THE RIEMANN-SIEGEL FORMULA AND LARGE SCALE COMPUTATIONS OF THE RIEMANN ZETA FUNCTION

by

GLENDON RALPH PUGH

B.Sc., University of New Brunswick, 1992

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

Mathematics

We accept this thesis as conforming
to the required standard

. . . .

. . . .

. . . .

. . . .

. . . .

THE UNIVERSITY OF BRITISH COLUMBIA

December 1998

Abstract

This thesis is a survey of the derivation and implementation of the Riemann-Siegel formula for computing values of Riemann's zeta function on the line $s = 1/2 + it$. The formula, devised by Riemann and later published by Siegel following study of Riemann's unpublished work, is the method of choice for both numerically verifying the Riemann Hypothesis and locating zeros on the critical line at large values of t.

Simply stated, the Riemann Hypothesis is that all of the zeros of $\zeta(s)$ in the strip $0 < \Re(s) < 1$ lie on the line $\Re(s) = 1/2$. Since Riemann made his conjecture in 1859, much work has been done towards developing efficient numerical techniques for verifying the hypothesis and possibly finding counter-examples.

This thesis is meant to serve as a guide book for using Riemann-Siegel. It is mostly a distillation of work done in the field since Siegel's results published in the early 1930's. Computer programs and examples are included, and error bounds are discussed. The question of how and why Riemann-Siegel is used to verify the Riemann Hypothesis is examined, and a detailed Riemann Hypothesis verification example is illustrated. Finally, recent work in the field is noted.

The derivation of the Riemann-Siegel formula for computing $\zeta(1/2 + it)$ is based on the saddle point method of evaluating integrals, and yields results of considerable accuracy in time $t^{1/2}$. The saddle point method is an approximation technique which concentrates the "bulk" of an integral on a path through a point at which the modulus of the integrand is a maximum.

Table of Contents

Acknowledgement

I would like to thank all those who have assisted, guided and supported me in my studies leading to this thesis. In particular, I wish to thank my supervisor Dr. Bill Casselman for his guidance and encouragement, Dr. David Boyd who kindly agreed to read my thesis at a busy time of year, and Dr. Leah Keshet and Ms. Helga Diersen for their assistance as I found my way through the administrative maze.

I would also like to thank my friend Michael Fellows for introducing me to the Linux operating system on which all programming and typesetting tasks were performed, and I wish to extend this thanks to the Linux community as a whole for giving students home access to high quality, low cost workstation-grade computing.

Finally, I would like to thank my family for their continual support and encouragement: my parents Anne and Charles, my brother John who grappled with his own thesis, and especially my wife Jacqueline for her help and understanding during my countless hours spent in front of the computer over the past year.

Chapter 1
Introduction

In 1859, the German mathematician Bernhard Riemann published his now famous paper on his studies of prime numbers. This eight page paper, entitled *On the Number of Primes Less Than a Given Magnitude*, introduced many new results in the fields of analytic number theory and complex functions, but contained a number of statements which were unsupported by rigorous proofs. Proofs for many of these conjectures were found over the fifty or so years following Riemann's death in 1866; however one statement has withstood repeated assaults by several generations of mathematicians. This conjecture, known universally as the *Riemann Hypothesis*, is now among the most famous unsolved problems in mathematics.

The Riemann Hypothesis is a statement about the location of the complex zeros of *the zeta function*. Riemann introduced the zeta function $\zeta(s)$ for $s \in \mathbb{C}$ during the course of his investigations of prime numbers and he explored many of its properties. In his paper he made the statement that "it is very likely that" all non-trivial zeros of the zeta function must have real part equal to $1/2$, but no proof was provided. An excellent survey of Riemann's results and work done in the field since Riemann's paper is Harold Edwards' treatise *Riemann's Zeta Function*[1].

The zeta function and related hypothesis are classical fields of study which in recent years have enjoyed renewed interest. Conjectured relationships between the hypothesis and results in mathematical physics have motivated extensive numerical investigations using modern supercomputers. Van de Lune et al. have demonstrated the correctness of the Riemann Hypothesis for the first one and a half billion non-trivial zeros of the zeta function[2], while Andrew Odlyzko has verified the correctness of the hypothesis for large sets of zeros around the 10^{20}th zero[5].

The modern studies of the Riemann Hypothesis use variants of a computational algorithm known as the *Riemann-Siegel formula*. This approximation algorithm permits very fast evaluation of the zeta function, and the accuracy of the approximations of $\zeta(1/2+it)$ improves with increasing t. However, eventually the machine floating point representation error imposed by the particular computer architecture limits the validity of the calculations. This accuracy problem can be overcome using extended precision calculation software, but the loss in computation speed makes calculations at complex numbers of even moderately large modulus infeasible. Despite these limitations, the Riemann-Siegel formula is still a vast improvement over methods previously available.

The Riemann-Siegel formula is a remarkable approximation derived and used by Riemann, but never published by him. It was not until 1932 that Carl Siegel uncovered the formula following a study of Riemann's unpublished notes. Prior to Siegel's discovery, the method of choice was Euler-Maclaurin summation which requires on the order of t steps to evaluate $\zeta(1/2+it)$ (here $t \in \mathbb{R}$). The Riemann-Siegel formula, on the other hand, requires only $\sqrt{t}$ steps.

The purpose of this thesis is to examine the Riemann-Siegel formula and understand how it is used to verify the correctness of the Riemann Hypothesis. It is meant to serve as a user's manual, providing the details of the method's derivation and limitations, as well as examples and computer programs for its implementation.

The next chapter begins with an overview of the Riemann zeta function. In that chapter we mention some familiar properties of the function and derive a global representation of it as done by Riemann himself. Some additional results which will be required in subsequent chapters are then noted. The third chapter contains the development of the Riemann-Siegel formula. An example of its implementation is presented followed by a discussion of errors inherent in the method. The fourth chapter examines the Riemann Hypothesis verification process. Major contributions by J. Gram and A. Turing are presented, and a curious behaviour of $\zeta(1/2+it)$ at large values of t is noted. In chapter five we put into practice the concepts developed in the first four chapters and undertake a detailed verification of the Riemann Hypothesis for t in the range 0 to 6 000 000. The sixth and final chapter contains concluding remarks and some suggestions for future investigations.

Chapter 2

Overview of the Riemann Zeta Function

2.1 Introduction

The point of this paper is to examine a particular method of evaluating the Riemann zeta function. This method, the Riemann-Siegel formula, is the method of choice for verifying the Riemann Hypothesis. In this chapter I begin by introducing the zeta function and discussing some of its familiar properties. Following this we will develop the global representation of the zeta function $\zeta(s)$ as Riemann originally derived it, and at the end of the chapter we will derive a decomposition of the zeta function which will be needed in the development of the Riemann-Siegel formula.

2.2 The Function $\zeta(s)$

For $s \in \mathbb{C}$ with $\Re(s) > 1$, the function $\zeta(s)$ is defined by

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s} \ .$$

This sum converges for $\Re(s) > 1$ and diverges otherwise. For each $\delta > 0$, convergence is uniform for $\Re(s) \geq 1 + \delta$.

