_{\boldsymbol{A}}(\boldsymbol{u})$ we see that $G_{\boldsymbol{A}}(x)$ satisfies Definition 5.11. The conditions of Theorem 5.12 are clearly satisfied, with the possible exception of the bound on the radius of curvature. To show that the radius of curvature is $O(\sqrt{x})$, we parameterize the curve $Q_{\boldsymbol{A}}(\boldsymbol{u}) = x$, writing $\boldsymbol{u}$ as a function of a parameter α. Writing u_1' for $du_1/d\alpha$, etc., we use a standard formula [KK68, Eq. 17.1-7] for the radius of curvature:

$$\varrho(\alpha) = \frac{(u_1'^2 + u_2'^2)^{3/2}}{u_1'u_2'' - u_2'u_1''}. \tag{5.12}$$

Three of our four quadratic forms may be written as $Q_{\boldsymbol{A}}(\boldsymbol{u}) = au_1^2 + bu_2^2$, where $a > 0$, $b > 0$. The corresponding curves may be parameterized as

$$u_1 = \sqrt{\frac{x}{a}}\cos(\alpha) \qquad u_2 = \sqrt{\frac{x}{b}}\sin(\alpha).$$

Applying (5.12) we find that

$$\varrho(\alpha) = \sqrt{abx}\big(\sin^2(\alpha)/a + \cos^2(\alpha)/b\big)^{3/2} \ll \sqrt{x}.$$

Our remaining quadratic form is $Q_{\boldsymbol{A}}(\boldsymbol{u}) = 3u_1^2 - u_2^2$, for which $M_{\boldsymbol{A}} = 1$. The segment of the curve $Q_{\boldsymbol{A}}(\boldsymbol{u}) = x$ with $0 \le u_2 \le u_1$ may be parameterized as

$$u_1 = \sqrt{\frac{x}{3}}\cosh(\alpha) \qquad u_2 = \sqrt{x}\sinh(\alpha) \qquad (0 \le \alpha \le \ln(2)/2).$$

Again, applying (5.12), we find that $\varrho(\alpha) = -\sqrt{3x}(\sinh^2(\alpha)/3 + \cosh^2(\alpha))^{3/2}$, and thus $\varrho(\alpha) \ll \sqrt{x}$ in the interval $0 \le \alpha \le \ln(2)/2$. ■

Theorem 5.14 *Let $\mathcal{K}_2$ denote the total number of lattice points enumerated by Algorithm 5.2. Then $\mathcal{K}_2 \ll x_2 - x_1 + x_2^{1/3}$.*

Proof: Fixing $Q_{\boldsymbol{A}}(\boldsymbol{u})$, we will first show that

$$\begin{aligned} &\big|\{\boldsymbol{u} \in \mathbb{Z}^2 :\ x_1 \le Q_{\boldsymbol{A}}(\boldsymbol{u}) \le x_2, 0 < u_2 \le u_1\}\big| \\ &\quad \ll x_2 - x_1 + x_2^{1/3}. \end{aligned} \tag{5.13}$$

Given $Q_{\boldsymbol{A}}(\boldsymbol{u})$, let $N(x)$ denote the number of lattice points in the region $G_{\boldsymbol{A}}(x)$ of Lemma 5.13. Theorem 5.12 then implies that the number of lattice points within $G_{\boldsymbol{A}}(x)$ is $c_1x + c_2\sqrt{x} + O(x^{1/3})$, where the term c_1x corresponds to the area of the region, while the term $c_2\sqrt{x}$ corresponds to the length of the linear boundaries. Furthermore, the number of points within the intersection of $G_{\boldsymbol{A}}(x)$ with the horizontal axis is $O(\sqrt{x_2} - \sqrt{x_1}) + O(1)$. We conclude that the left side of (5.13) is

$$
\begin{aligned}
&= N(x_2) - N(x_1-) + O(\sqrt{x_2} - \sqrt{x_1}) + O(1) \\
(5.14) \qquad &= O(x_2 - x_1) + O(\sqrt{x_2} - \sqrt{x_1}) + O(x_2^{1/3}) + O(1).
\end{aligned}
$$

Since Algorithm 5.2 requires $x_2 \geq 3$, Lemma 5.10 on page 103 implies that $\sqrt{x_2} - \sqrt{x_1} \ll x_2 - x_1$. Since $x_2 \geq 3$, the $O(x_2^{1/3})$ term dominates the $O(1)$ term in (5.14), and the bound (5.13) follows. Finally, summing over all four $Q_{\boldsymbol{A}}(\boldsymbol{u})$ used in Algorithm 5.2, the result follows. ■

We next show that Algorithm 5.4 (`SquareFreeSieve`) satisfies a similar bound on its operation count.

Theorem 5.15 *Let $\mathcal{K}_5$ denote the operation count for Algorithm 5.4. Then $\mathcal{K}_5 \ll x_2 - x_1 + x_2^{1/3}$.*

Proof: Referring to the listing of Algorithm 5.4 on page 103, we see that the operation count may be broken into three components:

$$
\begin{array}{lll}
 & O(1) & \text{Constant overhead} \\
(5.15) & + \displaystyle\sum_{3 \leq q \leq x_2^{1/3}} \sum_{\substack{m \\ x_1 \leq mq^2 \leq x_2}} O(1) & \text{for lines 3–5} \\
(5.16) & + \displaystyle\sum_{1 \leq m \leq x_2^{1/3}} \sum_{\substack{q \\ x_1 \leq mq^2 \leq x_2}} O(1) & \text{for lines 6–8.}
\end{array}
$$

The sum (5.15) is

$$
\ll \sum_{3 \leq q \leq x_2^{1/3}} 1 + \frac{x_2 - x_1}{q^2} \ll x_2^{1/3} + (x_2 - x_1) \int_2^{\infty} q^{-2}\, dq \ll x_2 - x_1 + x_2^{1/3}.
$$

Using Lemma 5.10, we find that the sum (5.16) is

$$
\ll x_2^{1/3} + (\sqrt{x_2} - \sqrt{x_1}) \int_2^{x_2^{1/3}} m^{-1/2}\, dm \ll x_2^{1/3} + (x_2 - x_1) x_2^{-1/3}.
$$

Totaling these bounds gives $O(x_2 - x_1 + x_2^{1/3})$ operations. ■

Turning now to estimating operation counts that depend on our parameter r, we begin by bounding $\mathcal{K}_3$, the number of scanlines enumerated by the dissected sieve. Figure 5.5 shows the crossing points used in our analysis.

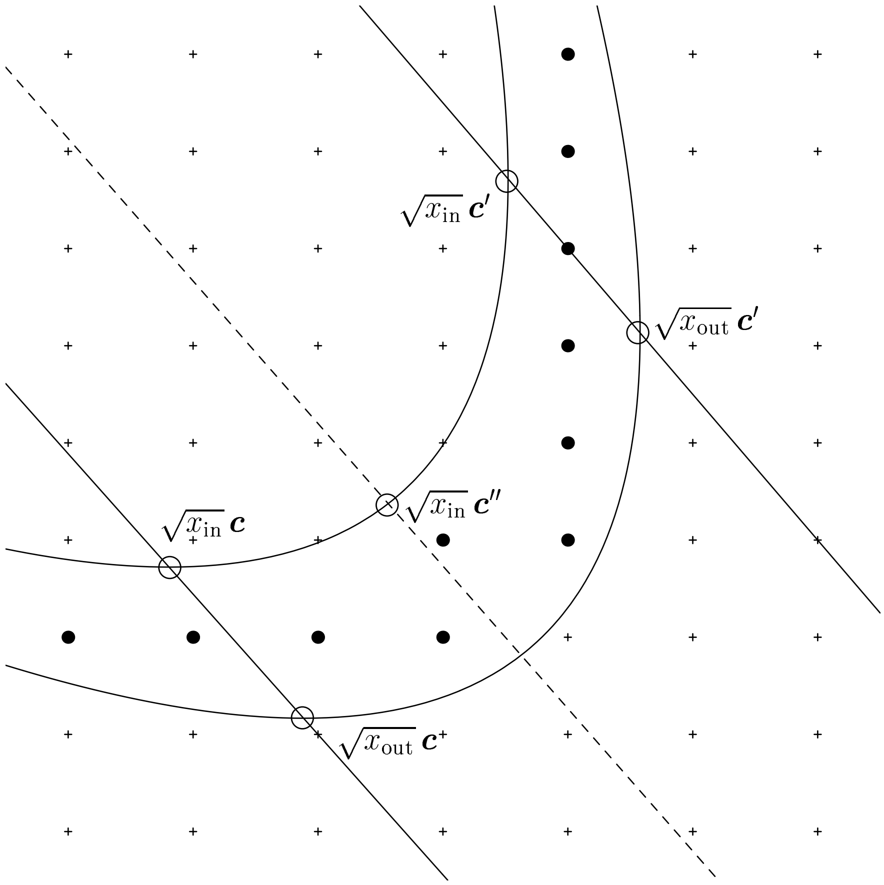


Figure 5.5: Crossing points (circled) used for scanline analysis. (Note that the horizontal axis and vertical axis are at different scales.)

Recall that for each $Q_{\boldsymbol{A}}(\boldsymbol{u})$ used in Algorithm 5.2 we run through tangents parallel to $\boldsymbol{\tau} = [\pm\beta \quad \alpha]^t$, $\boldsymbol{\tau}' = [\pm\beta' \quad \alpha']^t$, where $\beta/\alpha < \beta'/\alpha'$ are consecutive elements of the Farey sequence $\mathcal{F}(r)$, $\beta/\alpha < R_{\boldsymbol{A}}$, and where $R_{\boldsymbol{A}} := 1/3$ when $Q_{\boldsymbol{A}}(\boldsymbol{u}) = 3u_1^2 - u_2^2$, and $R_{\boldsymbol{A}} := 1$ in all other cases.

Using the definitions of $\boldsymbol{b}$ and $\boldsymbol{c}$ (see Equations (5.7) and Equation (5.8), starting on on page 99), and also using the fact that $|\det(\boldsymbol{T})| = 1$, where $\boldsymbol{T} = [\boldsymbol{\tau} \;\; \boldsymbol{\tau}']$, we find the alternative expressions:

$$\boldsymbol{c} = (a_1 a_3 Q_{\boldsymbol{A}}(\boldsymbol{\tau}))^{-1/2} \begin{bmatrix} \langle \boldsymbol{\tau}', \boldsymbol{\tau} \rangle_{\boldsymbol{A}} \\ -\langle \boldsymbol{\tau}, \boldsymbol{\tau} \rangle_{\boldsymbol{A}} \end{bmatrix} \tag{5.17}$$

$$\boldsymbol{c}' = (a_1 a_3 Q_{\boldsymbol{A}}(\boldsymbol{\tau}'))^{-1/2} \begin{bmatrix} \langle \boldsymbol{\tau}', \boldsymbol{\tau}' \rangle_{\boldsymbol{A}} \\ -\langle \boldsymbol{\tau}, \boldsymbol{\tau}' \rangle_{\boldsymbol{A}} \end{bmatrix} \tag{5.18}$$

$$\boldsymbol{c}'' = (a_1 a_3 Q_{\boldsymbol{A}}(\boldsymbol{\tau} + \boldsymbol{\tau}'))^{-1/2} \begin{bmatrix} \langle \boldsymbol{\tau}', \boldsymbol{\tau} + \boldsymbol{\tau}' \rangle_{\boldsymbol{A}} \\ -\langle \boldsymbol{\tau}, \boldsymbol{\tau} + \boldsymbol{\tau}' \rangle_{\boldsymbol{A}} \end{bmatrix}. \tag{5.19}$$

In these expressions, note that x_{in}, x_{out} depend on the sign of $\det(\boldsymbol{T})$, and that, throughout our dissection, $\det(\boldsymbol{T}) = \operatorname{sgn}(a_1 a_3) = \operatorname{sgn}(Q_{\boldsymbol{A}}(\boldsymbol{\tau}))$. Thus, in expressions like $(a_1 a_3 Q_{\boldsymbol{A}}(\boldsymbol{\tau}))^{-1/2}$ we take the root of a non-negative value.

Focusing on a single piece, we group the scanlines into two classes. The first class consists of those horizontal scanlines between the central point $\sqrt{x_{\text{in}}}\,\boldsymbol{c}''$ and the point $\sqrt{x_{\text{in}}}\,\boldsymbol{c}$, and those vertical scanlines between $\sqrt{x_{\text{in}}}\,\boldsymbol{c}''$ and $\sqrt{x_{\text{in}}}\,\boldsymbol{c}'$. We see that the number of such scanlines is

$$O(1) + \sqrt{x_{\text{in}}}\,([0 \quad 1](\boldsymbol{c}'' - \boldsymbol{c}) + [1 \quad 0](\boldsymbol{c}' - \boldsymbol{c}'')). \tag{5.20}$$

The second class consists of those remaining scanlines (horizontal and vertical) lying between the curves $Q_{\boldsymbol{B}}(\boldsymbol{v}) = x_{\text{in}}$ and $Q_{\boldsymbol{B}}(\boldsymbol{v}) = x_{\text{out}}$. Recalling that $Q_{\boldsymbol{A}}(\boldsymbol{u}) = \langle \boldsymbol{u}, \boldsymbol{u} \rangle_{\boldsymbol{A}}$, we see that the number of such scanlines is

$$\begin{aligned} & O(1) + \operatorname{sgn}(a_1 a_3)(\sqrt{x_2} - \sqrt{x_1})([1 \quad 0]\boldsymbol{c}' - [0 \quad 1]\boldsymbol{c}) \\ & \quad \ll 1 + \left(\sqrt{Q_{\boldsymbol{A}}(\boldsymbol{\tau})/(a_1 a_3)} + \sqrt{Q_{\boldsymbol{A}}(\boldsymbol{\tau}')/(a_1 a_3)}\right)(x_2 - x_1)/\sqrt{x_2}, \end{aligned} \tag{5.21}$$

by (5.17), (5.18), and by Lemma 5.10 on page 103.

To bound the total number of scanlines we sum (5.20) and (5.21) over all pieces. The number of summands is $O(|\mathcal{F}(r)|) \ll r^2$, by Lemma 5.6. Letting

$$C_{-1}(r;\boldsymbol{A}) := \sum_{\substack{\beta/\alpha\in\mathcal{F}(r)\\ \beta/\alpha<R_{\boldsymbol{A}}}} [0 \quad 1](\boldsymbol{c}''-\boldsymbol{c}) + [1 \quad 0](\boldsymbol{c}'-\boldsymbol{c}''), \tag{5.22}$$

and

$$C_3(r;\boldsymbol{A}) := \sum_{\substack{\beta/\alpha\in\mathcal{F}(r)\\ \beta/\alpha<R_{\boldsymbol{A}}}} \left(\sqrt{\frac{Q_{\boldsymbol{A}}([\pm\beta \quad \alpha]^t)}{a_1a_3}} + \sqrt{\frac{Q_{\boldsymbol{A}}([\pm\beta' \quad \alpha']^t)}{a_1a_3}}\right). \tag{5.23}$$

We see by Equations (5.20) and (5.21) that the total number of scanlines, over all pieces, is the sum over our four quadratic forms of

$$O(r^2) + \sqrt{x_2}\, C_{-1}(r;\boldsymbol{A}) + C_3(r;\boldsymbol{A})(x_2-x_1)/\sqrt{x_2}. \tag{5.24}$$

In the following two lemmas we bound $C_{-1}(r;\boldsymbol{A})$ and then $C_3(r;\boldsymbol{A})$.

Lemma 5.16 *We have* $C_{-1}(r;\boldsymbol{A}) \ll 1/r$.

Proof: Expanding a single term in (5.22), we have

$$\begin{aligned}
&[0 \quad 1](\boldsymbol{c}''-\boldsymbol{c}) + [1 \quad 0](\boldsymbol{c}'-\boldsymbol{c}'') \\
&= \frac{\langle\boldsymbol{\tau},\boldsymbol{\tau}\rangle_{\boldsymbol{A}}}{\sqrt{a_1a_3Q_{\boldsymbol{A}}(\boldsymbol{\tau})}} - \frac{\langle\boldsymbol{\tau},\boldsymbol{\tau}+\boldsymbol{\tau}'\rangle_{\boldsymbol{A}}}{\sqrt{a_1a_3Q_{\boldsymbol{A}}(\boldsymbol{\tau}+\boldsymbol{\tau}')}} + \frac{\langle\boldsymbol{\tau}',\boldsymbol{\tau}'\rangle_{\boldsymbol{A}}}{\sqrt{a_1a_3Q_{\boldsymbol{A}}(\boldsymbol{\tau}')}} - \frac{\langle\boldsymbol{\tau}',\boldsymbol{\tau}+\boldsymbol{\tau}'\rangle_{\boldsymbol{A}}}{\sqrt{a_1a_3Q_{\boldsymbol{A}}(\boldsymbol{\tau}+\boldsymbol{\tau}')}}
\end{aligned}$$

by Equations (5.17) through (5.19). Recalling that Algorithm 5.2 exclusively uses diagonal forms, we find that this simplifies by the definition of $Q_{\boldsymbol{A}}(\boldsymbol{u})$ as $\langle\boldsymbol{u},\boldsymbol{u}\rangle_{\boldsymbol{A}}$ and by the bilinearity of $\langle,\rangle_{\boldsymbol{A}}$, to yield

$$= \frac{Q_{\boldsymbol{A}}(\boldsymbol{\tau})}{\sqrt{a_1a_3Q_{\boldsymbol{A}}(\boldsymbol{\tau})}} + \frac{Q_{\boldsymbol{A}}(\boldsymbol{\tau}')}{\sqrt{a_1a_3Q_{\boldsymbol{A}}(\boldsymbol{\tau}')}} - \frac{Q_{\boldsymbol{A}}(\boldsymbol{\tau}+\boldsymbol{\tau}')}{\sqrt{a_1a_3Q_{\boldsymbol{A}}(\boldsymbol{\tau}+\boldsymbol{\tau}')}}$$

$$= \operatorname{sgn}(a_1a_3)\left(\sqrt{\frac{Q_{\boldsymbol{A}}(\boldsymbol{\tau})}{a_1a_3}} + \sqrt{\frac{Q_{\boldsymbol{A}}(\boldsymbol{\tau}')}{a_1a_3}} - \sqrt{\frac{Q_{\boldsymbol{A}}(\boldsymbol{\tau}+\boldsymbol{\tau}')}{a_1a_3}}\right). \tag{5.25}$$

Let $F(\rho) := \sqrt{Q_{\boldsymbol{A}}([\rho \quad 1]^t)/(a_1a_3)}$. Equation (5.3) gives

$$\sqrt{Q_{\boldsymbol{A}}([\pm\beta \quad \alpha]^t)/(a_1a_3)} = \alpha F(\beta/\alpha),$$

so, dropping the "sgn" factor from (5.25), we get

$$\begin{aligned} &\sqrt{Q_{\boldsymbol{A}}(\boldsymbol{\tau})/(a_1a_3)} + \sqrt{Q_{\boldsymbol{A}}(\boldsymbol{\tau}')/(a_1a_3)} - \sqrt{Q_{\boldsymbol{A}}(\boldsymbol{\tau}+\boldsymbol{\tau}')/(a_1a_3)} \\ &\quad = \alpha F\left(\frac{\beta}{\alpha}\right) + \alpha' F\left(\frac{\beta'}{\alpha'}\right) - (\alpha+\alpha')F\left(\frac{\beta+\beta'}{\alpha+\alpha'}\right). \end{aligned} \tag{5.26}$$

Taylor's Theorem, and a simple computation, give

$$F(\rho+\delta) = F(\rho) + \frac{\rho}{a_3F(\rho)}\delta + \frac{1}{2a_1a_3F^3(\rho)}\delta^2 + O(\delta^3),$$

while restricting ρ and $\rho+\delta$ to the closed interval $[0, R_{\boldsymbol{A}}]$ ensure the O-bound is uniform. Letting $\rho = \beta/\alpha$, $\rho' = \beta'/\alpha'$, and $\rho'' = (\beta+\beta')/(\alpha+\alpha')$, gives

$$\rho = \rho'' - \frac{1}{\alpha(\alpha+\alpha')}, \qquad \rho' = \rho'' + \frac{1}{\alpha'(\alpha+\alpha')},$$

and $0 \le \rho < \rho'' < \rho' \le R_{\boldsymbol{A}}$. Expanding (5.26) about ρ'' gives (uniformly)

$$\begin{aligned} &\alpha F(\rho) + \alpha' F(\rho') - (\alpha+\alpha')F(\rho'') \\ &\quad = \alpha\left(F(\rho'') - \frac{\rho''}{a_3F(\rho'')}\frac{1}{\alpha(\alpha+\alpha')} + \frac{1}{2a_1a_3F^3(\rho'')}\frac{1}{\alpha^2(\alpha+\alpha')^2}\right) \\ &\qquad + \alpha'\left(F(\rho'') + \frac{\rho''}{a_3F(\rho'')}\frac{1}{\alpha'(\alpha+\alpha')} + \frac{1}{2a_1a_3F^3(\rho'')}\frac{1}{\alpha'^2(\alpha+\alpha')^2}\right) \\ &\qquad - (\alpha+\alpha')F(\rho'') + O\left(\frac{\alpha}{\alpha^3(\alpha+\alpha')^3} + \frac{\alpha'}{\alpha'^3(\alpha+\alpha')^3}\right) \end{aligned}$$

which is, after considerable cancellation and some simplification,

$$\begin{aligned} &= \frac{1}{2a_1a_3F^3(\rho'')}\left(\frac{\alpha}{\alpha^2(\alpha+\alpha')^2} + \frac{\alpha'}{\alpha'^2(\alpha+\alpha')^2}\right) \\ &\qquad + O\left(\frac{\alpha}{\alpha^3(\alpha+\alpha')^3} + \frac{\alpha'}{\alpha'^3(\alpha+\alpha')^3}\right) \end{aligned} \tag{5.27}$$

$$\ll \frac{1}{\alpha\alpha'(\alpha+\alpha')}. \tag{5.28}$$

Summing (5.28), and then noting that $\alpha+\alpha' > r$ (Lemma 5.6) and collapsing the resulting telescoping sum, gives

$$\sum_{\substack{\beta/\alpha\in\mathcal{F}(r)\\ \beta/\alpha<R_{\boldsymbol{A}}}} [0 \quad 1](\boldsymbol{c}''-\boldsymbol{c})+[1 \quad 0](\boldsymbol{c}'-\boldsymbol{c}'') \ll \sum_{\substack{\beta/\alpha\in\mathcal{F}(r)\\ \beta/\alpha<R_{\boldsymbol{A}}}} \frac{1}{\alpha\alpha'(\alpha+\alpha')}$$

$$< \frac{1}{r}\sum_{\beta/\alpha\in\mathcal{F}(r)} \frac{1}{\alpha\alpha'} = \frac{1}{r}\sum_{\beta/\alpha\in\mathcal{F}(r)} \left(\frac{\beta'}{\alpha'}-\frac{\beta}{\alpha}\right) = 1/r. \qquad \blacksquare$$

Lemma 5.17 *We have* $C_3(r;\boldsymbol{A}) \ll r^3$.

Proof: This bound is easy to obtain, since $0 \le \beta \le \alpha \le r$ gives

$$\sqrt{(Q_{\boldsymbol{A}}([\pm\beta \quad \alpha]^t))/(a_1a_3)} = \sqrt{(a_1(\pm\beta)^2+a_3\alpha^2)/(a_1a_3)} \ll r,$$

and since the number of summands is $\ll |\mathcal{F}(r)| \ll r^2$. $\blacksquare$

Remarks Table 5.5 on page 114 shows values of $C_3(r;\boldsymbol{A})$ and of $C_{-1}(r;\boldsymbol{A})$ for our four quadratic forms and for several values of r. This table suggests that $C_3(r;\boldsymbol{A}) \sim c_3\, r^3$ as $r\to\infty$, for some constant c_3 depending on $\boldsymbol{A}$. We can prove this and calculate c_3 explicitly by using Möbius inversion on Formula (5.23) and estimating sums by integrals. For example, when $Q_{\boldsymbol{A}}(\boldsymbol{u}) = u_1^2+u_2^2$ we find that

$$c_3 = \frac{2}{3\zeta(2)}\int_0^{\pi/4} \sec^3(\theta)\,d\theta = \frac{\sqrt{2}+\ln\tan(3\pi/8)}{3\zeta(2)} \approx 0.4651832\ldots.$$

Similarly, and confirming the trend apparent in Table 5.5, Alexandru Zaharescu reports in a personal communication that one may show that $C_{-1}(r;\boldsymbol{A}) \sim c_{-1}r^{-1}$ for some c_{-1} depending on $\boldsymbol{A}$. Zaharescu has found that when $Q_{\boldsymbol{A}}(\boldsymbol{u}) = u_1^2+u_2^2$ we have $c_{-1} = 3\sqrt{2}\ln(2)/\pi^2 \approx 0.2979627\ldots$. The techniques used by Zaharescu were developed in the series of papers [BCZ01, ABCZ01, BCZ00, BGZ03].

The values in Table 5.5 were computed using 64-bit floating point arithmetic. The central column was found using Equation (5.25), which is badly behaved numerically due to a large amount of cancellation. This accounts for the irregular (or nonsensical) values in the central column when $r = 100000$. For large r, the right hand columns in Table 5.5 give a more satisfactory estimate of $C_{-1}(r;\boldsymbol{A})$. These values were found using:

$$\begin{aligned}(5.29)\qquad & C_{-1}(r;\boldsymbol{A}) \\ & = O(1/r^2) + \sum_{\substack{\beta/\alpha\in\mathcal{F}(r)\\ \beta/\alpha<R_{\boldsymbol{A}}}} \frac{\operatorname{sgn}(a_1a_3)}{2a_1a_3F^3((\beta+\beta')/(\alpha+\alpha'))}\frac{1}{\alpha\alpha'(\alpha+\alpha')},\end{aligned}$$

which follows from Equations (5.25) and (5.27). The $O(1/r^2)$ error term comes from summing the error term in (5.27), which gives

$$\begin{aligned}&\sum_{\substack{\beta/\alpha\in\mathcal{F}(r)\\ \beta/\alpha<R_{\boldsymbol{A}}}} \frac{\alpha}{\alpha^3(\alpha+\alpha')^3}+\frac{\alpha'}{\alpha'^3(\alpha+\alpha')^3} \ll \frac{1}{r^3}\sum_{\beta/\alpha\in\mathcal{F}(r)}(1/\alpha^2+1/\alpha'^2)\\ &\quad \ll \frac{1}{r^3}\sum_{\beta/\alpha\in\mathcal{F}(r)}(1/\alpha+1/\alpha') = \frac{1}{r^3}\sum_{\beta/\alpha\in\mathcal{F}(r)}\frac{\alpha+\alpha'}{\alpha\alpha'} \ll 1/r^2. \qquad \square\end{aligned}$$

We now conclude our analysis of the number of scanlines enumerated by Algorithm 5.2.

Theorem 5.18 *Let $\mathcal{K}_3$ denote the number of scanlines enumerated by Algorithm 5.2. Then*

$$\mathcal{K}_3 \ll r^3x_2^{-1/2}(x_2-x_1)+r^2+x_2^{1/2}/r. \tag{5.30}$$

Proof: The result follows from Equation (5.24) on page 110 and from Lemmas 5.16 and 5.17. ■

We can now bound the total number of operations performed by Algorithm 5.2, as well as its memory requirements:

Theorem 5.19 *Let $\mathcal{K}$ denote the number of operations performed by Algorithm 5.2. Then*

$$\mathcal{K} \ll x_2^{1/3}+(1+r^3x_2^{-1/2})(x_2-x_1)+r^2+x_2^{1/2}/r. \tag{5.31}$$

Furthermore, Algorithm 5.2 uses $O(\ln(x_2)+x_2-x_1)$ bits of memory.

Proof: The bound on memory usage is obvious since the bit vector `Pset` requires $O(\ln(x_2)+x_2-x_1)$ bits of memory, while all other quantities require $O(\ln(x_2))$ bits.

r	$r^{-3}C_3(r)$	$r\,C_{-1}(r)$	approx. $r\,C_{-1}(r)$
10	0.49613153	0.29047367	0.29094833
100	0.46619902	0.29777181	0.29778011
1000	0.46570740	0.29785074	0.29785085
10000	0.46520914	0.29798864	0.29795720
100000	0.46518671	0.46601781	0.29796200

(a) $Q_{\boldsymbol{A}}(\boldsymbol{u}) = u_1^2 + u_2^2$

r	$r^{-3}C_3(r)$	$r\,C_{-1}(r)$	approx. $r\,C_{-1}(r)$
10	0.45623773	0.11858174	0.11875558
100	0.42772269	0.12156555	0.12156814
1000	0.42727143	0.12159609	0.12159711
10000	0.42681429	0.14473659	0.12164052
100000	0.42679371	−136.21091493	0.12164248

(b) $Q_{\boldsymbol{A}}(\boldsymbol{u}) = u_1^2 + 3u_2^2$

r	$r^{-3}C_3(r)$	$r\,C_{-1}(r)$	approx. $r\,C_{-1}(r)$
10	0.34323070	0.35529676	0.35480283
100	0.32365145	0.36466715	0.36467333
1000	0.32331229	0.36479088	0.36479094
10000	0.32296640	0.36875505	0.36492160
100000	0.32295083	−20.64236905	0.36492743

(c) $Q_{\boldsymbol{A}}(\boldsymbol{u}) = 3u_1^2 + u_2^2$

r	$r^{-3}C_3(r)$	$r\,C_{-1}(r)$	approx. $r\,C_{-1}(r)$
10	0.08143222	0.28392399	0.28455475
100	0.07372946	0.29753753	0.29751630
1000	0.07350154	0.29784492	0.29784457
10000	0.07342030	0.29323661	0.29795719
100000	0.07341681	29.09729592	0.29796199

(d) $Q_{\boldsymbol{A}}(\boldsymbol{u}) = 3u_1^2 - u_2^2$

Table 5.1: Numerical approximations of sums related to scanline analysis. “Approximate” values for $r\,C_{-1}(r)$ computed using the expansion (5.29) on the preceding page, which, for large r, is better behaved numerically than the defining expression (5.22) on page 110. (See the remarks following Lemma 5.17.)

As noted at the beginning of this section, on page 103, $\mathcal{K} \ll \sum_{j=1}^{5} \mathcal{K}_j$, where the quantities $\mathcal{K}_j$ are also defined on page 103. We began by noting that $\mathcal{K}_1 = 1 + x_2 - x_1$. In Theorem 5.14 we showed that $\mathcal{K}_2 \ll x_2 - x_1 + x_2^{1/3}$, and in Theorem 5.15 we showed that $\mathcal{K}_5 \ll x_2 - x_1 + x_2^{1/3}$. The number of pieces enumerated by Algorithm 5.2, $\mathcal{K}_4$, satisfies $\mathcal{K}_4 \ll |\mathcal{F}(r)| \ll r^2$, by Lemma 5.6. Theorem 5.18 established $\mathcal{K}_3 \ll r^3 x_2^{-1/2}(x_2 - x_1) + r^2 + x_2^{1/2}/r$, and the result follows upon summing these bounds. ■

We now determine the order of our Farey dissection, r, so that Algorithm 5.2 performs $O(x_2 - x_1 + x_2^{1/3})$ operations.

Theorem 5.20 *Let $\mathcal{K}$ denote the number of operations performed by Algorithm 5.2. Provided r satisfies*

$$r \asymp \begin{cases} x_2^{1/4}(x_2 - x_1)^{-1/4} & \text{if } x_2 - x_1 \gg x_2^{1/3}, \\ x_2^{1/6} & \text{if } x_2 - x_1 \ll x_2^{1/3}, \end{cases}$$

we have $\mathcal{K} \ll x_2 - x_1 + x_2^{1/3}$.

Proof: In the first case, where $x_2 - x_1 \gg x_2^{1/3}$, we see that

$$r \ll x_2^{1/4}(x_2 - x_1)^{-1/4} \ll x_2^{1/6}. \tag{5.32}$$

Using (5.32) in the bound (5.31), we find that

$$\mathcal{K} \ll x_2^{1/3} + x_2 - x_1 + x_2^{1/2}/r. \tag{5.33}$$

Also in this case, $x_2 - x_1 \gg x_2^{1/3}$ implies $x_2^{1/4} \ll (x_2 - x_1)^{3/4}$, so we have $x_2^{1/2}/r \ll x_2^{1/4}(x_2 - x_1)^{1/4} \ll x_2 - x_1$, thus (5.33) gives $\mathcal{K} \ll x_2 - x_1 + x_2^{1/3}$.

In the second case, where $x_2 - x_1 \ll x_2^{1/3}$, the result follows from an easy computation using $r \asymp x_2^{1/6}$ and the bound (5.31). ■

Algorithm 5.5, below, presents the segmented version of the dissected sieve, using subintervals of size bounded by B, and with the appropriate value of r for each subinterval. By the bound (4.1) on page 84 we see that the segmented version requires $O((x_2 - x_1)(1 + x_2^{1/3}/B) + x_2^{1/3})$ operations to sieve $[x_1, x_2]$.

Algorithm 5.5 (`SegmentedDissectedSieve`)
Enumerate the primes in the interval $[x_1, x_2]$, $3 \leq x_1 \leq x_2$, working over subintervals of size $\leq B$. We choose the "order of dissection", r, to be roughly optimal, as in the discussion above.

```
SegmentedDissectedSieve(B ∈ ℕ, x1, x2) {
 assert 3 < x1 ≤ x2;
 Pset ← AllocateBitVector(B);
 for (Pset.x1 ← ⌈x1⌉; Pset.x1 ≤ x2; Pset.x1 ← Pset.x2 + 1) {
  Pset.x2 ← min(x2, Pset.x1 + B − 1);
   if (Pset.x2 − Pset.x1 < (Pset.x2)^(1/3))
    r ← (Pset.x2)^(1/6);
   else
    r ← (Pset.x2/(Pset.x2 − Pset.x1))^(1/4);
   DissectedSieve(max(3, r), Pset);
   for (n ← Pset.x1; n ≤ Pset.x2; n++)
    if (Pset[n] = 1)      Enumerate(n);
 }
 return k;}
```

5.6 Implementation notes

We have tested Algorithm 5.5 (`SegmentedDissectedSieve`) with a C-language implementation `dsieve`.

Although this is a rudimentary implementation, we made some optimizations. We take advantage of the fact that $|\det(\boldsymbol{T})| = 1$. We reduce the size of the numbers used in Algorithm 5.3 (`ScanPiece`) by working both in $\boldsymbol{u}$-space and in a coordinate system in which $\boldsymbol{v}$-space is re-centered around the lattice point given by rounding the components of $\sqrt{x_1}\, \boldsymbol{c}''$ to their nearest integers. We also get smaller numbers by computing $Q_{\boldsymbol{A}}(\boldsymbol{u}) - x_1$, rather than $Q_{\boldsymbol{A}}(\boldsymbol{u})$.

As in the Atkin-Bernstein paper, `ScanPiece` uses Equation (5.4) to reduce the number of multiplications needed to update $Q_{\boldsymbol{A}}(\boldsymbol{u}) - x_1$ as $\boldsymbol{u}$ varies. It reduces the number of square root operations by testing the values of $Q_{\boldsymbol{A}}(\boldsymbol{u}) - x_1$, $\langle \boldsymbol{\tau}, \boldsymbol{u} \rangle_{\boldsymbol{A}}$, and $\langle \boldsymbol{\tau}', \boldsymbol{u} \rangle_{\boldsymbol{A}}$, to decide whether a point lies within a piece and when to move to the next scanline.

Algorithm 5.4 (`SquareFreeSieve`) was modified to sieve only the odd numbers, and an additional parameter was added to control the point at which it switches from the loop of lines 3–5 to the loop of lines 6–8. The optimal transition point was found experimentally to be near $q = 1.5x_2^{1/3}$. It should be noted that the constant 1.5 is certainly dependent on the details of our implementation, and is likely to be machine dependent.

We also found experimentally, for a wide range of values of x_2 and B, that setting $r = \lfloor 0.5 + 0.7(x_2/B)^{1/4} \rfloor$ approximately minimizes both the number of scanlines and the operation count for Algorithm 5.2 (`DissectedSieve`). Again, the constant 0.7 is both implementation and machine dependent.

5.7 Possible improvements

There are several ways in which `dsieve` could be improved. For simplicity in coding and analysis, we used a single Farey dissection. It may be more efficient to use three Farey dissections, each of an order chosen to optimize the corresponding case in Theorem 5.1. Also, "Farey-like" sequences tailored to each quadratic form may be more efficient—Sierpiński used a sequence of the form

$$\{\beta/\alpha : \ \gcd(\alpha, \beta) = 1, \ \alpha^2 + \beta^2 \le r^2\}$$

in his work on the circle problem [Sie74]. Furthermore, besides the three pairs of congruence classes and quadratic forms used in Theorem 5.1, there are other choices—see the paper of Atkin and Bernstein for one example [AB02]. How to determine an optimal set of quadratic forms seems to be an open question.

Currently, `dsieve` enumerates too many points. Although it does not allocate storage for even indices, it does enumerate all points within a swath, including points yielding $Q_{\boldsymbol{A}}(\boldsymbol{u}) \equiv 0 \pmod 2$. Reducing the number of points enumerated, and avoiding the costly test "$n \bmod m = k$" used in Algorithm 5.3 (`ScanPiece`) could improve the speed. However, this will not reduce the number of scanlines enumerated—which may dominate the operation count when $x_2 - x_1$ is small enough in comparison with $x_2^{1/3}$. Also, the value of $\boldsymbol{T} \pmod m$ determines the periodic pattern of remainders taken by $n = Q_{\boldsymbol{A}}(\boldsymbol{u}) \pmod m$ as $\boldsymbol{u}$ moves along a scanline. Since the congruence class of $\boldsymbol{T} = [\boldsymbol{\tau} \ \ \boldsymbol{\tau}']$ changes irregularly between calls to `ScanPiece`, it may be preferable to restrict $\boldsymbol{T}$ to a limited set of congruence classes.

We have not carefully bounded the size of numbers used by Algorithms 5.2, 5.3, and 5.4. The implementation `dsieve` works with a mixture of 32-bit and 64-bit integers, and 64-bit floating point numbers, and becomes unreliable near 10^{18}.