Some well known values of this function are $\zeta(2) = \pi^2/6$, $\zeta(4) = \pi^4/90$. In general, for n a non-negative integer, $\zeta(2n) = (2\pi)^{2n} (-1)^{n+1} B_{2n}/ ((2n)!2)$ where B_k are the

Bernoulli numbers defined by the expansion $x/\left(e^x-1\right)=\sum_{n=0}^{\infty} B_n x^n/n!$. There is no known closed form expression for $\zeta(2n+1)$.

The ζ function so defined is related to the study of prime numbers by

$$\sum_{n=1}^{\infty}\frac{1}{n^s}=\prod_p\left(1-p^{-s}\right)^{-1}$$

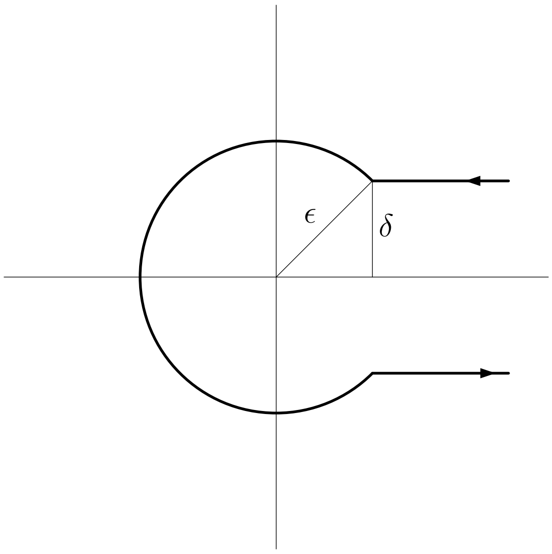
where p ranges over the prime numbers and $\Re(s)>1$. This identity can be shown by writing $\left(1-p^{-s}\right)^{-1}=\sum_{n=0}^{\infty}\left(1/p^s\right)^n$ for each p and expanding the product of these series.

2.3 Global Representation of $\zeta(s)$

The definition of the ζ function given in Section 2.2 is valid only for $\Re(s)>1$. The following definition extends ζ to all points $s\in\mathbb{C}\setminus\{1\}$:

$$\zeta(s)=\frac{\Pi(-s)}{2\pi i}\int_{C_{\epsilon,\delta}}\frac{(-x)^s}{\left(e^x-1\right)x}\,dx\ , \tag{2.1}$$

where $C_{\epsilon,\delta}$ is the contour



.

Here the definition of $(-x)^s$ is $e^{s \log(-x)}$ with $-\pi < \Im(\log(-x)) < \pi$.

The size of δ above may be as small as we please — the equivalence of (2.1) under different choices of δ is guaranteed by Cauchy's Theorem and the dominance away from the origin of the exponential term of the integrand. Further, Cauchy's Theorem also permits ϵ, the size of the radius of the path about the origin, to be any value between 0 and the radius at which the first singularity of the integrand occurs, 2π. These permitted variations in the contour $C_{\epsilon,\delta}$ will be used in the derivation to follow.

A standard definition of the global representation of the zeta function is one which uses the limiting contour as $\delta \to 0$ with $\epsilon = 1$, which is expressed thus:

$$\zeta(s) = \frac{\Pi(-s)}{2\pi i} \int_{+\infty}^{+\infty} \frac{(-x)^s}{(e^x - 1)\,x}\, dx \ .$$

The limits of integration are meant to indicate a path which starts at $+\infty$, descends parallel and *just above* the real-axis, circles the origin once in the positive direction, and returns to $+\infty$ parallel and *just below* the real-axis. Note that this path must not be considered to coincide with the positive real axis since $\log(-x)$ is not defined for $x \in \mathbb{R}_+$. We will express this version of the definition more concretely as

$$\zeta(s) \quad = \quad \frac{\Pi(-s)}{2\pi i} \int_{C_{1,0}} \frac{(-x)^s}{(e^x - 1)\,x}\, dx \quad = \quad \frac{\Pi(-s)}{2\pi i} \lim_{\delta \to 0} \int_{C_{1,\delta}} \frac{(-x)^s}{(e^x - 1)\,x}\, dx \ . \tag{2.2}$$

We will devote the remainder of this section to the derivation of (2.1). To begin, we consider the integral definition of the factorial function with argument $s \in \mathbb{R}$, $s > 1$:

$$\Pi(s-1) = \int_0^\infty e^{-x} x^{s-1}\, dx, \qquad s > 1 \ . \tag{2.3}$$

Here $\Pi(s-1)$ is used in place of the more familiar $\Gamma(s)$. This notation of Gauss for the extended factorial function was used by Riemann and is retained by Edwards[1] in his work, and we will do the same here. Note that $\Pi(n)$ coincides with $n!$ for n a positive integer. Now fix an integer $K > 0$ and set $x = nt$ in the integral (2.3), so that

$$\Pi(s-1) = \int_0^\infty e^{-nt}\,(nt)^{s-1}\,n\,dt\ .$$

From this we may develop

$$\begin{aligned}
\Pi(s-1)\,n^{-s} &= \int_0^\infty e^{-nt}t^{s-1}\,dt \\
\Rightarrow \sum_{n=1}^{K}\Pi(s-1)\,n^{-s} &= \sum_{n=1}^{K}\int_0^\infty e^{-nt}t^{s-1}\,dt \\
\Rightarrow \Pi(s-1)\sum_{n=1}^{K}n^{-s} &= \int_0^\infty \left(\sum_{n=1}^{K}e^{-nt}\right)t^{s-1}\,dt \\
\Rightarrow \Pi(s-1)\sum_{n=1}^{K}n^{-s} &= \int_0^\infty \left(\frac{1-e^{-Kt}}{e^t-1}\right)t^{s-1}\,dt\ . \qquad (2.4)
\end{aligned}$$

Observe that for each $t > 0$, the integrand in (2.4) is a positive function increasing with K to the $L^1(dt)$ function $t^{s-1}/\,(e^t-1)$. Thus we may apply the Monotone Convergence Theorem to yield

$$\begin{aligned}
\lim_{K\to\infty}\Pi(s-1)\sum_{n=1}^{K}n^{-s} &= \lim_{K\to\infty}\int_0^\infty \left(\frac{1-e^{-Kt}}{e^t-1}\right)t^{s-1}\,dt \\
&= \int_0^\infty \lim_{K\to\infty}\left(\frac{1-e^{-Kt}}{e^t-1}\right)t^{s-1}\,dt \\
&= \int_0^\infty \frac{t^{s-1}}{e^t-1}\,dt\ .
\end{aligned}$$