5.8 Timing data

Bernstein has implemented the Atkin-Bernstein sieve and posted it on the web as a package of routines, "`primegen 0.97`"; see the paper by Atkin and Bernstein for the URL [AB02]. Tables 5.2 and 5.3 show running times for `primegen` and for `dsieve`. Both programs were compiled to run on SUN SPARC computers, using the Gnu C-compiler (`gcc`) version 2.8.1, with compilation options `-O3 -mcpu=v8`. Times were measured on a 300 MHz UltraSPARC 5/10 with 64 megabytes of "main" memory. In addition, it has roughly 16 kilobytes of "level-1" cache memory—very fast compared to main memory—and 512 kilobytes of somewhat slower level-2 cache. (The amounts and speeds of cache were estimated using a C implementation of the `mem1d` program given in [DS98, Appendix E].)

	time (seconds)			
x_1	$B \approx 2.05 \cdot 10^6$	$B = 2^{24}$	$B = 2^{26}$	$B = 2^{28}$
10^9	20	27	73	156
10^{10}	26	27	75	194
10^{11}	49	30	72	204
10^{12}	121	38	72	204
10^{13}	340	67	75	207
10^{14}	1035	157	96	215
10^{15}	3231	438	174	241
10^{16}	11716	1527	491	362
10^{17}	74909	9142	2891	1313

Table 5.2: Time for Bernstein's `primegen` program to count primes in the interval $[x_1, x_1 + 10^9]$, using bit vector of size B

The program `primes.c` provided in the `primegen` package was modified to print the count of primes in an interval $[x_1, x_2]$, and was run to count primes in several intervals. These counts were compared with those found by `dsieve`, and by a third program based on Robert Bennion's "hopping sieve" [Gal98]. Although Bernstein warns that the `primegen` code is not valid past $x = 10^{15}$, all programs returned the same counts except for the interval

$[10^{17}, 10^{17} + 10^9]$, where `primegen` counts $10^{17} + 111377247 = 7 \cdot 119522861^2$ and $10^{17} + 158245891 = 11 \cdot 95346259^2$ as primes.

The buffer size used by `primegen` (B, in our notation) is set at compile time. Table 5.2 shows running times for `primegen` to count primes in the interval $[x_1, x_2 = x_1 + 10^9]$ for several combinations of B and x_1. In Bernstein's installation notes he suggests choosing B so that data used by the inner loop of the algorithm resides in level-1 cache. (The inner loop treats $n = 60k + d$ for fixed d, where d takes one of the 16 values relatively prime to 60.) Thus, for our smallest value of B, we used $B = 16 \cdot 128128 \approx 2.05 \cdot 10^6$, as Bernstein's notes suggest for UltraSPARC computers.

To avoid having a runtime that became linear in x_1 for very large x_1 (linear with a small O-constant), we modified the routine `primegen_skipto` to use division instead of repeated subtraction in its calculation of a quotient.

As well as showing that `primegen` slows as $\sqrt{x_2}$ grows larger than B, Table 5.2 illustrates that the operation count can be a poor predictor of the running time on a computer with cache memory. Increasing B reduces the operation count for sieving an interval, but also increases the chance that memory references will miss the level-1 cache. This slowdown as the locality of memory references decreases can be striking. On the computer used for these tests, widely scattered references to "main" memory were measured to be roughly 20 times slower than references to level-1 cache. An informative discussion of cache memory is given in [DS98, Chapter 3]. A more scholarly treatment may be found in [HP90, Chapter 5].

Table 5.3 shows running times for `dsieve` to count primes in the interval $[x_1, x_2 = x_1 + 10^9]$, using two different values of B, with B depending on x_2. The entries with $B \approx 10x_2^{1/3}$ illustrate the running time when using a "small" amount of memory, while the entries with $B \approx x_2^{1/2}$ show the running time with memory usage comparable to that needed for efficient operation of previously known sieves. In both cases and for all values of x_1, after computing `Pset` it took roughly 18 seconds to count the primes (the number of 1-bits in `Pset`). As expected, the running time does not greatly increase as x_1 increases. The slowdown for larger x_1 is presumably due in part to decreasing locality of reference, although more detailed statistics on operation counts should be collected to better understand these results.

	$B \approx 10x_2^{1/3}$		time (sec.)		$B \approx x_2^{1/2}$		time (sec.)	
x_1	B	r	sqfree	total	B	r	sqfree	total
10^9	$1.26 \cdot 10^4$	14	51	391	$4.47 \cdot 10^4$	10	25	333
10^{10}	$2.22 \cdot 10^4$	19	55	402	$1.05 \cdot 10^5$	13	23	329
10^{11}	$4.66 \cdot 10^4$	27	57	403	$3.18 \cdot 10^5$	17	22	334
10^{12}	$1.00 \cdot 10^5$	39	59	407	$1.00 \cdot 10^6$	22	20	334
10^{13}	$2.15 \cdot 10^5$	59	59	416	$3.16 \cdot 10^6$	30	18	339
10^{14}	$4.64 \cdot 10^5$	86	61	423	$1.00 \cdot 10^7$	39	20	414
10^{15}	$1.00 \cdot 10^6$	125	61	428	$3.16 \cdot 10^7$	52	20	453
10^{16}	$2.15 \cdot 10^6$	183	61	437	$1.00 \cdot 10^8$	70	19	456
10^{17}	$4.64 \cdot 10^6$	268	63	465	$3.16 \cdot 10^8$	93	19	466

Table 5.3: Time for `dsieve` (our implementation of Algorithm 5.5) to count primes in the interval $[x_1, x_2 = x_1 + 10^9]$, using a bit vector of size $\approx B$ and a Farey dissection of order $r \approx 0.7(x_2/B)^{1/4}$. The "sqfree" column gives the time required to sieve out square factors.

5.9 Miscellaneous remarks

The ideas of this chapter can also be used with $Q_{\boldsymbol{A}}(\boldsymbol{u}) = u_1u_2$, giving a dissected "Eratosthenes-like". sieve. This corresponds to the Dirichlet divisor problem, which is concerned with estimating the number, $D(x)$, of lattice points within the hyperbola $u_1u_2 \le x$, $u_1 > 0$, $u_2 > 0$. Voronoï [Vor03] used a dissection based on the Farey-like sequence

$$\{\beta/\alpha : \ \gcd(\alpha, \beta) = 1, \ \alpha\beta \le t\},$$

with $t = x^{1/3}$, to show that $D(x) = x\ln(x) + (2\gamma - 1)x + O(x^{1/3}\ln x)$, where $\gamma \approx 0.5772\ldots$ is Euler's constant. His result was an improvement of an earlier result of Dirichlet, who used the "hyperbola method" to get an error term of $O(x^{1/2})$ instead of $O(x^{1/3+\epsilon})$ for the approximation of $D(x)$. Voronoï's result for the Dirichlet divisor problem suggests that a dissected Eratosthenes-like sieve would require $O(x_2^{1/3+\epsilon})$ bits and $O(x_2^{\epsilon}(x_2 - x_1 + x_2^{1/3}))$ operations to sieve the interval $[x_1, x_2]$.

We can also use dissection to improve some factoring algorithms. For example, trial division searches for a solution to $n = Q_{\boldsymbol{A}}(\boldsymbol{u}) = u_1u_2$, and dissection would reduce the operation count to $O(n^{1/3+\epsilon})$. If $x_2 - x_1 \gg x_2^{1/3}$, it should be possible to factor the numbers n in an interval $x_1 \le n \le x_2$ using an average of $O(x_2^\epsilon)$ operations per integer. Dissection would similarly reduce the number of operations needed to solve the quadratics used by Derrick and Emma Lehmer [LL74] for factoring.

Sierpiński's $O(x^{1/3})$ bound for the circle problem, i.e., for the difference between the number of lattice points within a circle of radius $\sqrt{x}$ and its area, has since been improved to an $O(x^{35/108+\epsilon})$ bound [Ivi85, §13.8], and it is conjectured that the bound can be reduced to $O(x^{1/4+\epsilon})$. Ivic's improved bound was proven using analytic techniques, and it is not clear if these techniques could be applied to making sieving more efficient. However, the result does raise the intriguing possibility that some variation on the dissected sieve might sieve efficiently over intervals $[x_1, x_2]$ of length x_2^C, with C significantly less than 1/3.

5.10 Acknowledgments

While reading Hugh Williams' book on primality testing [Wil98] I was struck by the rich variety of quadratic forms that are related to primality testing, and the fact that some of these forms differ only by a linear change of variables. Also, Dan Bernstein had brought my attention to the Atkin-Bernstein sieving algorithm. These ideas led me to the idea of dissecting a region bounded by conics, using a change of variables appropriate to scanning each piece in a direction roughly tangent to the boundary curves.

I began to discuss with my colleagues the problem of how best to dissect a region, and Bruce Berndt pointed out the paper by Voronoï [Vor03] which then led to the material needed to design the dissected sieve.

Maarten Bergvelt, Martin Huxley, Ulrike Vorhauer, and Harold G. Diamond all provided help in translating and interpreting van der Corput's paper *Über Gitterpunkte in der Ebene* [vdC20].

Jared Bronski, at the University of Illinois Urbana-Champaign, made his SUN workstation available for the timing benchmarks documented in Tables 5.2 and 5.3.

The results in this chapter, without proof, were previously summarized in [Gal00]. I thank Gilbert Gosseyn for noting an error in the version of Algorithm 5.1 that was given in that paper.

6 Enumerating Primes with a Hybrid Sieve

6.1 Introduction: sieving using probable primes

In this chapter we give an algorithm which, provided $x_2 - x_1 \geq \sqrt{x_2}$, enumerates primes in the interval $[x_1, x_2]$ using $O(\ln\ln(x_2)(x_2 - x_1))$ arithmetic operations on numbers of $O(\ln x_2)$ bits.

We conjecture that the algorithm requires $O(x_2^{1/4})$ bits of memory, and provide arguments to support this belief. As in the previous chapter, we will make use of the terminology and analyses of Section 4.2.

Before presenting this "hybrid" method, we begin by introducing some additional terminology:

Definition 6.1 We say $n \in \mathbb{N}$ is *y-unsieved* if all $\ell \mid n$, ℓ prime, satisfy $\ell > y$. We write $\mathcal{U}_y$ for the set of y-unsieved numbers. Conversely, if there is a prime ℓ, $\ell \mid n$, $\ell \leq y$, then we say that n is *y-sieved*. □

Remarks Note that the number 1 is y-unsieved for any value of y.

The y-unsieved numbers are those numbers which remain after sieving out all multiples of primes $\ell \leq y$. The term *y-unsieved* is analogous to the term *y-smooth* for numbers free of "large" prime factors. The latter term is quite standard [Rie94, Chapter 5], while the terms *y-unsieved* (for numbers free of "small" prime factors) and *y-sieved* are not standard.

The distributions of both y-smooth and y-unsieved numbers have been analyzed using related techniques [Ten95, Part III, Chapters 5 and 6]. Much of the work on the distribution of y-unsieved numbers dates to Buchstab [Buc37], and is related to the Meissel-Lehmer algorithm for computing $\pi(x)$. (A brief historical summary of this work may be found in the end-notes to Chapter 2 of the book *Sieve Methods* by Halberstam and Richert [HR74, Note 2.1].) □

Definition 6.2 We say that $n \in \mathbb{N}$ is a Fermat *probable prime* to the base 2, or, more succinctly, that "n is a probable prime", when $2^{n-1} \equiv 1 \pmod{n}$. We say that n is a *pseudoprime* if it is a composite probable prime. We write $\mathrm{prp}(n)$ or $\mathrm{psp}(n)$ to abbreviate the statements that n is a probable prime or pseudoprime, respectively. □

Remarks Some sources use the term *pseudoprime* to mean what we call a *probable prime*, and other sources define n to be a pseudoprime to the base a if n is a composite satisfying $a^n \equiv a \pmod{n}$, but the terminology given here has become quite standard [PSW80], [Rie94, Chapter 4].

A prp-*test*, i.e., a test of $2^{n-1} \equiv 1 \pmod{n}$, requires $O(\ln n)$ arithmetic operations, using numbers of $O(\ln n)$ bits [Rie94, Chapter 4]. Fermat's "little theorem" states that if p is prime and $\gcd(p, a) = 1$ then $a^{p-1} \equiv 1 \pmod{p}$, so we see that odd primes are probable primes. Conversely, pseudoprimes (composite probable primes) are quite rare [Pom81], so a prp-test serves to eliminate most composite numbers from consideration as possible primes. □

In rough outline, our hybrid sieve works as follows. During an initial phase we generate a set $\mathcal{C} \subseteq [x_1, x_2]$ of "candidate" pseudoprimes which includes all y-unsieved pseudoprimes in the interval $[x_1, x_2]$. (The set $\mathcal{C}$ may also include "spurious" candidates, i.e., y-sieved pseudoprimes.) During the second, main phase, we use a sieve to enumerate all y-unsieved $n \in [x_1, x_2]$, and then enumerate n as a prime if it satisfies the two conditions $\text{prp}(n)$ and $n \notin \mathcal{C}$. This works if $y < x_1$, since the primes in $[x_1, x_2]$ coincide with those y-unsieved $n \in [x_1, x_2]$ that satisfy $\text{prp}(n)$ and $n \notin \mathcal{C}$. (The restriction $y < x_1$ is a minor technical point—required since primes $p \leq y$ are y-sieved.) For reasons discussed in the following sections a choice of $y \geq x_2^{1/4}$ seems best for this method.

Eliminating y-sieved numbers lets us reduce the number of (expensive) prp-tests. If we have B bits of memory available for sieving, then sieving is efficient provided we choose $y \leq B$, roughly speaking. Although a number n may be y-unsieved (free of small prime factors) and still satisfy $\text{psp}(n)$, the condition that $n \notin \mathcal{C}$ prevents n from being mistaken as a prime. The low memory requirements of this hybrid approach depends on the unproven observation that, at least for $y \geq x_2^{1/4}$, the set $\mathcal{C}$ has few entries.

6.2 Initial phase: finding unsieved pseudoprimes

Turning to the problem of finding our set $\mathcal{C}$, we begin by introducing some further notation. Throughout this section $\omega(n)$ denotes the number of distinct prime divisors of n and $\Omega(n)$ denotes the total number of prime divisors of n. For example, $\omega(12) = 2$ and $\Omega(12) = 3$. We write $\nu(\ell)$ for the order of

2 in the multiplicative group of integers modulo ℓ.

To find the set $\mathcal{C}$ we use Algorithm 6.1 (`CandidatePSPs`), below. This algorithm is roughly based on a sieving technique described by Richard Pinch [Pin00, §7]. This technique depends on the fact that if $\operatorname{psp}(n)$, and $\ell \mid n$, then $n \equiv \ell \pmod{\ell\,\nu(\ell)}$. (Proofs of this fact are given in [Pin00, Proposition 1] and [PSW80, Proposition 3].)

Algorithm 6.1 (`CandidatePSPs`: Find y-unsieved pseudoprimes)
Given $y > 0$ and $1 \le x_1 \le x_2$, this routine returns a set $\mathcal{C}$ of pseudoprimes satisfying $[x_1, x_2] \cap \mathcal{U}_y \subseteq \mathcal{C} \subseteq [x_1, x_2]$.

```
CandidatePSPs(y, x_1, x_2) {
 assert y > 0;   assert 1 ≤ x_1 ≤ x_2;
 C ← {};
 for (ℓ ∈ [⌊y + 1⌋, √x_2] ∩ P)
  for (n ← ℓ + ℓ ν(ℓ) max(1, ⌈(x_1 − ℓ)/(ℓ ν(ℓ))⌉); n ≤ x_2; n ← n + ℓ ν(ℓ))
   if (2^(n−1) mod n = 1)
      C ← C ∪ {n};
 return C;}
```

Before analyzing the storage needs and operation count for Algorithm 6.1, in Theorem 6.3 we confirm that the set $\mathcal{C}$ serves our needs:

Theorem 6.3 *Let $\mathcal{C}$ be the set returned by* `CandidatePSPs(`y`,`x_1`,`x_2`)`. *Provided $0 < y < x_1 \le x_2$ and given $n \in \mathbb{N} \cap [x_1, x_2]$, we have n is an odd prime if and only if $n \in \mathcal{U}_y$, $\operatorname{prp}(n)$, and $n \notin \mathcal{C}$.*

Proof: First assume that n is an odd prime. Then, by Fermat's little theorem, $\operatorname{prp}(n)$. Since n is prime and $y < x_1$ we have $n \in \mathcal{U}_y$. Note that $n \in \mathcal{C}$ has the form $n = \ell(1 + k\nu(\ell))$, $k > 0$, so $n \in \mathcal{C}$ is composite. Since we assume n is prime, we have $n \notin \mathcal{C}$.

Conversely, if $n = 2$ then $\operatorname{prp}(n)$ is false. If n is composite, $n \in \mathcal{U}_y$, and $\operatorname{prp}(n)$, then there is a prime $\ell \mid n$ with $y < \ell \le \sqrt{n} \le \sqrt{x_2}$ and so $n \in \mathcal{C}$. ■

Our cost analysis of Algorithm 6.1 is based on the costs of computing key values enumerated during its execution. These are: the primes ℓ; the factorizations of $\ell - 1$; the values $\nu(\ell)$; the values $2^{n-1} \bmod n$; and the set $\mathcal{C}$ (which is "enumerated" as elements are placed into it). We are interested in the factorizations of $\ell - 1$ since, as explained below, they are used in the computation of $\nu(\ell)$.

With this in mind, in our analysis of Algorithm 6.1 we break the storage requirement $\mathcal{B}$ and operation count $\mathcal{K}$ into five parts:

$$\mathcal{B} = \mathcal{B}_1 + \mathcal{B}_2 + \mathcal{B}_3 + \mathcal{B}_4 + \mathcal{B}_5,$$
$$\mathcal{K} = \mathcal{K}_1 + \mathcal{K}_2 + \mathcal{K}_3 + \mathcal{K}_4 + \mathcal{K}_5,$$

where $\mathcal{B}_1$, $\mathcal{K}_1$ are the cost of enumerating primes ℓ, $y < \ell \leq \sqrt{x_2}$; $\mathcal{B}_2$, $\mathcal{K}_2$ the cost of finding the factors of all $\ell - 1$; $\mathcal{B}_3$, $\mathcal{K}_3$ the cost of computing all $\nu(\ell)$; $\mathcal{B}_4$, $\mathcal{K}_4$ the cost of computing all values of $2^{n-1} \bmod n$; and $\mathcal{B}_5$, $\mathcal{K}_5$ the total cost of placing elements into $\mathcal{C}$. Although it will not influence our cost analysis, note that the same value may be enumerated several times. For example, if n has prime factors ℓ_1, ℓ_2, with $y < \ell_1 < \ell_2 \leq \sqrt{x_2}$, then that value of n will be enumerated more than once by the `for` loop at line 5.

The primes ℓ may be enumerated using thesieve of Eratosthenes, requiring $\mathcal{B}_1 = O(x_2^{1/4})$ bits and $\mathcal{K}_1 = O(\ln\ln(x_2)\sqrt{x_2})$ operations. The complete prime factorizations of all m, $y < m \leq \sqrt{x_2} + 1$ can also be found using a sieve, using $O(x_2^{1/4})$ bits and $O(\ln\ln(x_2)\sqrt{x_2})$ operations [Gal98, §6]. Since this includes the factorizations of $m = \ell - 1$, this gives $\mathcal{B}_2 = O(x_2^{1/4})$ bits and $\mathcal{K}_2 = O(\ln\ln(x_2)\sqrt{x_2})$.

Given the factorization of $\ell-1$, Algorithm 1.4.3 from [Coh93] will find $\nu(\ell)$ using $O(\ell^{\epsilon})$ bits and $O(\ln(\ell)\Omega(\ell-1))$ operations. Thus $\mathcal{B}_3 = O(x_2^{\epsilon})$. We now turn to analyzing $\mathcal{K}_3$, the cost of computing all $\nu(\ell)$ given the factorizations of $\ell-1$. Our analysis of $\mathcal{K}_3$ culminates in Theorem 6.6 on page 127. The key tool in our analysis is Theorem 6.5, below, which bounds the average value of $\Omega(\ell-1)$ by $O(\ln\ln(x))$, when the average is taken over all primes $\ell \leq x$.