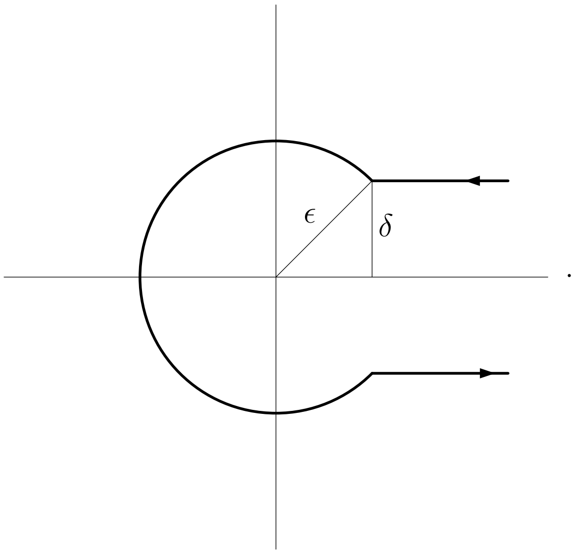
That is,

$$\Pi(s-1)\sum_{n=1}^{\infty}n^{-s} = \int_0^\infty \frac{t^{s-1}}{e^t-1}\,dt\ . \qquad (2.5)$$

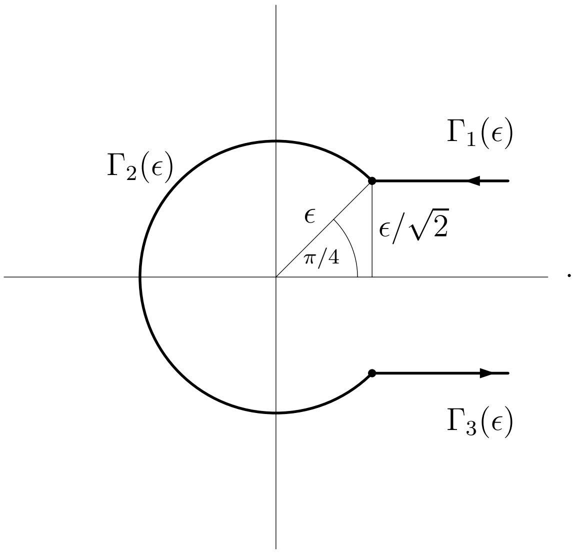
We now turn our attention to the integral in (2.1):

$$\int_{C_{\epsilon,\delta}} \frac{(-x)^s}{(e^x-1)\,x}\,dx\ , \qquad (2.6)$$

where $C_{\epsilon,\delta}$ was



Again, for the moment assume only that $s \in \mathbb{R}$, $s > 1$. As noted following definition (2.1), the paths $C_{\epsilon,\delta}$ may be altered to suit our purpose—equality of the integrals over the different paths is guaranteed by Cauchy's Theorem. We will consider the set of contours $C_{\epsilon,\delta(\epsilon)}$ where $\delta(\epsilon) = \epsilon/\sqrt{2}$:



Since the value of the integral is invariant under decreasing ϵ, (2.6) is equal to

$$\lim_{\epsilon \to 0} \int_{C_{\epsilon,\delta(\epsilon)}} \frac{(-x)^s}{(e^x - 1)\, x} \, dx \ .$$

To evaluate this last expression, fix an ϵ sufficiently small and consider the integral over each of the paths $\Gamma_1(\epsilon)$, $\Gamma_2(\epsilon)$ and $\Gamma_3(\epsilon)$. Writing $x = t + i\epsilon$, the integral over $\Gamma_1(\epsilon)$ may be written

$$\int_{+\infty}^{0} \frac{(-t - i\epsilon)^s}{(e^{t+i\epsilon} - 1)\,(t + i\epsilon)} I_\epsilon \, dt \tag{2.7}$$

where I_ϵ is the indicator function of the set $\left[\epsilon/\sqrt{2}, \infty\right)$. Now

$$\lim_{\epsilon \to 0} \left[\frac{(-t - i\epsilon)^s}{(e^{t+i\epsilon} - 1)\,(t + i\epsilon)} I_\epsilon \right]$$

exists for each t, and

$$\left| \frac{(-t - i\epsilon)^s}{(e^{t+i\epsilon} - 1)\,(t + i\epsilon)} I_\epsilon \right| \ \leq \ \frac{(t^2 + \epsilon^2)^{\frac{s-1}{2}}}{e^t - 1} \ \leq \ 2^s \frac{t^{s-1}}{e^t - 1},$$

where

$$2^s \int_0^\infty \frac{t^{s-1}}{e^t - 1} \, dt \ = \ 2^s \Pi(s-1) \sum_{n=1}^\infty \frac{1}{n^s} \ < \ \infty.$$

The Dominated Convergence Theorem then allows us to conclude that the limit as $\epsilon \to 0$ of (2.7) exists. Indeed,

$$\begin{aligned}
\lim_{\epsilon \to 0} \int_{+\infty}^{0} \frac{e^{s \log\,(-t - i\epsilon)}}{(e^{t+i\epsilon} - 1)\ (t + i\epsilon)} I_\epsilon \, dt \ &= \ \lim_{\epsilon \to 0} \int_{+\infty}^{0} \frac{e^{s \log\,(t + i\epsilon) - i\pi}}{(e^{t+i\epsilon} - 1)\ (t + i\epsilon)} I_\epsilon \, dt \\
&= \ -e^{-i\pi s} \lim_{\epsilon \to 0} \int_0^{+\infty} \frac{e^{s \log\,(t + i\epsilon)}}{(e^{t+i\epsilon} - 1)\ (t + i\epsilon)} I_\epsilon \, dt \\
&= \ -e^{-i\pi s} \int_0^{+\infty} \frac{t^{s-1}}{(e^t - 1)} \, dt \ .
\end{aligned}$$

A similar argument can be used to show that as $\epsilon \to 0$, the integral over $\Gamma_3(\epsilon)$ becomes

$$e^{i\pi s} \int_0^{+\infty} \frac{t^{s-1}}{(e^t - 1)} \, dt \ .$$

For the integral over $\Gamma_2(\epsilon)$, write $x = -\epsilon e^{i\theta}$, $dx = -\epsilon i e^{i\theta} d\theta$. Then

$$\begin{aligned}
\int_{\Gamma_2(\epsilon)} \frac{(-x)^s}{(e^x - 1)\, x}\, dx &= \int_{-3\pi/4}^{3\pi/4} \frac{-\left(\epsilon e^{i\theta}\right)^s \epsilon i e^{i\theta}}{\left(e^{-\epsilon e^{i\theta}} - 1\right)\left(-\epsilon e^{i\theta}\right)}\, d\theta \\
&= i \int_{-3\pi/4}^{3\pi/4} \frac{\epsilon e^{i\theta s}}{\sum_{j=1}^{\infty} \left(-\epsilon e^{i\theta}\right)^j / j!}\, d\theta \\
&= i\epsilon^{s-1} \int_{-3\pi/4}^{3\pi/4} \frac{e^{i\theta\,(s-1)}}{\sum_{j=1}^{\infty} \left(-\epsilon e^{i\theta}\right)^{j-1} / j!}\, d\theta\ ,
\end{aligned}$$

and this last expression approaches zero as $\epsilon \to 0$.

Combining the integrals over the three paths then yields

$$\begin{aligned}
\lim_{\epsilon \to 0} \int_{C_{\epsilon,\delta(\epsilon)}} \frac{(-x)^s}{(e^x - 1)\, x}\, dx &= \left(e^{i\pi s} - e^{-i\pi s}\right) \int_0^{+\infty} \frac{t^{s-1}}{(e^t - 1)}\, dt \\
&= 2i \sin(\pi s)\, \Pi(s-1) \sum_{n=1}^{\infty} \frac{1}{n^s}\ . \qquad (2.8)
\end{aligned}$$

Taking note of the factorial function identity

$$\frac{\pi s}{\Pi(s)\, \Pi(-s)} = \sin(\pi s)\ , \qquad (2.9)$$

we may replace $\sin(\pi s)$ in (2.8) to obtain

$$\lim_{\epsilon \to 0} \int_{C_{\epsilon,\delta(\epsilon)}} \frac{(-x)^s}{(e^x - 1)\, x}\, dx = \frac{2i\pi s}{\Pi(s)\, \Pi(-s)} \Pi(s-1) \sum_{n=1}^{\infty} \frac{1}{n^s}$$

from which

$$\frac{\Pi(-s)}{2\pi i} \lim_{\epsilon \to 0} \int_{C_{\epsilon,\delta(\epsilon)}} \frac{(-x)^s}{(e^x - 1)\, x}\, dx = \sum_{n=1}^{\infty} \frac{1}{n^s} = \zeta(s)\ . \qquad (2.10)$$

For the purposes of our derivation, we considered the limit of the integral in (2.10) as $\epsilon \to 0$. However, as observed earlier in this section, this limit is equal to the integral over any of the equivalent paths $C_{\epsilon,\delta}$. Following the convention introduced in equation (2.2), let

$$\frac{\Pi(-s)}{2\pi i}\int_{C_{0,0}}\frac{(-x)^s}{(e^x-1)\,x}\,dx$$

denote the limit expression in (2.10). Then the statement of equivalence over different contours is

$$\frac{\Pi(-s)}{2\pi i}\int_{C_{0,0}}\frac{(-x)^s}{(e^x-1)\,x}\,dx=\frac{\Pi(-s)}{2\pi i}\int_{C_{\epsilon,\delta}}\frac{(-x)^s}{(e^x-1)\,x}\,dx\ ,$$

and (2.10) may be expressed more generally as

$$\frac{\Pi(-s)}{2\pi i}\int_{C_{\epsilon,\delta}}\frac{(-x)^s}{(e^x-1)\,x}\,dx\quad=\quad\sum_{n=1}^{\infty}\frac{1}{n^s}\quad=\quad\zeta(s)\ . \tag{2.11}$$

At this point we may make two observations:

1. The $\Pi(-s)$ factor in (2.11) has singularities when s is a positive integer. The right hand side of (2.11), on the other hand, is defined at all such points except for $s=1$. Thus the the integral expression in (2.11) must have zeros at the positive integers greater than 1. Indeed, this fact follows immediately from (2.8).

2. Although we have derived (2.11) under the assumption that $s\in\mathbb{R}$, $s>1$, for each $\epsilon\delta$ path the integral in (2.11) converges for all $s\in\mathbb{C}$. In fact, the function

$$\int_{C_{\epsilon,\delta}}\frac{(-x)^s}{(e^x-1)\,x}I_{\{\Re(x)<N\}}\,dx$$

is an analytic function of s, and the convergence as $N\to\infty$ to

$$\int_{C_{\epsilon,\delta}}\frac{(-x)^s}{(e^x-1)\,x}\,dx$$

is uniform. As such, the left hand side of (2.11) is defined and analytic at all points $s\in\mathbb{C}\setminus 1$. At $s=1$ we note that $\Pi(-s)$ has a simple pole. Hence (2.11) with $s\in\mathbb{C}\setminus 1$ is our required global definition of the zeta function.

2.4 Properties of the Zeta Function

Riemann used his global definition of the zeta function to derive many interesting relations. Two of these which are relevant to the verification of the Riemann Hypothesis are the *functional equation of the zeta function* and the function $\xi(s)$.

The functional equation of the zeta function is

$$\zeta(s) = \Pi(-s)\ (2\pi)^{s-1}\, 2\sin(s\pi/2)\, \zeta(1-s)\ \ ,$$

which Riemann derives by considering the evaluation of (2.10) at negative real values of s. Using this equation one can immediately demonstrate that $\zeta(-2k) = 0$ for $\mathrm{k} = 1, 2, \ldots$.

Applying (2.9) and another familiar factorial identity

$$\Pi(s) = 2^s \Pi\left(\frac{s}{2}\right) \Pi\left(\frac{s-1}{2}\right) \pi^{-1/2} \tag{2.12}$$

to the functional equation puts it in the form

$$\Pi\left(\frac{s}{2}-1\right) \pi^{-s/2}\zeta(s) = \Pi\left(\frac{1-s}{2}-1\right) \pi^{-\,(1-s)/2}\zeta(1-s)\ \ .$$

Observe that this equation remains unchanged under the transformation $s \mapsto 1-s$.

The function $\xi(s)$ is defined by

$$\xi(s) = \Pi\left(\frac{s}{2}\right)\ (s-1)\, \pi^{-s/2}\zeta(s)\ \ .$$

The $s-1$ term in the ξ function definition eliminates the simple pole of ζ at $s=1$ so that $\xi(s)$ is an entire function. From the functional equation of the zeta function we have that $\xi(s) = \xi(1-s)$.

One important fact about $\xi(s)$ is that it is real when s lies on the line $1/2 + it$, $t \in \mathbb{R}$, (this set of points is called *the critical line*). This fact can be deduced as follows:

For $s \in \mathbb{R}$, $\xi(s) \in \mathbb{R}$. By the Schwartz reflection principle, $\overline{\xi(\overline{s})} = \xi(s)$, so that $\xi(\overline{s}) = \overline{\xi(s)}$. With $s = 1/2 + it$, t real, and using the functional equation we may then write

$$\xi(1/2+it) = \xi(1-\ (1/2+it)) = \xi\left(\overline{1/2+it}\right) = \overline{\xi(1/2+it)}\ . \tag{2.13}$$

So locating roots on the critical line reduces to locating sign changes of $\xi(1/2+it)$. Further analysis of $\xi(1/2+it)$ simplifies our task even further:

$$\xi(s) = \Pi\left(\frac{s}{2}\right)\ (s-1)\,\pi^{-s/2}\zeta(s) = \Pi\left(\frac{s}{2}-1\right)\frac{s\ (s-1)}{2}\pi^{-s/2}\zeta(s)\ ,$$

and substituting $s = 1/2 + it$ yields

$$\xi\left(\frac{1}{2}+it\right) = \left(e^{\Re\left[\log\Pi\left(\frac{it}{2}-\frac{3}{4}\right)\right]}\pi^{-1/4}\frac{-t^2-1/4}{2}\right) \times \left(e^{i\Im\left[\log\Pi\left(\frac{it}{2}-\frac{3}{4}\right)\right]}\pi^{-it/2}\zeta\left(\frac{1}{2}+it\right)\right)\ .$$