Lemma 6.4 *Let* $F(x) := \sum_{p^m \leq x} 1$. *Then* $F(x) \ll \dfrac{x}{\ln(x)}$, *uniformly for* $x \geq 2$.

Proof: The prime number theorem implies that $\pi(x) \ll x/\ln(x)$ uniformly for $x \geq 2$. Thus

$$F(x) = \sum_{p \leq x} \sum_{m=1}^{\lfloor \ln(x)/\ln(p) \rfloor} 1 \ll \sum_{p \leq \sqrt{x}} \ln(x) + \sum_{\sqrt{x} < p \leq x} 1 \ll \frac{x}{\ln(x)}. \qquad \blacksquare$$

Theorem 6.5 $\sum_{\ell \le x} \Omega(\ell - 1) \ll x \ln\ln(x)/\ln(x)$ *as* $x \to \infty$.

Proof: In the body of this proof let $\pi(x;q,a)$ denote the counting function for primes $\ell \equiv a \bmod q$. Beginning with the definition of $\Omega(n)$, we have

$$\sum_{\ell \le x} \Omega(\ell - 1) = \sum_{\ell \le x} \sum_{\substack{p^m \\ p^m \mid \ell - 1}} 1 = \sum_{p^m < x} \sum_{\substack{\ell \le x \\ \ell \equiv 1 \bmod p^m}} 1 = \sum_{p^m < x} \pi(x; p^m, 1)$$
$$= S_1 + S_2,$$

where $S_1 := \sum_{p^m \le x/2} \pi(x; p^m, 1)$, and $S_2 := \sum_{x/2 < p^m < x} \pi(x; p^m, 1)$.

To bound S_2 we note that $\pi(x; p^m, 1) \le 1$ when $p^m > x/2$, so, using Lemma 6.4,

$$S_2 = \sum_{x/2 < p^m < x} \pi(x; p^m, 1) \le \sum_{p^m < x} 1 \ll x/\ln(x). \tag{6.1}$$

To bound S_1, we use the Brun-Titchmarsh Theorem (see [MV73] or [Ten95, Part I, §4.6, Theorem 9]), which can be stated as

$$\pi(x; q, a) \ll \frac{x}{\varphi(q)\ln(x/q)} \tag{6.2}$$

uniformly for $x/q \ge 2$, where $\varphi(q)$ denotes Euler's totient function:

$$\varphi(q) := \sum_{\substack{1 \le m < q \\ \gcd(m,q) = 1}} 1.$$

Since $\varphi(p^m) = (p-1)p^{m-1} \gg p^m$, the bound (6.2) gives

$$S_1 = \sum_{p^m \le x/2} \pi(x; p^m, 1) \ll x \sum_{p^m \le x/2} \frac{1}{p^m \ln(x/p^m)}. \tag{6.3}$$

Rewriting the sum in the right side of (6.3) as a Stieltjes integral, in terms of the $F(x)$ of Lemma 6.4, gives

$$\sum_{p^m \le x/2} \frac{1}{p^m \ln(x/p^m)} = \int_{2-}^{x/2+} \frac{d\,F(u)}{u \ln(x/u)}. \tag{6.4}$$

We then integrate by parts, noting that $F(u) = 0$ for $u < 2$, while for $u \ge 2$

we have $F(u) \ll u/\ln(u)$. Thus, (6.4) is

$$= \left.\frac{F(u)}{u\ln(x/u)}\right|_{2-}^{x/2+} - \int_2^{x/2} F(u)\frac{d}{du}\frac{1}{u\ln(x/u)}\,du$$

$$= O(1/\ln(x)) + \int_2^{x/2} \frac{F(u)}{u^2\ln(x/u)}\left(1 - \frac{1}{\ln(x/u)}\right)du. \tag{6.5}$$

Again using $F(u) \ll u/\ln(u)$, we find that the integral in (6.5) is

$$\ll \int_2^{x/2} \frac{F(u)}{u^2\ln(x/u)}\,du \ll \int_2^{x/2} \frac{1}{u\ln(u)\ln(x/u)}\,du.$$

Setting $u = e^\tau$ and changing variables, this is

$$= \int_{\ln(2)}^{\ln(x/2)} \frac{1}{\tau(\ln(x)-\tau)}\,d\tau = \frac{1}{\ln(x)}\int_{\ln(2)}^{\ln(x/2)} \frac{1}{\tau} + \frac{1}{\ln(x)-\tau}\,d\tau$$
$$= \frac{2}{\ln(x)}(\ln\ln(x/2) - \ln\ln(2)) \ll \ln\ln(x)/\ln(x).$$

Substituting this bound for the integral in (6.5), and then working back to the right side of (6.3), we find that $S_1 \ll x\ln\ln(x)/\ln(x)$. This dominates our bound (6.1) for S_2, so the result follows. ■

Theorem 6.6 *Let $\mathcal{K}_3$ denote the number of operations required to enumerate all values of $\nu(\ell)$, totaled over all $\ell \in \mathcal{P}$, $y < \ell \le \sqrt{x_2}$, and assuming that the factorization of each $\ell - 1$ is precomputed. Then $\mathcal{K}_3 \ll \ln\ln(x_2)\sqrt{x_2}$, uniformly for $x_2 \ge 3$.*

Proof: As noted above, given the factorization of $\ell - 1$, $\nu(\ell)$ can be computed using $O(\ln(\ell)\Omega(\ell-1))$ operations. Thus, by Theorem 6.5, we find that

$$\mathcal{K}_3 \ll \sum_{y<\ell\le\sqrt{x_2}} \ln(\ell)\Omega(\ell-1) \ll \ln(x_2)\sum_{\ell\le\sqrt{x_2}} \Omega(\ell-1) \ll \ln\ln(x_2)\sqrt{x_2}.$$

This holds uniformly since $\ln\ln(x)$ is bounded away from 0 for $x \ge 3$. ■

Throughout the remainder of this section we let N denote the number of times that line 6 of Algorithm 6.1 is executed. In other words, N is the number of prp-tests performed by the algorithm and N is also the number of enumerations of n, with the convention that repeated enumerations of the same value are treated as distinct. Much of our analysis in the remainder of this section depends on bounds and estimates for N, as given below in Theorem 6.11 and as discussed in Conjecture 6.16. Here, the key tool in our analysis is Lemma 6.9, below, which bounds $\sum_{\ell>y} \frac{1}{\ell\,\nu(\ell)}$.

Lemma 6.7 *Let $S(x) := \sum_{\ell\le x} 1/\nu(\ell)$. Then $S(x) \ll x^{1/2}$ as $x \to \infty$.*

The following proof of Lemma 6.7 is due to Gergely Harcos (personal communication).

Proof: Recall that $\omega(n)$ denotes the number of distinct prime divisors of n, and note that $\omega(n) \ll \ln(n)$. For any $z > 0$ we have

$$\begin{aligned} zS(x) - x &\le \sum_{\ell\le x}\left(\frac{z}{\nu(\ell)} - 1\right) \le \sum_{\ell\le x} \lfloor z/\nu(\ell)\rfloor \\ &= \sum_{\ell\le x}\sum_{\substack{m\le z\\ \nu(\ell)|m}} 1 = \sum_{\ell\le x}\sum_{\substack{m\le z\\ \ell|2^m-1}} 1 = \sum_{m\le z}\sum_{\substack{\ell\le x\\ \ell|2^m-1}} 1 \\ &\le \sum_{m\le z}\omega(2^m-1) \ll \sum_{m\le z} m \ll z^2. \end{aligned} \tag{6.6}$$

It follows that $S(x) \ll z + x/z$. Setting $z = \sqrt{x}$ concludes the proof. ∎

Conjecture 6.8 *In the notation of Lemma 6.7, $S(x) = x^{o(1)}$ as $x \to \infty$.*

Supporting Arguments: We know that, "on average", $\omega(n) = \ln\ln(n)$ [HW79, Theorem 430]. Provided that numbers of the form $2^m - 1$ factor like "average" numbers, we would expect, "on average", that $\omega(2^m - 1) \ll \ln(m)$. Thus, picking up the proof of Lemma 6.7 at (6.6), we conjecture that, for $z > 0$

$$zS(x) - x \le \sum_{m\le z}\omega(2^m-1) \ll \sum_{m\le z}\ln(m) \ll z\ln(z).$$

Setting $z = x$ then gives $S(x) \ll \ln(x) = x^{o(1)}$.

The following argument—which seems to be better supported by some cursory computational tests—suggests that $S(x)$ may be somewhat larger, but still $S(x) = x^{o(1)}$.

Note that $2^m - 1 = \prod_{d|m}\psi_d(2)$, where $\psi_d(w)$ denotes the dth cyclotomic polynomial. Now, the number of divisors of m has average order $\ln(m)$. This suggests that if $\psi_d(2)$ factors like an "average" number then we might expect

$\omega(2^m-1)$ to have average order $\sum_{d|m} \ln\ln(\psi_d(2)) \ll \ln^2(m)$. Arguing as before, this implies

$$zS(x) - x \le \sum_{m\le z} \omega(2^m-1) \ll \sum_{m\le z} \ln^2(m) \ll z\ln^2(z),$$

and setting $z = x$ gives $S(x) \ll \ln^2(x) = x^{o(1)}$. ■

Lemma 6.9 *Let* $T(y) = \displaystyle\sum_{\ell>y} \frac{1}{\ell\,\nu(\ell)}$. *Then* $T(y) \ll y^{-1/2}$ *as* $y \to \infty$.

Proof: Writing the sum for $T(y)$ as a Stieltjes integral, using integration by parts and applying Lemma 6.7 gives

$$\sum_{\ell>y} \frac{1}{\ell\,\nu(\ell)} = \int_{y+}^{\infty} \frac{1}{x}\, dS(x) = -S(y)/y + \int_{y}^{\infty} S(x)/x^2\, dx \ll y^{-1/2}. \quad ■$$

Conjecture 6.10 $T(y) \ll y^{-1+o(1)}$ *as* $y \to \infty$.

Supporting argument: Provided Conjecture 6.8 holds, the result would follow by the same argument as used in the proof of Lemma 6.9. ■

Lemma 6.11 *Let N denote the number of enumerations of n in Algorithm 6.1, where repeated occurrences of the same value are treated as distinct enumerations. Then*

$$N \ll y^{-1/2}(x_2 - x_1) + \frac{\sqrt{x_2}}{\ln(x_2)}. \tag{6.7}$$

Proof: Let N_ℓ denote the number of values of n enumerated for a fixed value of ℓ. The conditions of the `for` loop in line 5 ensure that $n \equiv \ell \pmod{\ell\,\nu(\ell)}$, $x_1 \le n \le x_2$. Thus $N_\ell \le 1 + (x_2 - x_1)/(\ell\,\nu(\ell))$. Summing over ℓ, applying the prime number theorem and Lemma 6.9, gives

$$N = \sum_{y<\ell\le\sqrt{x_2}} N_\ell \ll \pi(\sqrt{x_2}) + (x_2-x_1)\sum_{y<\ell} \frac{1}{\ell\,\nu(\ell)} \ll y^{-1/2}(x_2-x_1) + \frac{\sqrt{x_2}}{\ln(x_2)}. \quad ■$$

Conjecture 6.12 $N \ll y^{-1+o(1)}(x_2 - x_1)$.

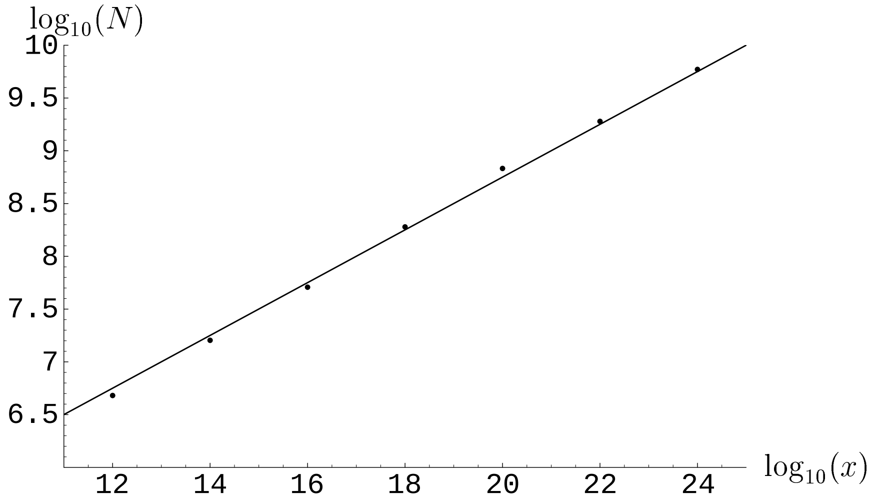


Figure 6.1: Number of prp-tests, N, required by Algorithm 6.1—compared to the approximation $\log_{10}(N) \approx 0.25 \log_{10}(x) + 0.375$

Supporting arguments: Recall, if $\mathrm{psp}(n)$, and $\ell \mid n$ then $n \equiv \ell \pmod{\ell\, \nu(\ell)}$. As in our proof of Theorem 6.11, let

$$N_\ell := \left|\{n \in \mathbb{Z} : \ x_1 \le n \le x_2, n \equiv \ell \pmod{\ell\, \nu(\ell)}\}\right|.$$

We have shown that $N_\ell \le 1 + (x_2 - x_1)/(\ell\, \nu(\ell))$. However, for fixed ℓ the probability that a "random" n satisfies $n \equiv \ell \pmod{\ell\, \nu(\ell)}$ is $1/(\ell\, \nu(\ell))$. Thus, on average over $\ell > y$, we expect $N_\ell = (x_2 - x_1)/(\ell\, \nu(\ell))$. This belief and Conjecture 6.10 suggest that

$$N = \sum_{y < \ell \le \sqrt{x_2}} N_\ell \ll (x_2 - x_1) \sum_{\ell > y} \frac{1}{\ell\, \nu(\ell)} \ll y^{-1+o(1)}(x_2 - x_1).$$

To further test our conjecture, Figure 6.1, and also Table 6.1 on page 132, present data showing the growth of N as a function of a parameter x, with $x_1 = x - 5000\sqrt{x}$, $x_2 = x + 5000\sqrt{x}$, and $y = x^{1/4}$. Although Conjecture 6.12 is stated as giving an upper bound on N, the argument immediately above leads us to expect $N = x^{1/4+o(1)}$, and this expectation seems to be well supported by the data. ■

We now turn to bounding $\mathcal{K}_4$ and $\mathcal{K}_5$, i.e., the number of operations required in Algorithm 6.1 to compute all values of $2^{n-1} \bmod n$ and to place elements into the set $\mathcal{C}$.

Theorem 6.13 *Let $\mathcal{K}_4$ denote the number of operations required in Algorithm 6.1 to compute all values of $2^{n-1} \bmod n$, and let $\mathcal{K}_5$ denote the number of operations required to place elements into the set $\mathcal{C}$. Then*

$$\mathcal{K}_4 \ll y^{-1/2}\ln(x_2)(x_2 - x_1) + \sqrt{x_2}, \quad \textit{and} \tag{6.8}$$

$$\mathcal{K}_5 \ll y^{-1/2}\ln(x_2)(x_2 - x_1) + \sqrt{x_2}. \tag{6.9}$$

Furthermore, if Conjecture 6.12 holds then

$$\mathcal{K}_4 \ll y^{-1+o(1)}\ln(x_2)(x_2 - x_1), \quad \textit{and} \tag{6.10}$$

$$\mathcal{K}_5 \ll y^{-1+o(1)}\ln(x_2)(x_2 - x_1). \tag{6.11}$$

Proof: The algorithm computes N instances of $2^{n-1} \bmod n$. For fixed $n > 1$, $2^{n-1} \bmod n$ can be computed in $O(\ln(n))$ operations. The bound (6.8) then follows from (6.7), since $n \leq x_2$.

Assuming $\mathcal{C}$ is implemented as a balanced tree, an element may be placed into $\mathcal{C}$ using $O(\ln(2 + |\mathcal{C}|))$ operations [Knu73, §6.2.3]. The number of elements placed into $\mathcal{C}$ is bounded by N, while, trivially, $|\mathcal{C}| \leq x_2$. Thus, the bound (6.9) follows from (6.7).

The bounds (6.10) and (6.11) follow similarly under Conjecture 6.12. ■

We now finish our cost analysis that we began just after the presentation of Theorem 6.3 on page 124.

Theorem 6.14 *Algorithm 6.1 requires $\mathcal{B} = O\big(x_2^{1/4} + \ln(x_2)\,|\mathcal{C}|\big)$ bits and $\mathcal{K} = O\big(y^{-1/2}\ln(x_2)(x_2 - x_1) + \ln\ln(x_2)\sqrt{x_2}\big)$ operations. Furthermore, if Conjecture 6.12 holds then $\mathcal{K} = O\big(y^{-1+o(1)}\ln(x_2)(x_2 - x_1) + \ln\ln(x_2)\sqrt{x_2}\big)$.*

Proof: To summarize our analysis of the previous several pages, recall that $\mathcal{B} = \sum_{j=1}^{5} \mathcal{B}_j$, $\mathcal{K} = \sum_{j=1}^{5} \mathcal{K}_j$, where the meanings of $\mathcal{B}_j$ and $\mathcal{K}_j$ are explained on page 125.

In the remarks preceding Lemma 6.4, we noted that

$$\begin{aligned} \mathcal{B}_1 &\ll x_2^{1/4}, & \mathcal{K}_1 &\ll \ln\ln(x_2)\sqrt{x_2} \\ \mathcal{B}_2 &\ll x_2^{1/4}, & \mathcal{K}_2 &\ll \ln\ln(x_2)\sqrt{x_2}, \end{aligned}$$

and that $\mathcal{B}_3 = O(x_2^{\epsilon})$.

In Theorem 6.6 we showed that $\mathcal{K}_3 \ll \ln\ln(x_2)\sqrt{x_2}$, while Theorem 6.13 above gives our bounds on $\mathcal{K}_4$ and $\mathcal{K}_5$. Clearly $\mathcal{B}_4 = O(x_2^\epsilon)$.

Again assuming that $\mathcal{C}$ is implemented as a balanced tree, we have $\mathcal{B}_5 \ll \ln(x_2)\,|\mathcal{C}|$, and the unconditional result follows.

Finally, the result conditional upon Conjecture 6.12 follows from replacing the bounds (6.8) and (6.9) with the bounds (6.10) and (6.11). ■

x	time	N	$\lvert\mathcal{C}_{\text{list}}\rvert$	$\lvert\mathcal{C}\rvert$	$\frac{\lvert\mathcal{C}\cap\mathcal{U}_y\rvert}{\lvert\mathcal{C}\rvert}$
10^{12}	$3.1\cdot10^2$	$4.8\cdot10^6$	538	411	0.52
10^{14}	$1.2\cdot10^3$	$1.6\cdot10^7$	322	266	0.56
10^{16}	$4.6\cdot10^3$	$5.1\cdot10^7$	218	187	0.64
10^{18}	$2.2\cdot10^4$	$1.9\cdot10^8$	127	119	0.82
10^{20}	...	$6.8\cdot10^8$	94	93	0.82
10^{22}	...	$1.9\cdot10^9$	68	66	0.85
10^{24}	...	$5.9\cdot10^9$	45	45	0.96

Table 6.1: Results from Algorithm 6.1 (`CandidatePSPs`). $x_1 = x - 5000\sqrt{x}$, $x_2 = x + 5000\sqrt{x}$, $y = x^{1/4}$. The runtime, in seconds, is given for $x \le 10^{18}$. Times are not given for $x > 10^{18}$ since those computations were run in parallel on several machines of varying speeds. The variable N denotes the number of prp-tests performed in line 6 of the algorithm. Further details are given in Remark 6.15.

Remark 6.15 Table 6.1 shows some results from a C implementation of Algorithm 6.1 (`CandidatePSPs`). For various values of x we used $y = x^{1/4}$ and took intervals of width $10^4\sqrt{x}$ centered around x.

For larger values of x the computation was performed on several computers running in parallel—after splitting the range $y < \ell \le \sqrt{x_2}$ into segments which were allocated among the computers. The computation for $x = 10^{24}$ made use of the Condor system [BL99, CON] for "high throughput computing" on a distributed system of computers. This computation took roughly four days, running on roughly 30 SUN UltraSPARC workstations with processor speeds ranging from 300 to 500 MHz.

Table 6.1 shows both the size of $\mathcal{C}$ and also the size of $\mathcal{C}_{\text{list}}$, i.e., the list of all candidates enumerated by Algorithm 6.1 with duplications. The smallness of $\mathcal{C}$ is striking. For example, in the interval of width 10^{16} about 10^{24} we found only 45 candidates. Interestingly, for this choice of parameters, we see that as x increases $|\mathcal{C}|$ decreases. However, it is far from certain that this trend would continue for even larger values of x.

The rightmost column of the table shows the density of y-unsieved candidates, $|\mathcal{C} \cap \mathcal{U}_y|/|\mathcal{C}|$. (The remaining candidates, those which are not members of $\mathcal{C} \cap \mathcal{U}_y$, are "spurious", y-sieved candidates.) We see that—for this choice of parameters—the density of y-unsieved candidates increases as x increases. □

The data presented in Table 6.1 suggests that $|\mathcal{C}|$ is very small when $y \gg x_2^{1/4}$ and $x_2 - x_1 \asymp \sqrt{x_2}$. Although we cannot prove that $|\mathcal{C}|$ is small, we will show in Theorem 6.20 on page 136 that the plausible, and relatively weak, conditions of Conjecture 6.16, below, suffice to ensure that the hybrid sieve can efficiently enumerate primes near x_2 using $O(x_2^{1/4})$ bits.

Conjecture 6.16 *Provided $y \gg x_2^{1/4}$ and $x_2 - x_1 \gg \sqrt{x_2}$ we conjecture that $|\mathcal{C}| \ll (x_2 - x_1)x_2^{-1/4}/\ln(x_2)$.*

Supporting Argument: With $y \gg x_2^{1/4}$, Conjecture 6.12 on page 129 suggests that $N \ll (x_2 - x_1)x_2^{-1/4+o(1)}$, and, trivially, $|\mathcal{C}| \leq N$. However, N counts composite n which satisfy $n \equiv \ell \pmod{\ell\,\nu(\ell)}$ for some $\ell > y$. For n to be in $\mathcal{C}$ it is necessary for n to also satisfy $2^{n-1} \equiv 1 \pmod{n}$, which we believe to be a *much* stronger condition. Thus, when bounding $|\mathcal{C}|$ it seems reasonable to drop both the $x_2^{o(1)}$ factor in our conjectured bound on N and also an additional factor of $\ln(x_2)$. ■

6.3 Main phase: the hybrid sieve

Having analyzed the first phase of the hybrid sieve, which finds the candidate pseudoprimes in an interval, we will now complete our treatment of the hybrid sieve—following the outline given in Section 6.1. We begin by presenting a simple variation on thesieve of Eratosthenes which finds y-unsieved numbers. This exposition uses some of the terminology and analyses given in Section 4.2.

Algorithm 6.2 (`PartialSieve`: Find y-unsieved numbers)
Given $y > 0$ and given a preallocated bit vector `Uset` *with* `Uset.x1` $= x_1$, `Uset.x2` $= x_2$, $1 \leq x_1 \leq x_2$, *this algorithm sets* `Uset`$[n]$ *such that upon completion we have* `Uset`$[n] = 1$ *if and only if n is y-unsieved, i.e., if and only if*

$n \in \mathcal{U}_y$.

```
PartialSieve(y, Uset) {
 x1 ← Uset.x1;  x2 ← Uset.x2;
 assert x1 ∈ ℕ ∧ x2 ∈ ℕ;  assert 1 ≤ x1 ≤ x2;
 // Initialize Uset to have all 1 entries.
 for (n ← x1; n ≤ x2; n++)     Uset[n] ← 1;
 // Now zero out all entries at multiples of "small" primes.
 for (ℓ ∈ [2, y] ∩ 𝒫)
  for (m ← ⌈x1/ℓ⌉;  m ≤ x2;  m++)
   Uset[mℓ] ← 0;
}
```

Theorem 6.17 *Given $1 \le x_1 \le x_2$, Algorithm 6.2 enumerates the y-unsieved numbers in the interval $[x_1, x_2]$ using $O\big(\ln\ln(y)(x_2 - x_1 + y)\big)$ operations and $O\big(x_2 - x_1 + \sqrt{y} + \ln(x_2)\big)$ bits—uniformly for $y \ge 3$.*

Proof: We require $y \ge 3$ to ensure $\ln\ln(y) > 0$. Using the segmented sieve of Eratosthenes we can enumerate the primes $\ell \le y$ using $O(\ln\ln(y)y)$ operations and $O(\sqrt{y})$ bits. We require $O(\ln(x_2) + x_2 - x_1)$ bits of storage for the bit vector `Uset` and $O(\ln(x_2))$ bits for the other quantities used.

It takes $O(1 + x_2 - x_1)$ operations to initialize `Uset`. This is dominated by the number of operations needed to cross out y-sieved numbers, which is

$$\begin{aligned}
&\ll \sum_{\ell \le y} \left(\frac{1 + x_2 - x_1}{\ell} + 1 \right) \ll (1 + x_2 - x_1) \sum_{\ell \le y} \frac{1}{\ell} + \pi(y) \\
&\ll \ln\ln(y)(x_2 - x_1) + y/\ln(y).
\end{aligned}$$

Summing these bounds on bits and operations, the result follows. ■

To analyze the density of y-unsieved numbers, we will use Corollary 6.19, below, which uses the following theorem from *Sieve Methods* [HR74]:

Theorem 6.18 (The case $K = 2$ of Theorem 3.6 in [HR74]) *Given $x_1 \le x_2$, and $z \ge e^6$, then*

$$|\{n \in \mathcal{U}_z : \ x_1 \le n \le x_2\}| \le \frac{x_2 - x_1 + 1}{\ln(z)} + 48\frac{z^2}{\ln^2(z)}. \tag{6.12}$$

Corollary 6.19 *Given $C > 0$ and $2 \le y \le C(x_2 - x_1)$, the number of y-unsieved numbers in the interval $[x_1, x_2]$ is $O_C((x_2 - x_1)/\ln(y))$.*

Proof: The bound (6.12), and $2 \le y \le C(x_2 - x_1)$, imply

$$|\{n \in \mathcal{U}_z : x_1 \le n \le x_2\}| = O_C\left(\frac{x_2 - x_1}{\ln(z)} + \frac{z^2}{\ln^2(z)}\right) \tag{6.13}$$

uniformly for $z \ge \sqrt{2}$. Note that for any $z \le y$, $\mathcal{U}_y \subseteq \mathcal{U}_z$. Setting $z = \sqrt{y}$ and applying (6.13), we have

$$|\{n \in \mathcal{U}_y : x_1 \le n \le x_2\}| \le |\{n \in \mathcal{U}_z : x_1 \le n \le x_2\}| \ll \frac{x_2 - x_1}{\ln(y)} + \frac{y}{\ln^2(y)}.$$

The rightmost bound is $O((x_2 - x_1)/\ln(y))$ since $x_2 - x_1 \gg y$. ■

Algorithm 6.3, below, implements the hybrid sieve as a two-tiered segmented sieve. Because Algorithm 6.1 (`CandidatePSPs`) has a relatively large overhead of at least $O(\ln\ln(x_2)\sqrt{x_2})$ operations, independent of the width of the interval being processed, we find the set $\mathcal{C}$ of candidate pseudoprimes for "long" subintervals of $[x_1, x_2]$. We then segment each long subinterval into short subintervals of size $y \approx x_2^{1/4}$, so as to keep the memory requirements of Algorithm 6.2 (`PartialSieve`) low.

To ensure that Algorithm 6.3 enumerates only primes, we place some minor restriction on its arguments so that the conditions spelled out in Theorem 6.3 are satisfied—namely that $x_1 > 2$ and $x_1 > y$.

Algorithm 6.3 (`HybridSieve`: Enumerate primes by hybrid method)
Given $2 < \lceil x_2^{1/4} \rceil < x_1 \le x_2$, enumerate the primes in the interval $[x_1, x_2]$.

`HybridSieve`(x_1, x_2) {
  assert $2 < \lceil x_2^{1/4} \rceil < x_1 \le x_2$;
  $L \leftarrow \lceil x_2^{1/2} \rceil$; $y \leftarrow \lceil x_2^{1/4} \rceil$;
  `Uset` $\leftarrow$ `AllocateBitVector`(y);
  // Find candidate pseudoprimes within "long" subintervals of size L.
  for $(k_1 \leftarrow \lceil x_1 \rceil; k_1 \le x_2; k_1 \leftarrow k_2 + 1)$ {
    $k_2 \leftarrow \min(x_2, k_1 + L - 1)$;

We assume that space used for $\mathcal{C}$ is freed and re-used each time we invoke `CandidatePSPs`$(\ldots)$.

8 $\mathcal{C} \leftarrow$ `CandidatePSPs`(y, k_1, k_2) ;
9 *// Now enumerate primes over "short" subintervals of size y.*
10 **for** (`Uset.x1` $\leftarrow k_1$; `Uset.x1` $\leq k_2$; `Uset.x1` $\leftarrow$ `Uset.x2` $+ 1$) {
11 `Uset.x2` $\leftarrow \min(k_2,$ `Uset.x1` $+ y)$;
12 `PartialSieve`$(y,$ `Uset`$)$; *// Find y-unsieved numbers using Algorithm 6.2.*

At this point, by Theorem 6.3 on page 124, n is prime $\iff$ `Uset`$[n] = 1$, *$2^{n-1} \bmod n = 1$, and $n \notin \mathcal{C}$.*

13 **for** ($n \leftarrow$ `Uset.x1` ; $n \leq$ `Uset.x2` ; n++) {
14 **if** (`Uset` $[n] = 1$)
15 **if** ($2^{n-1} \bmod n = 1$ && $n \notin \mathcal{C}$)
16 `Enumerate`(n) ; *// Enumerate n as a prime*
17 }}}}

Theorem 6.20 *Provided its arguments satisfy $x_2 - x_1 \gg \sqrt{x_2}$, Algorithm 6.3 enumerates the primes in the interval $[x_1, x_2]$ using $O(\ln\ln(x_2)(x_2 - x_1))$ operations and $O(x_2^{1/4} + C\ln(x_2))$ bits, where C denotes the maximum value of $|\mathcal{C}|$ over all invocations of line 8. Furthermore, Algorithm 6.3 uses $O(x_2^{1/4})$ bits of memory provided Conjecture 6.16 holds.*

Proof: The operation count for Algorithm 6.3 is dominated by the calls to `CandidatePSPs` at line 8; the calls to `PartialSieve` at line 12; and the prp-tests and checks for $n \notin \mathcal{C}$ at line 15.

Since $y \geq x_2^{1/4}$ and $L \ll \sqrt{x_2}$, Theorem 6.14 implies that each invocation of `CandidatePSPs` requires $\ll \ln(x_2)x_2^{3/8} + \ln\ln(x_2)\sqrt{x_2} \ll \ln\ln(x_2)\sqrt{x_2}$ operations. The number of calls is $\ll (x_2 - x_1)/L \ll (x_2 - x_1)/\sqrt{x_2}$ since we also have $L \gg \sqrt{x_2}$. Thus the total cost of all calls to `CandidatePSPs` is $O(\ln\ln(x_2)(x_2 - x_1))$ operations.

Similarly, by Theorem 6.17 each invocation of `PartialSieve` requires $O(\ln\ln(y)y)$ operations. Thus $O(\ln\ln(x_2)y)$ operations since $y \ll x_2^{1/4}$. The number of calls is $O((x_2 - x_1)/y)$, so the total cost of all calls to `PartialSieve` is $O(\ln\ln(x_2)(x_2 - x_1))$ operations.

Line 15 is executed only when $n \in \mathcal{U}_y$. Since $x_2 - x_1 \gg \sqrt{x_2} \gg x_2^{1/4} \asymp y$ Corollary 6.19 implies that line 15 is executed $O((x_2 - x_1)/\ln(x_2))$ times. In the proof of Theorem 6.13 we noted that the computation of $2^{n-1} \bmod n$ requires $O(\ln(x_2))$ operations when $n \ll x_2$. If $\mathcal{C}$ is implemented as a balanced tree, the test $n \notin \mathcal{C}$ also requires $O(\ln(x_2))$ operations. Thus the total cost over all executions of line 15 is $O(x_2 - x_1)$ operations. Summing all these bounds gives the claimed bound on the operation count for Algorithm 6.3.

Turning now to bounding the memory requirements, we see that the bit vector `Uset` requires $O(x_2^{1/4})$ bits. By Theorem 6.17 this dominates the storage needed by `PartialSieve`. Other than the storage required for $\mathcal{C}$, all other quantities require $O(\ln(x_2))$ bits. The bound of $O(x_2^{1/4} + C\ln(x_2))$ bits then follows unconditionally, where C is defined in the statement of this theorem. The conditional result follows immediately from Conjecture 6.16 on page 133. ■

6.4 Acknowledgments

The idea of the hybrid sieve was inspired by a discussion with Carl Pomerance about how quickly an implementation of a primality test such as the "APRCL test" might run. Pomerance suggested that I first start by finding how fast an implementation of the much simpler prp-test would be, i.e., how quickly a computer could find $2^{n-1} \bmod n$. This led me to consider how one could best use a prp-test to determine primality, in a situation where a large amount of precomputation might be justified, but under the constraint of keeping memory requirements low.

The idea of computing a set of "candidate" pseudoprimes, implemented in Algorithm 6.1 (`CandidatePSPs`), was suggested by an early draft of Richard Pinch's paper [Pin00], which describes similar techniques used in Pinch's compilation of tables of pseudoprimes.

In addition to thanking Gergely Harcos for the proof of Lemma 6.7, I thank Adolf Hildebrand for suggestions which were incorporated into the proofs of Lemma 6.9, Theorem 6.5, and Corollary 6.19.

The C implementation of Algorithm 6.1 (`CandidatePSPs`) made use of the GNU Multiple Precision Arithmetic Library (GMP), version 3.1.1, developed by Torbjörn Granlund and others [G$^+$].

Miron Livny, Jim Basney, and other members of the Condor development team provided excellent support for their Condor system for "high throughput computing".

7 Computing $\zeta(s)$ by Numerical Integration

7.1 Introduction

In Section 3.4 we reduced the problem of approximating $\pi^*(x;\lambda)$ to the computation of a finite sum of the form

$$\frac{h}{\pi}\left(\frac{1}{2}\Psi(\sigma;x,\lambda)+\sum_{k=1}^{K}\operatorname{Re}\Psi(\sigma+ikh;x,\lambda)\right), \tag{7.1}$$

where $\Psi(s;x,\lambda)=e^{\lambda^2 s^2/2}x^s\ln(\zeta(s))/s$. Given the complexity of known methods for computing $\zeta(s)$, it is clear that the computations of $\zeta(\sigma+it)$ will be the dominant component of the complexity of computing (7.1).

In this chapter we propose a method for computing $\zeta(s)$ which uses a variation on the Riemann-Siegel formula [Sie66, Edw74, Gab79] for $\zeta(s)$. For fixed σ, the classical Riemann-Siegel formula gives an asymptotic expansion of $\zeta(\sigma+it)$ as $t\to\infty$. This expansion is derived by applying the saddle point method to an integral closely related to $\zeta(s)$. Our variation differs from the classical formula in that it uses numerical quadrature to evaluate the integral to arbitrary accuracy.

We will first present our "quadrature method" for computing $\zeta(s)$, and in Section 7.6 we will estimate its computational complexity. In Section 7.7 we will comment further on the advantages of the quadrature method over the classical Riemann-Siegel formula. This chapter is quite informal, with only outlines of proofs given.

7.2 An integral representation for $\zeta(s)$

We begin by relating $\zeta(s)$ to the integral (7.2), below. Although this representation is due to Riemann, he applied the saddle point method to a different integral representation in his development of the Riemann-Siegel formula (see [Sie66] or [Edw74, Chapter 7]).

As is standard, we define z^{-s} to be $\exp(-s\ln(z))$, with a branch cut along the negative real axis of the z-plane. Let

$$f(z;s) := \frac{z^{-s}e^{i\pi z^2}}{e^{i\pi z} - e^{-i\pi z}}.$$

Given $N \in \mathbb{Z}_0$, define the integral $I_N(s)$ by

$$I_N(s) := \int_{N \swarrow N+1} f(z;s)\,dz, \tag{7.2}$$

where for $z_0,\ z_1 \in \mathbb{C}$ we write $z_0 \swarrow z_1$ for the path $z = (z_0+z_1)/2 + e^{-3\pi i/4}\alpha$, $-\infty < \alpha < \infty$.

Let $\arg(z) := \operatorname{Im}\operatorname{Ln}(z)$, so that $-\pi < \arg(z) \leq \pi$. The path of integration in Equation (7.2) avoids the poles of the integrand at $z \in \mathbb{Z}$. It follows that along this path, for any $\epsilon > 0$, we have

$$f(z;s) \ll |z|^{-\sigma}\, e^{t\arg(z) - 2\pi xy - \pi|y|} \ll e^{(-\pi+\epsilon)\alpha^2} \quad \text{as } \alpha \to \pm\infty, \tag{7.3}$$

where $z =: x + iy$ defines x and y as functions of α. It follows from the bound (7.3) that the integral of (7.2) converges to give $I_N(s)$ as an entire function of s.

Let

$$\chi(s) := \pi^{s-1/2}\,\frac{\Gamma((1-s)/2)}{\Gamma(s/2)} = 2^{s-1}\pi^s \sec(\pi s/2)/\Gamma(s).$$

It is well-known that $\zeta(s)$ satisfies the functional equation $\zeta(s) = \chi(s)\zeta(1-s)$. Furthermore, Riemann showed ([Sie66, §3], [Edw74, §7.9]) that

$$\zeta(s) = I_0(s) + \chi(s)\,\overline{I_0(1-\overline{s})}. \tag{7.4}$$

This representation converges for $s \in \mathbb{C}$ with the exception of the pole at $s = 1$ and removable singularities at $s = 2k+3$, $k \in \mathbb{Z}_0$, arising from poles of $\chi(s)$ at these points. Outside these singular points, algorithms for computing $\chi(s)$ and $I_0(s)$ yield an algorithm for computing $\zeta(s)$.

The computation of $\chi(s)$ is easily reduced to the task of computing $\Gamma(s)$, which is a well-understood special function. Without further analysis, we note that the formula of Spouge [Spo94] for $\Gamma(s)$, presented in [Cra96, Section 2.4], can be used to find $\Gamma(s)$, and thus $\chi(s)$, with a relative error bounded by ε, using $O(\ln(1/\varepsilon))$ operations.

Turning to the computation of $I_0(s)$, we observe that when we shift the path for $I_0(s)$ to $N \swarrow N+1$ and apply the Cauchy residue formula, we have

$$I_0(s) = \sum_{n=1}^{N} n^{-s} + I_N(s). \tag{7.5}$$

To ensure that $I_N(s)$ is a numerically tractable integral, we choose N so that the path passes near the saddle point of $f(z;s)$. (See Figure 7.1 on the next page.) For fixed σ, we will have $N \sim \sqrt{t/(2\pi)}$ as $t \to \infty$.