The point to notice is that the first term above is always negative, so that sign changes in $\xi(1/2+it)$ correspond to sign changes of the second term. This second term is denoted $Z(t)$, and the relationship between $Z(t)$ and $\zeta(1/2+it)$ is written

$$Z(t) = e^{i\vartheta(t)}\zeta\left(\frac{1}{2}+it\right)$$

where

$$\vartheta(t) = \Im\left[\log\Pi\left(\frac{it}{2}-\frac{3}{4}\right)\right] - \frac{t}{2}\log(\pi)\ .$$

The Riemann-Siegel formula is an approximation formula for $Z(t)$. Once $Z(t)$ is known, a straightforward approximation of $\vartheta(t)$ can then be used to easily compute $\zeta(1/2+it)$.

Chapter 3

Derivation of the Riemann-Siegel Formula

3.1 Introduction

Now that we have motivated our discussion by introducing the Riemann zeta function and stating the Riemann Hypothesis, let us develop a method for evaluation of the ζ function.

Prior to Carl Siegel's 1932 rediscovery of Riemann's method, the most widely used algorithm for evaluating the ζ function was *Euler-Maclaurin summation.* Euler-Maclaurin summation is essentially an approximation method for summing a series (finite or infinite) by splitting off a certain finite part and estimating the remainder. This method can be used to evaluate not only the ζ function, but many other functions that satisfy certain regularity conditions. Further, Euler Maclaurin Summation can be used to compute the values of the ζ function at all points of $\mathbb{C}\setminus\{1\}$ to any prescribed degree of accuracy. For evaluating $\zeta(1/2+it)$, Euler Maclaurin Summation requires on the order of $|t|$ steps for a single evaluation.

The Riemann-Siegel formula, on the other hand, is a very specialized method for evaluating the ζ function on the critical line. Riemann-Siegel is actually a formula for computing $Z(t)$ from which $\zeta(1/2+it)$ can easily be computed (see Section 2.4). The algorithm has been extended to accommodate points not on the critical line. However Riemann's derivation was concerned only with the more restricted set. Riemann-Siegel is a true approximation in the sense that certain contour integrals are truncated to permit their closed form evaluation, and as a result, small errors are introduced which are unrelated to the truncation of an asymptotic series. These error terms decrease very rapidly as $|1/2+it|$ increases, and therefore do not cause a serious problem. The strength of Riemann-Siegel is the speed. The evaluation of $\zeta(1/2+it)$ requires only order $\sqrt{|t|}$ steps, which allows for very efficient root counting algorithms. For the purposes of verifying the Riemann Hypothesis, this is of particular importance—we will see that it is not the precise location of a root which is important, but rather its existence.

In the sections that follow, the main formula is presented, followed by an overview of the derivation broken down into its key steps. An example and a discussion of errors is presented in the last two sections.

3.2 The Formula

The Riemann-Siegel formula for computing $\zeta(1/2+it)$, $t \in \mathbb{R}_+$, is the following (from [1]):

Set $N = \left\lfloor (t/2\pi)^{1/2} \right\rfloor$ (the integer part of $(t/2\pi)^{1/2}$), $p = (t/2\pi)^{1/2} - N$. Then

$$Z(t) = 2\sum_{n=1}^{N} n^{-1/2} \cos\left[\vartheta(t) - t\log n\right] + R$$

where

$$\vartheta(t) = \Im\left[\log \Pi\left(\frac{it}{2} - \frac{3}{4}\right)\right] - \frac{t}{2}\log\pi$$

and

$$R \approx (-1)^{N-1}\left(\frac{t}{2\pi}\right)^{-1/4}\left[C_0 + C_1\left(\frac{t}{2\pi}\right)^{-1/2} + C_2\left(\frac{t}{2\pi}\right)^{-2/2} + C_3\left(\frac{t}{2\pi}\right)^{-3/2} + C_4\left(\frac{t}{2\pi}\right)^{-4/2}\right]$$

with

$$
\begin{aligned}
C_0 &= \Psi(p) = \frac{\cos\left[2\pi\left(p^2 - p - 1/16\right)\right]}{\cos\left(2\pi p\right)}, \\
C_1 &= -\frac{1}{96\pi^2}\Psi^{(3)}(p)\,, \\
C_2 &= \frac{1}{18\,432\pi^4}\Psi^{(6)}(p) + \frac{1}{64\pi^2}\Psi^{(2)}(p)\,, \\
C_3 &= -\frac{1}{5\,308\,416\pi^6}\Psi^{(9)}(p) - \frac{1}{3840\pi^4}\Psi^{(5)}(p) - \frac{1}{64\pi^2}\Psi^{(1)}(p)\,, \\
C_4 &= \frac{1}{2\,038\,431\,744\pi^8}\Psi^{(12)}(p) + \frac{11}{5\,898\,240\pi^6}\Psi^{(8)}(p) + \frac{19}{24\,576\pi^4}\Psi^{(4)}(p) + \frac{1}{128\pi^2}\Psi(p) \;.
\end{aligned}
$$

As noted previously, $\Pi(x)$ is equal to $\Gamma(x+1)$. Also, $\vartheta(t)$ can be approximated using Stirling's series:

$$
\vartheta(t) = \frac{t}{2}\log\frac{t}{2\pi} - \frac{t}{2} - \frac{\pi}{8} + \frac{1}{48t} + \frac{7}{5760t^3} + \cdots \tag{3.1}
$$

which, for large t, has a small error even if truncated at only a few terms.

The derivation to follow will show how to compute as many of the $C_j\left[t/\left(2\pi\right)\right]^{-j/2}$ terms as desired; Riemann himself developed the formula with 5.

Once $Z(t)$ and $\vartheta(t)$ are known, $\zeta(1/2+it)$ is computed as

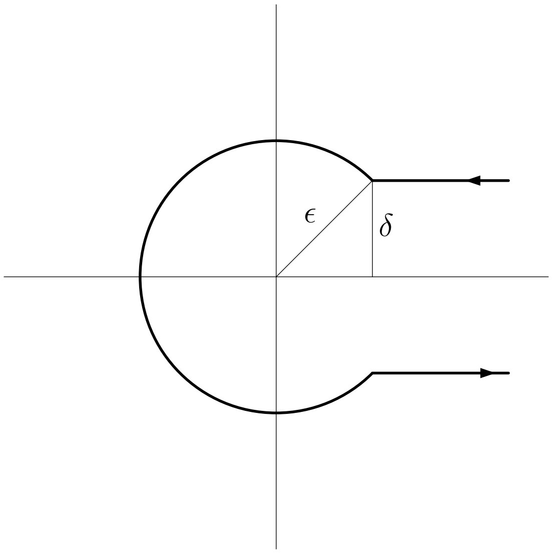
$$
\zeta\left(\frac{1}{2}+it\right) = Z(t)\,e^{-i\vartheta(t)} \;.
$$

3.3 Derivation of $Z(t)$

We begin with the global form of $\zeta(s)$ derived in Section 2.3:

$$\zeta(s) \quad = \quad \frac{\Pi(-s)}{2\pi i} \int_{C_{\epsilon,\delta}} \frac{(-x)^s}{(e^x - 1)\,x}\,dx$$

where $C_{\epsilon,\delta}$ is the contour



and $0 < \epsilon < 2\pi$.