Wolfgang Gabcke [Gab79] has applied the saddle point method to $I_N(s)$ to derive and analyze the Riemann-Siegel expansion for

$$Z(t) := \chi(1/2+it)^{-1/2}\zeta(1/2+it).$$

This expansion is asymptotic as $t \to \infty$.

For $t \in \mathbb{R}$, $Z(t)$ satisfies the properties $Z(t) \in \mathbb{R}$, $|Z(t)| = |\zeta(1/2+it)|$. For this reason, $Z(t)$ is preferred over $\zeta(s)$ in studies of zeros of $\zeta(s)$ along the critical line, $s = 1/2+it$, since the location of zeros can be reduced to the task of locating sign changes of $Z(t)$. However, for analytic computation of $\pi(x)$ we need a formula for $\zeta(s)$, so we will only consider that task here.

7.3 A quadrature formula for $I_0(s)$

Instead of applying the saddle point method to give an asymptotic expansion, we note that the integral (7.2) defining $I_N(s)$ is well suited to numerical quadrature when N lies near the saddle point of $f(z;s)$. (Other examples of special functions computed by quadrature of saddle point integrals are given in [Tem77].)

After some intial results, along with analysis and commentary, this section culminates in the quadrature formula for $I_0(s)$, Formula (7.9), given in Theorem 7.3 on page 143. The results of this section follow easily from the Cauchy residue formula, so we will not include their proofs. (See [Hen88, §4.9] for a discussion of how sums may be expressed in terms of residues arising from integrals of the form used below.)

Theorem 7.1 *Let $\mathcal{L}$ be the path $z = z_{\mathcal{L}} + e^{-3\pi i/4}\alpha$, and let $\mathcal{R}$ be the path $z = z_{\mathcal{R}} + e^{-3\pi i/4}\alpha$, parameterized by α, $-\infty < \alpha < \infty$. Further, let $\mathcal{L}$ intersect $\mathbb{R}$ in the open interval $(N, N+1/2)$, and $\mathcal{R}$ intersect $\mathbb{R}$ in the open*

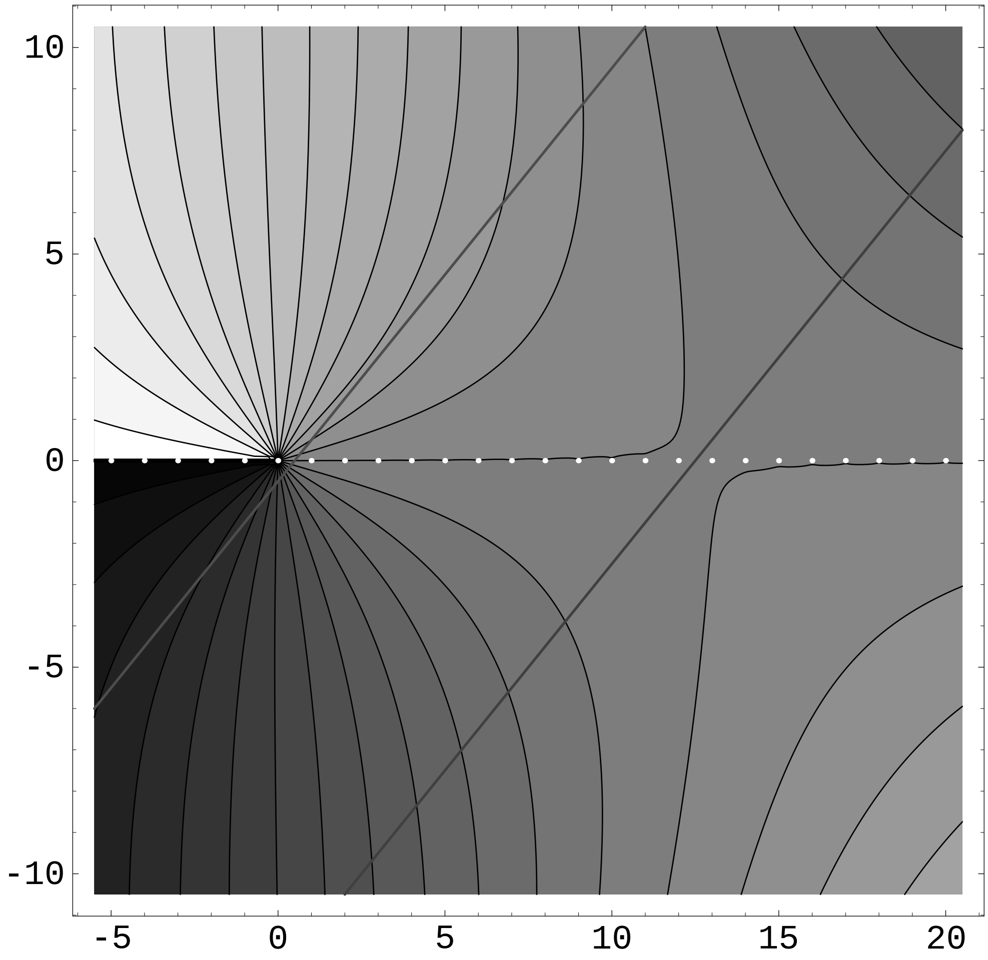


Figure 7.1: Contour plot of $\log_{10}|f(z;s)|$, where $f(z;s)$ is the integrand for $I_N(s)$, $s = 1.5+1000i$. The paths $0 \swarrow 1$ and $12 \swarrow 13$ are shown. The contour interval is 100, larger values are lighter. White dots are placed where $z \in \mathbb{Z}$; the poles of $f(z;s)$ are not apparent at the scale used.
For reference: above the branch cut along $z < 0$, at $z = -5.5 + 0.1i$, we have $|f(z;s)| \approx 10^{1356.55}$ while below the branch cut, at $z = -5.5 - 0.1i$, we have $|f(z;s)| \approx 10^{-1359.41}$. On the path $12 \swarrow 13$, $f(z;s)$ takes a maximum value of ≈ 0.0117.

interval $(N+1/2, N+1)$. *Let* $H(w) = 1/(1-e^{2i\pi w})$. *Given* $h \in \mathbb{C}$ *with* $\arg(h) = -3\pi/4$, *and given* z_1 *lying on the path* $N \swarrow N+1$, *we have*

$$\begin{aligned} I_N(s) &= h \sum_{m\in\mathbb{Z}} f(z_1 + mh; s) \\ &\quad + \int_{\mathcal{L}} f(z;s) H\Big(\frac{z-z_1}{h}\Big)\, dz + \int_{\mathcal{R}} f(z;s) H\Big(\frac{z_1-z}{h}\Big)\, dz. \end{aligned} \tag{7.6}$$

We think of the "side-integrals" along the "side-paths", $\mathcal{L}$ and $\mathcal{R}$, as error terms. We may bound these terms by noting $|H(w)| \le e^{2\pi \operatorname{Im}(w)}/(1-e^{2\pi \operatorname{Im}(w)})$ when $\operatorname{Im}(w) < 0$. Writing Δ for the distance between the paths $\mathcal{L}$ and $N \swarrow N+1$, this gives

$$\begin{aligned} \left| \int_{\mathcal{L}} f(z;s) H\Big(\frac{z-z_1}{h}\Big)\, dz \right| &\le \frac{e^{-2\pi\Delta/|h|}}{1-e^{-2\pi\Delta/|h|}} \int_{\mathcal{L}} |f(z;s)|\; |dz| \\ &= O_\Delta(e^{-2\pi\Delta/|h|}) \quad \text{as } |h| \to 0. \end{aligned} \tag{7.7}$$

Similarly, writing Δ for the distance between the paths $\mathcal{R}$ and $N \swarrow N+1$, we have

$$\left| \int_{\mathcal{R}} f(z;s) H\Big(\frac{z_1-z}{h}\Big)\, dz \right| = O_\Delta(e^{-2\pi\Delta/|h|}) \quad \text{as } |h| \to 0. \tag{7.8}$$

For fixed Δ, the side-integrals decrease exponentially in $1/|h|$. Shifting the side-paths away from the central path increases Δ, and may decrease these error terms. The optimal choices for $\mathcal{L}$ and $\mathcal{R}$ should lie near saddle points of the integrands in (7.7) and (7.8), respectively. We illustrate the new paths in Figure 7.2 on the next page. In Section 7.4 we give heuristics for finding points $z_{\mathcal{L}}$ and $z_{\mathcal{R}}$ which are near the desired saddle points.

Shifting the side-paths introduces "side-sums", arising from residues picked up from the poles of $f(z;s)$. Corollary 7.2 gives the resulting formula.

Corollary 7.2 *Given* $N_{\mathcal{L}}, N, N_{\mathcal{R}} \in \mathbb{Z}_0$, $0 < N_{\mathcal{L}} \le N < N_{\mathcal{R}}$, *let* $\mathcal{L}$ *be a path between* $N_{\mathcal{L}}-1$ *and* $N_{\mathcal{L}}$, *parallel to and traveling in the same direction as* $N \swarrow N+1$, *and let* $\mathcal{R}$ *be a similar path between* $N_{\mathcal{R}}$ *and* $N_{\mathcal{R}}+1$. *Then*

$$\begin{aligned} I_N(s) = h \sum_{m\in\mathbb{Z}} f(z_1 + mh; s) &- \sum_{n=N_{\mathcal{L}}}^{N} n^{-s} H\Big(\frac{n-z_1}{h}\Big) + \sum_{n=N+1}^{N_{\mathcal{R}}} n^{-s} H\Big(\frac{z_1-n}{h}\Big) \\ &+ \int_{\mathcal{L}} f(z;s) H\Big(\frac{z-z_1}{h}\Big)\, dz + \int_{\mathcal{R}} f(z;s) H\Big(\frac{z_1-z}{h}\Big)\, dz. \end{aligned}$$

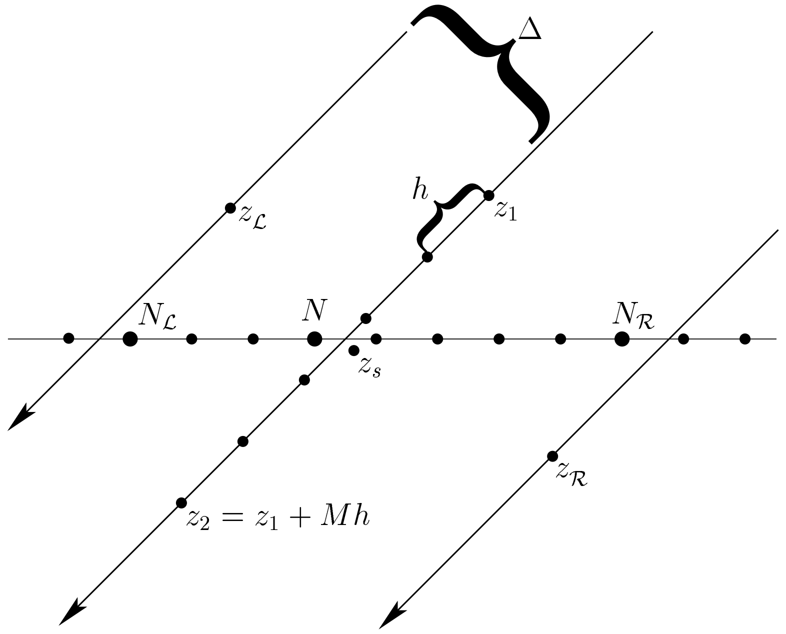


Figure 7.2: Paths and parameters related to Formula (7.9). Note that the left and right paths are shifted from the locations specified in Theorem 7.1.

Combining Corollary 7.2 with Equation (7.5) gives our quadrature formula for $I_0(s)$:

Theorem 7.3 *Given $N_{\mathcal{L}}$, N, $N_{\mathcal{R}} \in \mathbb{Z}_0$, $0 < N_{\mathcal{L}} \le N < N_{\mathcal{R}}$, let $\mathcal{L}$ and $\mathcal{R}$ denote the paths defined in the statement of Corollary 7.2. Then*

$$\begin{aligned} I_0(s) = {} & \sum_{n=1}^{N} n^{-s} + h \sum_{m=0}^{M} f(z_1 + mh; s) \\ & - \sum_{n=N_{\mathcal{L}}}^{N} n^{-s} H\Big(\frac{n - z_1}{h}\Big) + \sum_{n=N+1}^{N_{\mathcal{R}}} n^{-s} H\Big(\frac{z_1 - n}{h}\Big) \\ & + \mathcal{E}_{\Sigma} + \mathcal{E}_{\int}, \end{aligned} \tag{7.9}$$

where $\mathcal{E}_{\Sigma}$ is the error due to truncating $\sum_m f(z_1 + mh; s)$ to a finite sum,

$$\mathcal{E}_{\Sigma} := h \sum_{m<0} f(z_1 + mh; s) + h \sum_{m>M} f(z_1 + mh; s), \tag{7.10}$$

and where

$$\mathcal{E}_{\int} := \int_{\mathcal{L}} f(z; s) H\Big(\frac{z - z_1}{h}\Big)\, dz + \int_{\mathcal{R}} f(z; s) H\Big(\frac{z_1 - z}{h}\Big)\, dz. \tag{7.11}$$

(Note that the $\mathcal{E}_\Sigma$ defined by Equation (7.10) is completely unrelated to the $\mathcal{E}_\Sigma$ defined in Chapter 3.)

Remark The side-sums in Formula (7.9) serve to "smooth" the truncated Dirichlet series, $\sum_{n=1}^{N} n^{-s}$, giving a more accurate result, as discussed below. This is similar to a parameterized family of expansions for $\zeta(s)$ studied by Berry and Keating [BK92], which are analogous to the classical Riemann-Siegel formula but more accurate, and which use a sum $\sum_n a_n n^{-s}$ where a_n drops "smoothly" to zero as $n \to \infty$. □

7.4 Heuristics for choosing parameters

Using the functional equation for $\zeta(s)$ and the fact that $\zeta(\overline{s}) = \overline{\zeta(s)}$, we can reduce the problem of computing $\zeta(s)$ to the case where $\sigma \geq 1/2$, $t \geq 0$, so throughout the rest of this chapter we will assume that these conditions hold when computing $\zeta(s)$.

Referring to Equation (7.4), we see that to compute $\zeta(s)$ with an error bounded by $\varepsilon > 0$ it suffices to compute $I_0(s)$ with an error bounded by $\varepsilon/2$ and to compute $I_0(1-\overline{s})$ with an error bounded by $|\chi(s)|^{-1}\varepsilon/2$. With an obvious change of variables we need only consider the problem of approximating $I_0(\sigma + it)$, $\sigma \in \mathbb{R}$, $t > 0$, with an error bounded by $\varepsilon > 0$.

In this section we give heuristics for choosing the parameters in Formula (7.9), focusing only on truncation errors and ignoring the issue of round-off error. The choice of parameters suggested by our analysis are suitable for the analytic algorithm for computing $\pi(x)$, since they apply for "moderate" σ, "large" t, and an error bound ε which is not "too small". Since we have not completely analyzed Formula (7.9) we ensure only that our error is $O(\varepsilon)$, uniformly under restrictions spelled out below.

We begin by sketching a computational procedure for determining our parameters, rather than giving analytic expressions for them. Figure 7.2 on the preceding page illustrates the paths and most of the parameters used in our analysis.

In part, our task is to bound $|\mathcal{E}_\Sigma| + |\mathcal{E}_\int|$, where $\mathcal{E}_\Sigma$ and $\mathcal{E}_\int$ are the error terms in Formula (7.9). We also want to choose our parameters to minimize the number of operations. To simplify our analysis, we assume that this is equivalent to minimizing the number of terms in Formula (7.9), i.e., to minimizing $N + M + N_{\mathcal{R}} - N_{\mathcal{L}}$. Since N must lie within a limited range to yield a tractable integrand, and to further simplify our analysis, we then assume that minimizing the number of terms is equivalent to minimizing M. In Section 7.6 we will see that M typically dominates $N_{\mathcal{R}} - N_{\mathcal{L}}$, so this assumption is reasonable.

Let $g(z;s) := z^{-s}e^{i\pi z^2}$, so $f(z;s) = g(z;s)/(e^{i\pi z} - e^{-i\pi z})$, and note that

$$f(z;s) = O_\delta\big(g(z;s)e^{-\pi|\operatorname{Im} z|}\big) \ll g(z;s), \tag{7.12}$$

where δ is the distance from z to the nearest integer. Of the two saddle points of $g(z;s)$, we find that the saddle point, z_s, with $\operatorname{Re} z_s > 0$ is

$$z_s := \sqrt{s/(2i\pi)}.$$

Given z_s, we set $N = \lfloor \operatorname{Re} z_s - \operatorname{Im} z_s \rfloor$, which ensures that $N \swarrow N+1$ passes near z_s.

To bound the error terms in Formula (7.9), we focus on the magnitude of $f(z;s)$ and $g(z;s)$ at four key points. Two of these points, z_1 and z_2, lie on $N \swarrow N+1$ and denote the endpoints of the sum $\sum_m f(z_1 + mh; s)$. The other two points, $z_{\mathcal{L}}$, $z_{\mathcal{R}}$, lie on the paths $\mathcal{L}$ and $\mathcal{R}$. They are chosen to lie near the maximum along these paths of the integrands appearing in the integrals $\int_{\mathcal{L}} \cdots$ and $\int_{\mathcal{R}} \cdots$, respectively, of Equation (7.11).

We may parameterize our paths as $z = e^{-i\pi/4}R + e^{-3\pi i/4}\alpha$, $-\infty < \alpha < \infty$, where R measures the signed distance from the path to the origin. (We have $R < 0$ if the path intersects $\mathbb{R}$ to the left of the origin.) With this parameterization, we have

$$\frac{d}{d\alpha} \ln|g(z;s)| = -\frac{2\pi\alpha^3 + (2\pi R^2 + \sigma)\alpha + Rt}{\alpha^2 + R^2}. \tag{7.13}$$

An analysis of (7.13) shows that taking $R \geq \sqrt{|\sigma|/\pi}$ suffices to ensure that $|g(z;s)|$ as a function of α is both unimodal and has a "sharp" maximum along a path parameterized as above.

With this in mind, assume that $N \swarrow N+1$ is sufficiently far from the origin, e.g., that $N + 1/2 \geq \sqrt{2|\sigma|/\pi}$. We then bound $\mathcal{E}_\Sigma$ by choosing z_1, z_2, to lie on the path $N \swarrow N+1$, subject to the restrictions that they lie on opposite sides of the point where $|g(z;s)|$ reaches its maximum along this path, and to satisfy the four additional conditions: $\operatorname{Im} z_1 > 0$, $\operatorname{Im} z_2 < 0$,

$$|f(z_1; s)| \le \varepsilon, \tag{7.14}$$

$$|f(z_2; s)| \le \varepsilon. \tag{7.15}$$

An analysis of (7.13) and the bound (7.12) shows that these conditions ensure that $|\mathcal{E}_\Sigma| \ll |f(z_1; s)| + |f(z_2; s)| \ll \varepsilon$. Of course, to avoid an unnecessarily long interval of summation, the ideal choice for z_1 and z_2 would satisfy (7.14) and (7.15) with equality holding—unless this choice violated our conditions that $\operatorname{Im} z_1 > 0$, $\operatorname{Im} z_2 < 0$.