For the time being, let us return to the case where $s \in \mathbb{R}, s > 1$, and suppose that N is a fixed positive integer. Using the same techniques as those of Section 2.3, we may write

$$\begin{aligned}\zeta(s) \quad &= \quad \frac{\Pi(-s)}{2\pi i} \int_{C_{\epsilon,\delta}} \frac{(-x)^s}{(e^x - 1)\,x}\,dx \\ &= \quad \frac{\Pi(-s)}{2\pi i} \int_{C_{0,0}} \frac{(-x)^s}{(e^x - 1)\,x}\,dx\end{aligned}$$

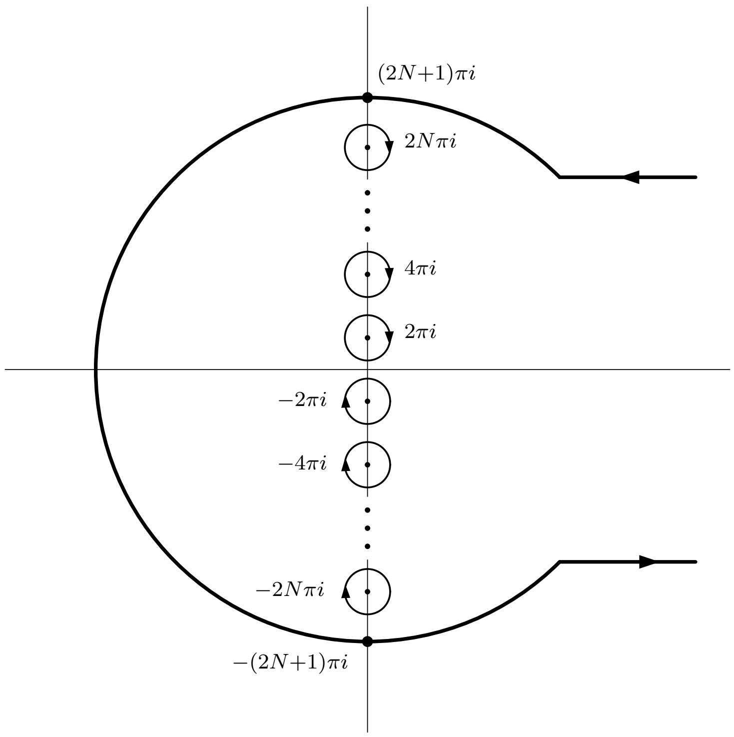
$$= \frac{\Pi(-s)}{2\pi i}\int_{C_{0,0}}\left(\sum_{n=1}^{N} e^{-nx}\right)\frac{(-x)^s}{x}\,dx + \frac{\Pi(-s)}{2\pi i}\int_{C_{0,0}}\frac{e^{-Nx}\,(-x)^s}{(e^x-1)\,x}\,dx$$

$$= \sum_{n=1}^{N}\frac{1}{n^s} + \frac{\Pi(-s)}{2\pi i}\int_{C_{0,0}}\frac{e^{-Nx}\,(-x)^s}{(e^x-1)\,x}\,dx$$

$$= \sum_{n=1}^{N}\frac{1}{n^s} + \frac{\Pi(-s)}{2\pi i}\int_{C_{\epsilon,\delta}}\frac{e^{-Nx}\,(-x)^s}{(e^x-1)\,x}\,dx\ .$$

Once again, the equalities above under transformed contours is justified by Cauchy's Theorem.

Next, expand the contour $C_{\epsilon,\delta}$ beyond the first $2N$ singularities of the integrand to $C_{(2N+1)\pi,\delta}$, and to maintain equality, add integrals around each singularity over paths with reverse orientation. That is, transform the contour of integration to the following:



where each of the small circular paths have radius $0 < \rho < \pi$. For convenience, denote $C_{(2N+1)\pi,\delta}$ by $\mathcal{C}_{N,\delta}$.

The residue theorem can then be used to compute the integrals about the singularities $\pm 2\pi i, \pm 4\pi i, \ldots, \pm 2N\pi i$. For the singularities above the real axis, letting $\oint$ denote reverse orientation of the path we may write

$$\begin{aligned}
\frac{\Pi(-s)}{2\pi i} \oint_{|x-2j\pi i|=\rho} \frac{e^{-Nx}\,(-x)^s}{(e^x-1)\,x}\,dx &= -\frac{\Pi(-s)}{2\pi i} 2\pi i \operatorname{Res}\left[\frac{e^{-Nx}\,(-x)^s}{(e^x-1)\,x}\right]_{(x=2j\pi i)} \\
&= \Pi(-s)\,(2\pi)^{s-1}\,ie^{-i\pi s/2} j^{s-1}\ .
\end{aligned}$$

Similarly, for the singularities below the real axis,

$$\begin{aligned}
\frac{\Pi(-s)}{2\pi i} \oint_{|x+2j\pi i|=\rho} \frac{e^{-Nx}\,(-x)^s}{(e^x-1)\,x}\,dx &= -\frac{\Pi(-s)}{2\pi i} 2\pi i \operatorname{Res}\left[\frac{e^{-Nx}\,(-x)^s}{(e^x-1)\,x}\right]_{(x=-2j\pi i)} \\
&= -\Pi(-s)\,(2\pi)^{s-1}\,ie^{-i\pi s/2} j^{s-1}\ .
\end{aligned}$$

Pairing the integrals over conjugate singularities and summing the result yields

$$\Pi(-s)\,i\,(2\pi)^{s-1}\left(e^{-i\pi s/2} - e^{i\pi s/2}\right)\sum_{j=1}^{N} j^{-(1-s)} \;=\; \Pi(-s)\,(2\pi)^{s-1}\,2\sin\!\left(\frac{\pi s}{2}\right)\sum_{j=1}^{N} j^{-(1-s)}\ ,$$

so that

$$\begin{aligned}
\zeta(s) &= \sum_{n=1}^{N} n^s \\
&+ \Pi(-s)\,(2\pi)^{s-1}\,2\sin\!\left(\frac{\pi s}{2}\right)\sum_{n=1}^{N} n^{-(1-s)} \\
&+ \frac{\Pi(-s)}{2\pi i}\int_{\mathcal{C}_{N,\delta}} \frac{e^{-Nx}\,(-x)^s}{(e^x-1)\,x}\,dx\ . \qquad (3.2)
\end{aligned}$$

As with our derivation of the global representation of the zeta function, we have arrived at this result under the restrictive assumption that $s \in \mathbb{R}, s > 1$. However, again in this case, the right hand side of (3.2) is defined and analytic at all complex numbers except possibly the positive integers. By defining the value of the right hand side of (3.2) at a positive integer to be the limit of its values nearby, it is possible to show that (3.2) is defined for all s except $s = 1$ where it has a simple pole. Thus (3.2) is another (global) representation of (2.11).