Turning to the analysis of $\mathcal{E}_\int$, we begin by examining the "left" side-integral appearing in Equation (7.11):

$$\int_{\mathcal{L}} f(z; s) H\Big(\frac{z - z_1}{h}\Big)\, dz.$$

As outlined in Section 7.3, if $\mathcal{L}$ lies a distance Δ to the left of z_1, and if z lies on $\mathcal{L}$, we have

$$H((z - z_1)/h) = e^{-2i\pi(z - z_1)/h} + O_\Delta(e^{-4\pi\Delta/|h|}),$$

where the O-term is uniform provided Δ is bounded away from zero. In other words, if z lies well to the left of $N \swarrow N+1$ then $H((z - z_1)/h)$ is well approximated by $e^{-2i\pi(z-z_1)/h}$. For this reason, for fixed $h > 0$, we let $\mathcal{L}$ pass through $z_{\mathcal{L}}$, where we define $z_{\mathcal{L}}$ to be the saddle point of $g(z; s)e^{-2i\pi(z-z_1)/h}$ having positive real part:

$$z_{\mathcal{L}} := (2h)^{-1} + \sqrt{(2h)^{-2} + s/(2i\pi)}\,. \tag{7.16}$$

Again, if R denotes the distance from $\mathcal{L}$ to the origin, and if $R \ge \sqrt{|\sigma|/\pi}$, then $|g(z; s)|$ assumes a single maximum along $\mathcal{L}$, at $z = z_{\mathcal{L}}$. Under the additional condition that $\mathcal{L}$ is bounded a distance δ from the singularities of $f(z; s)$, we can show that

$$\int_{\mathcal{L}} f(z; s) H\Big(\frac{z - z_1}{h}\Big)\, dz \ll \left|g(z_{\mathcal{L}}; s)e^{-2i\pi(z_{\mathcal{L}} - z_1)/h}\right|, \tag{7.17}$$

with an O-constant depending on δ and Δ.

By deforming $\mathcal{L}$ to a region where the integrand is well behaved, we believe that the bound (7.17) can be made to hold uniformly in δ, even when the original path passes near or through a pole of $f(z;s)$. Since $z_{\mathcal{L}}$ is a saddle point, we expect our choice of $\mathcal{L}$ to be nearly optimal, again provided Δ is bounded away from zero, and that R is sufficiently large.

A similar analysis applies to the "right" side-integral along $\mathcal{R}$. Thus, we let $\mathcal{R}$ pass through $z_{\mathcal{R}}$, where we define $z_{\mathcal{R}}$ to be the saddle point of $g(z;s)e^{2i\pi(z-z_1)/h}$ with positive real part:

$$z_{\mathcal{R}} := -(2h)^{-1} + \sqrt{(2h)^{-2} + s/(2i\pi)}\,. \tag{7.18}$$

Given $z_{\mathcal{L}}$ and $z_{\mathcal{R}}$ as defined above, the requirement that $\mathcal{L}$ passes between $N_{\mathcal{L}} - 1$ and $N_{\mathcal{L}}$, and that $\mathcal{R}$ passes between $N_{\mathcal{R}}$ and $N_{\mathcal{R}} + 1$ imply that

$$\begin{aligned} N_{\mathcal{L}} &= \lceil \operatorname{Re} z_{\mathcal{L}} - \operatorname{Im} z_{\mathcal{L}} \rceil\,, \\ N_{\mathcal{R}} &= \lfloor \operatorname{Re} z_{\mathcal{R}} - \operatorname{Im} z_{\mathcal{R}} \rfloor\,. \end{aligned}$$

We conclude, given fixed $h > 0$, that if $\mathcal{L}$ is sufficiently far to the right of the origin, and if $\mathcal{L}$ and $\mathcal{R}$ are bounded away from $N \swarrow N+1$, then the choices for $\mathcal{L}$, $\mathcal{R}$ outlined above should be nearly optimal and that

$$\left|\mathcal{E}_f\right| \ll \left|g(z_{\mathcal{L}};s)e^{-2i\pi(z_{\mathcal{L}}-z_1)/h}\right| + \left|g(z_{\mathcal{R}};s)e^{2i\pi(z_{\mathcal{R}}-z_1)/h}\right|.$$

With $z_{\mathcal{L}}$ and $z_{\mathcal{R}}$ determined as functions of h by Equations (7.16) and (7.18), we determine h and M by letting $h := (z_2 - z_1)/M$, so that h, $z_{\mathcal{L}}$, and $z_{\mathcal{R}}$ can be treated as functions of M. Finally, we choose M to be the least integer with $M > 0$ and

$$\left|g(z_{\mathcal{L}};s)e^{-2i\pi(z_{\mathcal{L}}-z_1)/h}\right| \le \varepsilon, \tag{7.19}$$

$$\left|g(z_{\mathcal{R}};s)e^{2i\pi(z_{\mathcal{R}}-z_1)/h}\right| \le \varepsilon. \tag{7.20}$$

7.5 Examples

Table 7.2 on page 150 illustrates the parameters found by using the heuristics of Section 7.4. These heuristics were implemented using the PARI/GP calculator, version 2.1.0 using the 32-bit MicroSparc kernel.

In rough outline, we used the following procedure to find z_1 satisfying the condition (7.14). With the parameterization $z = N + 1/2 - e^{-3\pi i/4}\alpha$ we began with an initial guess for α and then repeatedly doubled α until $|f(z;s)| \le \varepsilon$. Similarly, we repeatedly halved α until $|f(z;s)| \ge \varepsilon$. This process gives values bracketing a solution to $|f(z;s)| = \varepsilon$. Given these bracketing values, we then we used PARI's `solve` function to find the solution to $|f(z;s)| = \varepsilon$. Only limited accuracy is needed for this solution, so we implemented this procedure using 19 digits of precision.

Similar techniques were used to find z_2 satisfying (7.15); and to find h, $z_{\mathcal{L}}$, and $z_{\mathcal{R}}$ defined as functions of M with $z_{\mathcal{L}}$ satisfying (7.19), and $z_{\mathcal{R}}$ satisfying (7.20). Although the use of PARI's `solve` function is somewhat computationally expensive, we found this method to be quite practical, and as $t \to \infty$ or $\varepsilon \to 0$ the cost of determining parameters is dominated by the cost of evaluating Formula (7.9).

Table 7.1 illustrates the accuracy achieved when using these parameters. To collect the data summarized in that table, we computed $\zeta(s_0 + i\,m\,d)$, $0 \le m < 100$, starting at $s_0 = \sigma + it_0$. When t_0 is sufficiently large compared to σ, our analysis in Section 7.4 suggests that N should increase by unity as t increases from t_0 to $t_0 + 2\sqrt{2\pi t_0}$. For this reason, we used a step value of $d = \sqrt{2\pi\,\mathrm{Im}(s_0)}/5$, which ensured that N increased by a few units, and which made it more likely that for some values of s the saddle point z_s lay near a singularity of $f(z;s)$.

s_0	ε	min	max	mean
$1/2 + 10^3 i$	10^{-25}	$2.87 \cdot 10^{-32}$	$1.45 \cdot 10^{-28}$	$3.03 \cdot 10^{-29}$
	10^{-50}	$1.49 \cdot 10^{-56}$	$7.89 \cdot 10^{-54}$	$2.30 \cdot 10^{-54}$
	10^{-100}	$3.37 \cdot 10^{-106}$	$5.36 \cdot 10^{-104}$	$1.95 \cdot 10^{-104}$
$5 + 10^3 i$	10^{-25}	$6.91 \cdot 10^{-30}$	$2.02 \cdot 10^{-28}$	$5.07 \cdot 10^{-29}$
	10^{-50}	$4.02 \cdot 10^{-55}$	$6.12 \cdot 10^{-54}$	$2.95 \cdot 10^{-54}$
	10^{-100}	$1.93 \cdot 10^{-105}$	$4.71 \cdot 10^{-104}$	$2.00 \cdot 10^{-104}$

Table 7.1: Minimum error, maximum error, and mean error in computations of $\zeta(s_0 + 2ik\sqrt{2\pi\,\mathrm{Im}(s_0)}/5)$, $0 \le k < 100$.

For a given "error goal", ε, and for each value of s, we found $I_0(s)$ using parameters selected for an error goal of $\varepsilon/2$, while we found $I_0(1-\overline{s})$ using parameters selected for an error goal of $|\chi(s)|^{-1}\varepsilon/2$, and finally $\zeta(s)$ was computed using Equation (7.4). These values were compared against the values returned by PARI's `zeta` function, which uses Euler-Maclaurin summation to find $\zeta(s)$.

All computations were performed using 10 extra digits of precision. For example, when $\varepsilon = 10^{-25}$ the computations were done using at least 35 digits of precision, using the PARI/GP construct `default(realprecision,35)`. Note that Table 7.1 shows results well within the error goal.

7.6 Complexity of the quadrature method

To summarize the results of Section 7.4: provided t is sufficiently large and ε is not too small, the parameters chosen by the heuristics outlined in that section ensure an error in the computation of $I_0(s)$ that is uniformly $O(\varepsilon)$. This is certainly the case if these heuristics give $N_{\mathcal{L}} \geq 1/2 + \sqrt{2\,|\sigma|\,/\pi}$. In contrast, when t or ε are small enough, and especially when $\mathrm{Re}(s)$ is negative, this choice of parameters may cause the path $\mathcal{L}$ to pass too near the origin to ensure the validity of our analysis. (It is even possible that the procedure above will give a nonsensical result with $N_{\mathcal{L}} < 0$.)

For fixed σ and ε, as $t \to \infty$, we can give estimates of our parameters which are more analytically tractable than the characterization given in Section 7.4. Table 7.3 on the next page shows the parameter values based on these estimates. Those values can be contrasted against those found using the procedures of Section 7.4—which are shown in Table 7.2.

Recall that we set $N = \lfloor \mathrm{Re}\, z_s - \mathrm{Im}\, z_s \rfloor$ with $z_s := \sqrt{s/(2i\pi)}$. Provided $t \geq 2\,|\sigma|$, it is easy to show that $z_s = \sqrt{t/(2\pi)} + (1/\sqrt{2\pi})O(\sigma/\sqrt{t})$. It follows that $N = \sqrt{t/(2\pi)} + 1.5\,O(1)$ provided $t \geq \max(\sigma^2, 2\,|\sigma|)$.

Taking the Taylor expansion of $\ln g(z;s)$ about z_s, we find that provided $|z - z_s| \leq |z_s|\,/2$ we have

$$\ln g(z;s) = i\pi z_s^2 - s\ln(z_s) + 2i\pi(z - z_s)^2 + O\left((z - z_s)^3/\sqrt{|s|}\right). \tag{7.21}$$

Dropping the O-term and exponentiating both sides of (7.21) gives a sufficiently accurate approximation to both $g(z;s)$ and to $f(z;s)$ to estimate our remaining parameters. Setting

$$\Delta := \sqrt{\max\left(0,\, \ln(N^{-\sigma}/\varepsilon)/(2\pi)\right)}, \tag{7.22}$$

we find that z_1 and z_2 lie near $N + 1/2 \pm e^{i\pi/4}\Delta$ respectively; $N_{\mathcal{L}}$ and $N_{\mathcal{R}}$ lie near $N + 1/2 \mp \sqrt{2}\Delta$ respectively; and $M \approx 4\Delta^2$. The case $\Delta = 0$ corresponds

s	ε	M	$\lvert z_2 - z_1\rvert$	$N_{\mathcal{L}}$	N	$N_{\mathcal{R}}$
	10^{-25}	36	5.64	9	12	17
$1/2 + 10^3 i$	10^{-50}	75	8.18	7	12	19
	10^{-100}	155	11.80	6	12	23
	10^{-25}	37	5.58	395	398	403
$1/2 + 10^6 i$	10^{-50}	75	8.12	393	398	405
	10^{-100}	149	11.69	390	398	408
	10^{-25}	33	5.46	12612	12615	12619
$1/2 + 10^9 i$	10^{-50}	70	8.04	12610	12615	12621
	10^{-100}	143	11.64	12607	12615	12624
	10^{-25}	7	2.25	12614	12615	12617
$5 + 10^9 i$	10^{-50}	43	6.24	12611	12615	12620
	10^{-100}	116	10.45	12608	12615	12623
$-4 + 10^9 i$	$\frac{10^{-100}}{\lvert\chi(5+10^9 i)\rvert}$	116	10.45	12608	12615	12623

Table 7.2: Data on parameters for computing $\zeta(s)$ by quadrature, as functions of s and ε. These parameters were found using the heuristics of Section 7.4.

s	ε	M	$\lvert z_2 - z_1\rvert$	$N_{\mathcal{L}}$	N	$N_{\mathcal{R}}$
	10^{-25}	36	5.97	8	12	16
$1/2 + 10^3 i$	10^{-50}	72	8.51	6	12	18
	10^{-100}	146	12.07	4	12	20
	10^{-25}	35	5.89	394	398	402
$1/2 + 10^6 i$	10^{-50}	71	8.45	393	398	403
	10^{-100}	145	12.03	390	398	406
	10^{-25}	34	5.80	12611	12615	12619
$1/2 + 10^9 i$	10^{-50}	70	3.38	12610	12615	12620
	10^{-100}	144	11.93	12607	12615	12623
	10^{-25}	7	2.57	12614	12615	12616
$5 + 10^9 i$	10^{-50}	43	6.58	12611	12615	12619
	10^{-100}	117	10.79	12608	12615	12622
$-4 + 10^9 i$	$\frac{10^{-100}}{\lvert\chi(5+10^9 i)\rvert}$	117	10.79	12068	12615	12622

Table 7.3: Data on alternate choice of parameters for computing $\zeta(s)$ by quadrature, as functions of s and ε. These parameters were found using the estimates of Section 7.6—see the discussion near Equation (7.22) on the previous page.

to the situation where $I_0(s)$ is well approximated by $\sum_{n=1}^{N} n^{-s}$, which is the case for $t > 2\pi(e^{2\pi}\varepsilon)^{-2/\sigma}$, roughly speaking.

As illustrated in the last lines of Tables 7.2 and 7.3, nearly identical parameters suffice to compute $I_0(1-\overline{s})$ with an error bounded by $|\chi(s)|^{-1}\varepsilon$, since $|\chi(\sigma+it)| \sim (t/(2\pi))^{1/2-\sigma}$ as $t \to \infty$ [Tit86, Eq. 4.12.3].

We now briefly consider the more intractable case where t may be small, or where ε may be *very* small. The quadrature method can be used to compute $\zeta(s)$ to arbitrary accuracy provided s is bounded away from points of the form $s = 2k+1$, $k \in \mathbb{Z}_0$, corresponding to the poles of $\chi(s)$. To accomplish this, we again set $z_s := \sqrt{s/(2i\pi)}$ and $N = \lfloor \operatorname{Re} z_s - \operatorname{Im} z_s \rfloor$. Although it might not be the optimal choice, set $N_{\mathcal{L}} = N+1$ and $N_{\mathcal{R}} = N$, so that the two side-sums of Formula (7.9) are empty sums. In this case, we take $\mathcal{L}$ to be $N \swarrow N+1/2$ and $\mathcal{R}$ to be $N+1/2 \swarrow N+1$.

As outlined on page 142, in the discussion of the bounds (7.7) and (7.8), we can choose h so that $|\mathcal{E}_{\int}| \leq \varepsilon/2$ and so that $1/\ln(1/\varepsilon) = O_s(h)$. By the bound (7.3), we can also choose z_1 and $z_2 = z_1 + Mh$ so that $|\mathcal{E}_{\Sigma}| \leq \varepsilon/2$ and $|z_1 - z_1| = O_s\big(\sqrt{\ln(1/\varepsilon)}\big)$, which implies that $M = O_s\big(\ln^{3/2}(\varepsilon/2)\big)$.

It is clear from the analyses above that for fixed ε and large t the time required to find a single value of $\zeta(\sigma+it)$ using Formula (7.9) will be dominated by the $O(t^{1/2+\epsilon})$ operations required to compute the truncated Dirichlet series: $\sum_{n=1}^{N} n^{-s}$, assuming that the sum is implemented in the form in which it is written.

However, Odlyzko and Schönhage have developed an algorithm that computes this truncated Dirichlet series with an average cost of $O(t^{\epsilon})$ operations, provided σ is fixed and that t runs through many values within a limited range. More precisely, their theorem can be paraphrased as:

Theorem 7.4 (Theorem 1.1 of [OS88]) *Given any positive constants ϵ, σ, and c_1, there is an effectively computable constant $c_2 = c_2(\epsilon, \sigma, c_1)$ and an algorithm that for every $T > 0$ will perform $\leq c_2 T^{1/2+\epsilon}$ arithmetic operations on numbers of $\leq c_2 \ln(T)$ bits using $\leq c_2 T^{1/2+\epsilon}$ bits of storage, and will then be capable of computing any value $\zeta(\sigma+it)$, $T \leq t \leq T + T^{1/2}$, to within $\pm T^{-c_1}$ in $\leq c_2 T^{\epsilon}$ operations using the precomputed values.*

Although their paper is based on computing $\zeta(s)$ using the classical Riemann-Siegel formula, the same result holds when we use our quadrature method.

7.7 Remarks and conclusions

We must resolve several remaining issues in order to complete the analysis given in Section 7.4. To begin, we need a more careful analysis to find explicit O-constants. There should be constants C_1 and C_2 such that

$$\begin{aligned} |\mathcal{E}_\Sigma| &\le C_1(|f(z_1;s)| + |f(z_1;s)|) \\ \left|\mathcal{E}_\int\right| &\le C_2\left(\left|g(z_\mathcal{L};s)e^{-2i\pi(z_\mathcal{L}-z_1)/h}\right| + \left|g(z_\mathcal{R};s)e^{2i\pi(z_\mathcal{R}-z_1)/h}\right| \right). \end{aligned}$$

Given C_1, C_2, we could then modify the conditions (7.14), (7.15), (7.19) and (7.20) to ensure $|\mathcal{E}_\Sigma| + \left|\mathcal{E}_\int\right| \le \varepsilon$. As it is, ε serves more as an "error goal" than an error bound, and we only have experimental evidence that the rough analysis of Section 7.4 yield parameters for Formula (7.9) which give satisfactory accuracy (see Table 7.1).

Secondly, we have ignored the computational cost of determining the parameters used in Formula (7.9). This cost will depend largely on the method used to ensure conditions such as (7.14).

Finally, our analysis ignores roundoff error and it also assumes that $\chi(s)$ in Equation (7.4) and all values in Formula (7.9) are computed exactly. A complete analysis would determine error bounds required of $\chi(s)$, and of the subexpressions in Formula (7.9), to ensure a total error bounded by ε.

We expect that the evaluation of $\sum_{n=1}^N n^{-s}$ will always be the dominant component of computing Formula (7.9). However, provided $\sum_{n=1}^N n^{-s}$ can be computed efficiently enough, it may be worth noting that many of the other quantities in Formula (7.9) can be calculated infrequently and then reused many times. For large t_0, with t roughly within the interval $[t_0, t_0 + 2\sqrt{2\pi t_0}]$, the parameters z_1, N, M, $N_\mathcal{L}$, $N_\mathcal{R}$ can stay fixed as t varies, and the quantities $H(\dots)$ in the side-sums need be calculated only once. Many subexpressions in Formula (7.9)—such as $\ln(z_1 + mh)$, implicitly calculated while evaluating $\sum_{m=0}^M f(z_1 + mh; s)$—may be reused so long as N stays constant. However, in this case a more complicated error analysis than given in Section 7.4 would be required to ensure a given error bound.

As noted following Equation (7.4), the quadrature method is not appropriate for computing $\zeta(2k+3)$, $k \in \mathbb{Z}_0$. More generally, for large σ and t near 0 the analysis of the quadrature method is difficult, largely because in this situation a path passing near z_s also passes near the branch cut along

$z < 0$. In these situations, the Euler-Maclaurin summation formula is probably preferable for computing $\zeta(s)$.