Now multiply both sides of (3.2) by $s\,(s-1)\,\Pi(s/2-1)\,\pi^{-s/2}/2$ and apply the identities (2.9) and (2.12) to obtain

$$\begin{aligned}
\xi(s) &= (s-1)\,\Pi\Big(\frac{s}{2}\Big)\,\pi^{-s/2}\sum_{n=1}^{N} n^{-s} \\
&+ (-s)\,\Pi\left(\frac{1-s}{2}\right)\pi^{-(1-s)/2}\sum_{n=1}^{N} n^{-(1-s)} \\
&+ \frac{(-s)\,\Pi\big(\frac{1-s}{2}\big)\,\pi^{-(1-s)/2}}{(2\pi)^{s-1}\,2\sin(\pi s/2)\,2\pi i}\int_{\mathcal{C}_{N,\delta}} \frac{e^{-Nx}\,(-x)^{s}}{(e^{x}-1)\,x}\,dx\ . \qquad (3.3)
\end{aligned}$$

From this point on, we will restrict s to the critical line. Further, as a result of (2.13), we need only consider those points $1/2+it$ on the critical line with positive imaginary part. Set $s = 1/2+it$ in (3.3), and let $f(t) = (-1/2+it)\,\Pi(1/2\,(-1/2+it))\,\pi^{-(1/2+it)/2}$. Then (3.3) may be written

$$\begin{aligned}
\xi\left(\frac{1}{2}+it\right) &= f(t)\sum_{n=1}^{N} n^{-(1/2+it)} \\
&+ f(-t)\sum_{n=1}^{N} n^{-(1/2-it)} \\
&+ \frac{-f(-t)}{(2\pi)^{1/2+it}\,2\sin\big(\frac{\pi}{2}\,\big(\frac{1}{2}+it\big)\big)\,i}\int_{\mathcal{C}_{N,\delta}} \frac{e^{-Nx}\,(-x)^{-1/2+it}}{e^{x}-1}\,dx\ . \qquad (3.4)
\end{aligned}$$

Defining $r(t) = f(t)\, e^{-i\vartheta(t)}$, we have

$$\xi\left(\frac{1}{2} + it\right) = r(t)\, Z(t) \ ,$$

and it is not difficult to show that $r(t)$ is an even function. Further, from the definition of $\vartheta(t)$ we can deduce that this last function is odd. Inserting this new information into (3.4) and canceling like factors then yields

$$\begin{aligned} Z(t) &= e^{i\vartheta(t)} \sum_{n=1}^{N} n^{-(1/2+it)} \\ &+ e^{-i\vartheta(t)} \sum_{n=1}^{N} n^{-(1/2-it)} \\ &+ \frac{e^{-i\vartheta(t)}}{(2\pi)^{1/2+it}\, 2\sin\left(\frac{\pi}{2}\left(\frac{1}{2}+it\right)\right) i} \int_{\mathcal{C}_{N,\delta}} \frac{e^{-Nx}\,(-x)^{-1/2+it}}{e^x - 1}\, dx \ . \end{aligned} \tag{3.5}$$

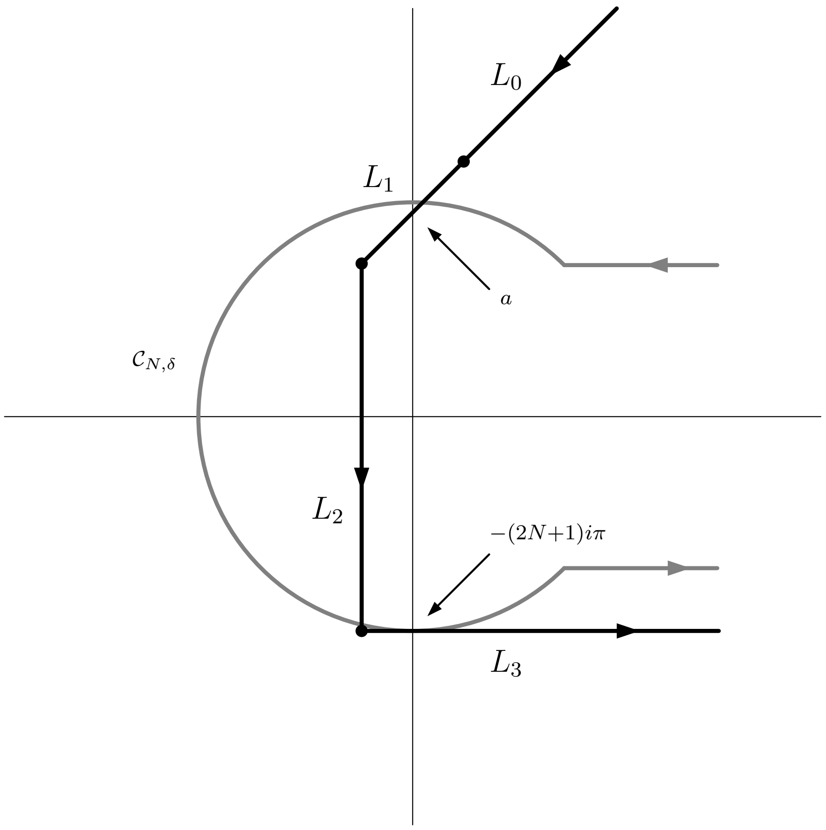
Finally, convert the sine function in (3.5) to its complex exponential form, and combine the two sum terms to get

$$\begin{aligned} Z(t) &= \sum_{n=1}^{N} n^{-1/2}\left[e^{i\vartheta(t)} n^{-it} + e^{-i\vartheta(t)} n^{it}\right] \\ &+ \frac{e^{-i\vartheta(t)} e^{-t\pi/2}}{(2\pi)^{1/2+it}\, e^{-i\pi/4}\,(1 - ie^{-t\pi})} \int_{\mathcal{C}_{N,\delta}} \frac{e^{-Nx}\,(-x)^{-1/2+it}}{e^x - 1}\, dx \\ &= 2\sum_{n=1}^{N} n^{-1/2} \cos\left[\vartheta(t) - t\log n\right] \\ &+ \frac{e^{-i\vartheta(t)} e^{-t\pi/2}}{(2\pi)^{1/2+it}\, e^{-i\pi/4}\,(1 - ie^{-t\pi})} \int_{\mathcal{C}_{N,\delta}} \frac{e^{-Nx}\,(-x)^{-1/2+it}}{e^x - 1}\, dx \\ &= S_N + R_N \ . \end{aligned} \tag{3.6}$$

This then is the starting point for our approximation. The S_N term above is readily computable, we merely have to choose a value for N. The R_N term, on the other hand, must be approximated, and this will be the focus of the following sections. In the process of approximating R_N, we will see that N is determined for us by the approximation method, and it turns out to be a value which greatly reduces the computation time for S_N.