The quadrature method has several strengths. These include its ability to find $\zeta(s)$ to arbitrary accuracy (for most values of s), and the simplicity of its error analysis. This analysis, outlined in Section 7.4, is relatively insensitive to our choice of σ and to our error bound ε. In contrast, the classical Riemann-Siegel formula gives limited accuracy; good error bounds for the Riemann-Siegel formula are available only when $\sigma = 1/2$; and the error analysis becomes significantly more complicated with each additional term included in the expansion [Gab79].

The integral representation for $\zeta(s)$ used here may be generalized to Dirichlet L-functions. Deuring has used these representations to develop analogues of the Riemann-Siegel formula for L-functions [Deu67]. In a similar manner, it should be possible to generalize the quadrature method to the computation of L-functions.

Several other promising methods for computing $\zeta(s)$ have been proposed recently. These include the work of Berry and Keating mentioned above, methods discussed in the survey article by Borwein, Bradley, and Crandall [BBC00], and a method given in the Ph.D. dissertation of Michael Rubinstein [Rub98]. Rubinstein's dissertation considers the problem of computing values of a very general family of L-functions—his results apply to $\zeta(s)$ and to Dirichlet L-functions as special cases.

7.8 Acknowledgments

Much of the material in this chapter has appeared in a paper published in the Proceedings of the AMS Conference on Dynamical, Spectral and Arithmetic Zeta-Functions [Gal01]. Both the anonymous referee of that paper, and my advisor Harold G. Diamond suggested improvements to that paper which have been incorporated into this chapter.

I also thank Andrew Odlyzko for providing guidance in the interpretation of [OS88].

Bibliography

Citations to the American Mathematical Society's *Mathematical Reviews* are given as an "MR number", e.g., "MR 94b:00012".

[AB02] A. O. L. Atkin and D. J. Bernstein, *Prime sieves using binary quadratic forms*, Dept. of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, 60607-7045. Preprint available at `http://pobox.com/~djb/papers/primesieves.dvi`, 2002.

[ABCZ01] Volker Augustin, Florin P. Boca, Cristian Cobeli, and Alexandru Zaharescu, *The h-spacing distribution between Farey points*, Math. Proc. Cambridge Philos. Soc. **131** (2001), no. 1, 23–38. MR 2002h:11017

[AHU75] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley Publishing Co., Reading, Mass., 1975. MR 54 #1706

[AKS02] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena, *PRIMES is in P*, Tech. report, Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, Kanpur-208106, INDIA, August 2002, `http://www.cse.iitk.ac.in/users/manindra/primality.ps`.

[AM93] A. O. L. Atkin and F. Morain, *Elliptic curves and primality proving*, Math. Comp. **61** (1993), no. 203, 29–68. MR 93m:11136

[Apo76] Tom M. Apostol, *Introduction to analytic number theory*, Springer-Verlag, New York, 1976, Undergraduate Texts in Mathematics. MR 55 #7892

[APR83] Leonard M. Adleman, Carl Pomerance, and Robert S. Rumely, *On distinguishing prime numbers from composite numbers*, Ann. of Math. (2) **117** (1983), no. 1, 173–206. MR 84e:10008

[AS92] Milton Abramowitz and Irene A. Stegun (eds.), *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, Dover Publications Inc., New York, 1992, Reprint of the 1972 edition. MR 94b:00012

[BB87] Jonathan M. Borwein and Peter B. Borwein, *Pi and the AGM, a study in analytic number theory and computational complexity*, John Wiley & Sons Inc., New York, 1987, A Wiley-Interscience Publication. MR 89a:11134

[BBC00] Jonathan M. Borwein, David M. Bradley, and Richard E. Crandall, *Computational strategies for the Riemann zeta function*, J. Comput. Appl. Math. **121** (2000), no. 1-2, 247–296, Numerical analysis in the 20th century, Vol. I, Approximation theory. MR 2001h:11110

[BCZ00] Florin P. Boca, Cristian Cobeli, and Alexandru Zaharescu, *Distribution of lattice points visible from the origin*, Comm. Math. Phys. **213** (2000), no. 2, 433–470. MR 2001j:11094

[BCZ01] ———, *A conjecture of R. R. Hall on Farey points*, J. Reine Angew. Math. **535** (2001), 207–236. MR 2002f:11127

[BGZ03] Florin P. Boca, Radu N. Gologan, and Alexandru Zaharescu, *The statistics of the trajectory of a certain billiard in a flat two-torus*, Comm. Math. Phys. **240** (2003), no. 1-2, 53–73. MR 2 004 979

[BH77] Carter Bays and Richard H. Hudson, *The segmented sieve of Eratosthenes and primes in arithmetic progressions to* 10^{12}, Nordisk Tidskr. Informationsbehandling (BIT) **17** (1977), no. 2, 121–127. MR 56 #5405

[BK92] M. V. Berry and J. P. Keating, *A new asymptotic representation for* $\zeta(\frac{1}{2}+it)$ *and quantum spectral determinants*, Proc. Roy. Soc. London Ser. A **437** (1992), no. 1899, 151–173. MR 93j:11057

[BL99] J. Basney and M. Livny, *Deploying a high throughput computing cluster*, High Performance Cluster Computing (Rajkumar Buyya, ed.), vol. 1, Prentice Hall, May 1999.

[Bor03] Folkmar Bornemann, *PRIMES is in P: a breakthrough for "Everyman"*, Notices Amer. Math. Soc. **50** (2003), no. 5, 545–552. MR 2004c:11237

[Bre73] Richard P. Brent, *The first occurrence of large gaps between successive primes*, Math. Comp. **27** (1973), 959–963. MR 48 #8360

[Buc37] A. A. Buchstab, *An asymptotic estimation of a general number-theoretic function*, Mat. Sbordnik (2) **44** (1937), 1239–1246.

[CL84] H. Cohen and H. W. Lenstra, Jr., *Primality testing and Jacobi sums*, Math. Comp. **42** (1984), no. 165, 297–330. MR 86g:11078

[Coh93] Henri Cohen, *A course in computational algebraic number theory*, Springer-Verlag, Berlin, 1993. MR 94i:11105

[CON] *Condor project homepage*, `http://www.cs.wisc.edu/condor`.

[CR68] C. Chiarella and A. Reichel, *On the evaluation of integrals related to the error function*, Math. Comp. **22** (1968), 137–143. MR 36 #6117

[Cra96] Richard E. Crandall, *Topics in advanced scientific computation*, Springer-Verlag, New York, 1996. MR 97g:65005

[Cra02] ———, personal e-mail communication, April 2002.

[Deu67] Max Deuring, *Asymptotische Entwicklungen der Dirichletschen L-Reihen*, Math. Ann. **168** (1967), 1–30. MR 35 #4173

[DR96a] M. Deléglise and J. Rivat, *Computing $\pi(x)$: the Meissel, Lehmer, Lagarias, Miller, Odlyzko method*, Mathematics of Computation **65** (1996), no. 213, 235–245. MR 96d:11139

[DR96b] ———, *Some new values of $\pi(x)$*, e-mail, June 1996.

[DS98] Kevin Dowd and Charles R. Severance, *High performance computing*, second ed., O'Reilly and Associates, Inc., 101 Morris Street, Sebastopol, CA 95472, 1998.

[Edw74] H. M. Edwards, *Riemann's zeta function*, Pure and Applied Mathematics, Vol. 58, Academic Press, New York, 1974. MR 57 #5922

[G$^+$] Torbjörn Granlund et al., *The GNU multiple precision arithmetic library*, At `http://swox.com/gmp`.

[Gab79] Wolfgang Gabcke, *Neue Herleitung und Explizite Restabschätzung der Riemann-Siegel-Formel*, Ph.D. thesis, Georg-August-Universität zu Göttingen, 1979.

[Gal98] William F. Galway, *Robert Bennion's "hopping sieve"*, Algorithmic Number Theory (ANTS-III) (Joe P. Buhler, ed.), Lecture Notes in Computer Science, vol. 1423, Springer, Berlin, June 1998, pp. 169–178. MR 2000m:11124

[Gal00] ______, *Dissecting a sieve to cut its need for space*, Algorithmic number theory (ANTS-IV, Leiden, 2000) (Wieb Bosma, ed.), Lecture Notes in Computer Science, vol. 1838, Springer, Berlin, 2000, pp. 297–312. MR 2002g:11176

[Gal01] ______, *Computing the Riemann zeta function by numerical quadrature*, Dynamical, spectral, and arithmetic zeta functions (San Antonio, TX, 1999) (Providence, RI) (Machiel van Frankenhuysen and Michel L. Lapidus, eds.), Contemporary Mathematics, vol. 290, American Mathematical Society, 2001, pp. 81–91. MR 2002i:11131

[Gou01a] Xavier Gourdon, *Computation of pi(x) : improvements to the Meissel, Lehmer, Lagarias, Miller, Odlyzko, Deléglise and Rivat method*, Preprint available at `http://numbers.computation.free.fr/Constants/Primes/Pix/piNalgorithm.ps`, February 2001.

[Gou01b] ______, *The pi(x) project*, Webpage at `http://numbers.computation.free.fr/Constants/Primes/Pix/pixproject.html`, 2001.

[Har33] G. H. Hardy, *A theorem concerning Fourier transforms*, J. London Math. Soc. (1933), 227–231.

[Hen88] Peter Henrici, *Applied and computational complex analysis. Vol. 1*, John Wiley & Sons Inc., New York, 1988, Power series—integration—conformal mapping—location of zeros, Reprint of the 1974 original. MR 90d:30002

[Hen91] ———, *Applied and computational complex analysis. Vol. 2*, John Wiley & Sons Inc., New York, 1991, Special functions—integral transforms—asymptotics—continued fractions, Reprint of the 1977 original. MR 93b:30001

[HP90] John L. Hennessy and David A. Patterson, *Computer architecture: a quantitative approach*, Morgan Kaufmann Publishers, Inc., San Mateo, California, 1990.

[HR74] H. Halberstam and H.-E. Richert, *Sieve methods*, Academic Press, New York, 1974, London Mathematical Society Monographs, No. 4. MR 54 #12689

[Hux96] M. N. Huxley, *Area, lattice points, and exponential sums*, London Mathematical Society Monographs, New Series, vol. 13, The Clarendon Press, Oxford University Press, New York, 1996, Oxford Science Publications. MR 97g:11088

[HW79] G. H. Hardy and E. M. Wright, *An introduction to the theory of numbers*, fifth ed., Oxford University Press, Oxford, 1979. MR 8li:10002

[Ivi85] Aleksandar Ivić, *The Riemann zeta-function*, John Wiley & Sons, New York, 1985. MR 87d:11062

[KK68] Granino A. Korn and Theresa M. Korn, *Mathematical handbook for scientists and engineers*, McGraw-Hill Book Co., New York, 1968. MR 36 #3618

[Knu73] Donald E. Knuth, *The art of computer programming*, vol. 3: Sorting and searching, Addison-Wesley Publishing Co., Reading, Mass., 1973, Addison-Wesley Series in Computer Science and Information Processing. MR 56 #4281

[Knu75] ______, *The art of computer programming*, second ed., vol. 1: Fundamental Algorithms, Addison-Wesley Publishing Co., Reading, Mass., 1975, Addison-Wesley Series in Computer Science and Information Processing. MR 51 #14624

[Knu81] ______, *The art of computer programming*, second ed., vol. 2: Seminumerical algorithms, Addison-Wesley Publishing Co., Reading, Mass., 1981, Addison-Wesley Series in Computer Science and Information Processing. MR 83i:68003

[Leh66] R. Sherman Lehman, *On the difference* $\pi(x) - \mathrm{li}(x)$, Acta Arithmetica **11** (1966), 397–410. MR 34 #2546

[LL74] D. H. Lehmer and Emma Lehmer, *A new factorization technique using quadratic forms*, Mathematics of Computation **28** (1974), 625–635. MR 49 #7204

[LMO85] J. C. Lagarias, V. S. Miller, and A. M. Odlyzko, *Computing* $\pi(x)$*: The Meissel-Lehmer method*, Mathematics of Computation **44** (1985), no. 170, 537–560. MR 86h:11111

[LO84] J. C. Lagarias and A. M. Odlyzko, *New algorithms for computing* $\pi(x)$, Number theory (New York, 1982), Lecture Notes in Mathematics, vol. 1052, Springer-Verlag, New York, 1984, pp. 176–193. MR 85j:11182

[LO87] ______, *Computing* $\pi(x)$*: an analytic method*, Journal of Algorithms **8** (1987), no. 2, 173–191. MR 88k:11095

[McG01] Catherine C. McGeoch, *Experimental analysis of algorithms*, Notices of the American Mathematical Society **48** (2001), no. 3, 304–311.

[Mih98] Preda Mihăilescu, *Cyclotomy primality proving—recent developments*, Algorithmic number theory (Portland, OR, 1998), Lecture Notes in Comput. Sci., vol. 1423, Springer, Berlin, 1998, pp. 95–110. MR 2000j:11195

[Mor98] F. Morain, *Primality proving using elliptic curves: an update*, Algorithmic number theory (Portland, OR, 1998), Lecture Notes in Comput. Sci., vol. 1423, Springer, Berlin, 1998, pp. 111–127. MR 2000i:11190

[MV73] H. L. Montgomery and R. C. Vaughan, *The large sieve*, Mathematika **20** (1973), 119–134. MR 51 #10260

[OS88] A. M. Odlyzko and A. Schönhage, *Fast algorithms for multiple evaluations of the Riemann zeta function*, Trans. Amer. Math. Soc. **309** (1988), no. 2, 797–809. MR 89j:11083

[Pin00] Richard G. E. Pinch, *The pseudoprimes up to* 10^{13}, Algorithmic Number Theory (ANTS-IV) (Wieb Bosma, ed.), Lecture Notes in Computer Science, vol. 1838, Springer, Berlin, July 2000, pp. 459–473. MR 2002g:11177

[Pom81] Carl Pomerance, *On the distribution of pseudoprimes*, Math. Comp. **37** (1981), no. 156, 587–593. MR 83k:10009

[Pri81] Paul Pritchard, *A sublinear additive sieve for finding prime numbers*, Comm. ACM **24** (1981), no. 1, 18–23. MR 82c:10011

[Pri82] ———, *Explaining the wheel sieve*, Acta Inform. **17** (1982), no. 4, 477–485. MR 84g:10015

[PSW80] Carl Pomerance, J. L. Selfridge, and Samuel S. Wagstaff, Jr., *The pseudoprimes to* $25 \cdot 10^9$, Mathematics of Computation **35** (1980), no. 151, 1003–1026. MR 82g:10030

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical recipes in C*, second ed., Cambridge University Press, Cambridge, 1992. MR 93i:65001b

[Rie90] Bernhard Riemann, *Ueber die Anzahl der Primzahlen unter einer gegebenen Grösse*, Gesammelte mathematische Werke, wissenschaftlicher Nachlass und Nachträge (Raghavan Narasimhan, ed.), Springer-Verlag, Berlin, 1990, Based on the edition by Heinrich Weber and Richard Dedekind, Edited and with a preface by Raghavan Narasimhan, pp. 177–185. MR 91j:01070b

[Rie94] Hans Riesel, *Prime numbers and computer methods for factorization*, second ed., Progress in Mathematics, vol. 126, Birkhäuser, Boston, MA, 1994. MR 95h:11142

[Rub98] Michael Oded Rubinstein, *Evidence for a spectral interpretation of the zeros of L-functions*, Ph.D. thesis, Princeton University, June 1998.

[Sie66] C. L. Siegel, *Über Riemanns Nachlaß zur analytischen Zahlentheorie*, Gesammelte Abhandlungen, vol. 1, Springer-Verlag, New York, 1966.

[Sie74] Wacław Sierpiński, *Sur un problème du calcul des fonctions asymptotiques, 1906*, Oeuvres choisies, Tome I, PWN—Éditions Scientifiques de Pologne, Warsaw, 1974, pp. 73–108.

[Sor98] Jonathan P. Sorenson, *Trading time for space in prime number sieves*, Algorithmic Number Theory (ANTS-III) (Joe P. Buhler, ed.), Lecture Notes in Computer Science, vol. 1423, Springer, Berlin, June 1998, pp. 179–195. MR 2000i:11191

[Spo94] John L. Spouge, *Computation of the gamma, digamma, and trigamma functions*, SIAM J. Numer. Anal. **31** (1994), no. 3, 931–944. MR 95g:33002

[Ste93] Frank Stenger, *Numerical methods based on sinc and analytic functions*, Springer Series in Computational Mathematics, vol. 20, Springer-Verlag, New York, 1993. MR 94k:65003

[Tem77] Nico M. Temme, *The numerical computation of special functions by use of quadrature rules for saddle point integrals. I. Trapezoidal integration rules*, Tech. Report TW 164/77, Mathematisch Centrum, Afdeling Toegepaste Wiskunde, Amsterdam, 1977. MR 57 #4483

[Ten95] Gérald Tenenbaum, *Introduction to analytic and probabilistic number theory*, Cambridge University Press, Cambridge, 1995. MR 97e:11005b

[Tit86] E. C. Titchmarsh, *The theory of the Riemann zeta-function*, second ed., The Clarendon Press Oxford University Press, New York, 1986, Edited and with a preface by D. R. Heath-Brown. MR 88c:11049

[vdC20] J. G. van der Corput, *Über Gitterpunkte in der Ebene*, Mathematische Annalen **81** (1920), 1–20.

[Vet91] Ekkehart Vetter, *Algorithmen zur Berechnung von* $\pi(x)$, Diplomarbeit, Rheinische Friedrich-Wilhelms-Universität Bonn, January 1991.

[Vor03] Georges Voronoï, *Sur un problème du calcul des fonctions asymptotiques*, J. Reine Angew. Math. **126** (1903), 241–282.

[Wil98] Hugh C. Williams, *Édouard Lucas and primality testing*, John Wiley & Sons Inc., New York, 1998, A Wiley-Interscience Publication. MR 2000b:11139

Index and Glossary

Vita

William Floyd Galway was born September 7, 1952, in Salt Lake City, Utah.

He attended the University of Utah from 1970 to 1976, receiving a B.A. in Mathematics. From 1980 to 1984, he attended graduate school at the University of Utah and earned his M.S. in Computer Science. His master's thesis was titled: *A Generic, Interactive Editor for Portable Standard Lisp.*

Before receiving his master's degree, he spent several years working as a computer consultant. After receiving his degree he worked as a senior systems programmer at the University of Utah and later spent two years in Bath, England, programming computers as a research officer at the School of Mathematics at the University of Bath. He also continued work as a computer consultant.

In the late 1970s, while working as a consultant for the Hospal Medical Corporation Research Center, he developed software to measure the lung capacity of premature infants, as described in the paper *Computerized Estimates of Functional Residual Capacity in Infants*, which he coauthored in 1981 with Peter Richardson, Steven Olsen, and J. Bert Bunnell. Working with D. John Anderson in the early 1990s as a consultant for Storm Technology, Inc., he coauthored a program to test image files for conformance to the ISO 10918-1 JPEG Draft International Standard, which had recently been developed by the Joint Photographic Experts Group.

He returned to a career in mathematics when in 1993 he entered the Ph.D. program at the University of Illinois at Urbana-Champaign. While at Illinois, in 1999 he received a Hohn-Nash Award "for outstanding scholarship and promise in applied mathematics, computational science, or scientific computing." After leaving Illinois in 2001, he spent two years in Canada as a Fellow at the Pacific Institute for Mathematical Sciences (PIMS) at Simon Fraser University, in Burnaby, British Columbia.

He currently lives in New Jersey with his wife, Miriam Paschetto, who is a structural engineer. In his spare time he enjoys hiking, bicycle touring, playing the recorder, and choral singing.