3.4 Saddle Point Argument

At this point, let us suppose we are given a $t > 0$ for which to compute $Z(t)$. Then set $N = \left\lfloor (t/2\pi)^{1/2} \right\rfloor$, $a = i\,(2\pi t)^{1/2}$, and deform $\mathcal{C}_{N,\delta}$ into the path $L = L_0 \cup L_1 \cup L_2 \cup L_3$:



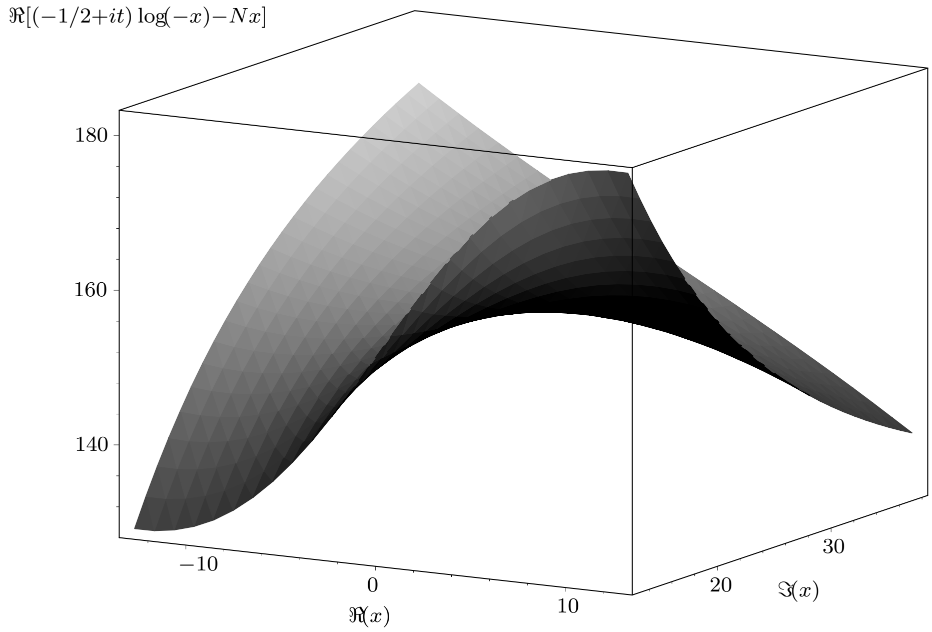
.

The length of L_1 above is $|a|$. Then

$$R_N = \frac{e^{-i\vartheta(t)} e^{-t\pi/2}}{(2\pi)^{1/2+it} e^{-i\pi/4} \left(1 - ie^{-t\pi}\right)} \int_L \frac{e^{-Nx} \left(-x\right)^{-1/2+it}}{e^x - 1} dx \ . \tag{3.7}$$

The motivation here is that the absolute value of the numerator of the integrand in (3.7) has a saddle point at $x = \left(-1/2 + it\right)/N \approx a$, and the path through $x = \left(-1/2 + it\right)/N$ parallel to L_1 is the line of steepest descent through the saddle point. In other words the integral along L_1 approximates the integral through the saddle point along the line of steepest descent, and hence "concentrates" the integral along L_1 near a.

For example, consider $t = 100$. Then $N = 4$ and $a = i\sqrt{200\pi} \approx 25i$. To see that a is near a saddle point of $\left|e^{-Nx} \left(-x\right)^{1/2+it}\right| = e^{\Re[(-1/2+it)\log(-x)-Nx]}$, it suffices to show that a is near a saddle point of the exponent $\Re\left[(-1/2 + it)\log(-x) - Nx\right]$. Plotting this last function over the region $\{x \mid -13 < \Re(x) < 13, 12 < \Im(x) < 38\}$ containing a shows that this is indeed the case:



Once again, equality of (3.6) and (3.7) under deformation of the contour is justified by Cauchy's Theorem and the dominance of the exponential terms of the integrand away from the origin. In particular, although L_0 and L_3 no longer run toward infinity along parallel conjugate paths near the real axis, their real parts still become arbitrarily large.

3.5 First Approximation

Note that so far, no approximations have been made. This is the first: ignore the integrals over L_0, L_2 and L_3 in (3.7) so that

$$R_N \approx R_{1,N} = \frac{e^{-i\vartheta(t)}e^{-t\pi/2}}{(2\pi)^{1/2}(2\pi)^{it}e^{-i\pi/4}(1-ie^{-t\pi})}\int_{L_1}\frac{(-x)^{-1/2+it}e^{-Nx}}{e^x-1}dx\ . \tag{3.8}$$

Edwards[1] demonstrates that the error in this approximation is at most $e^{-t/11}$ for $t > 100$. So the integral in (3.7) is in fact very highly concentrated near $a = i(2\pi t)^{1/2}$ over the short path L_1 passing close to the saddle point $x = (-1/2+it)/N$.

3.6 Change of Variables

The next step is to obtain a rapidly convergent expansion in terms of $(x-a)$ of the numerator $(-x)^{-1/2+it}e^{-Nx}$ from (3.8). To this end, set $p = (t/2\pi)^{1/2} - N$, the fractional part of $(t/2\pi)^{1/2}$, and write the numerator of the integrand in (3.8) as

$$(-x)^{-1/2+it}e^{-Nx} = (-a)^{-1/2+it}g(x-a)\exp\left[-Na + p(x-a) + i(x-a)^2/(4\pi)\right]$$

where

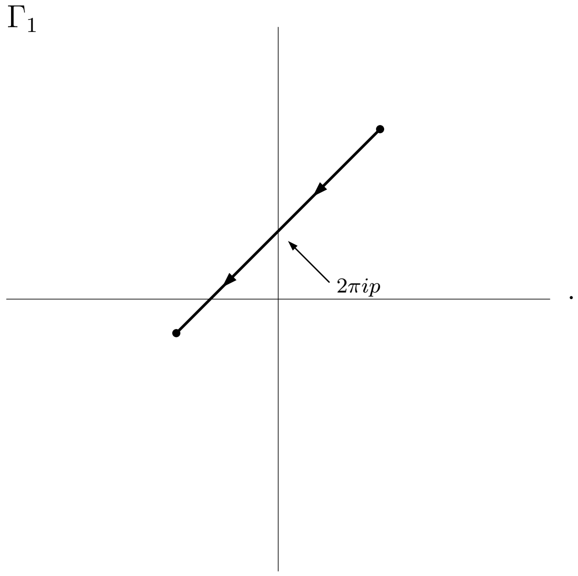
$$g(x-a) = \exp\left[\frac{-i(x-a)^2}{4\pi} - p(x-a) - N(x-a) + \left(-\frac{1}{2}+it\right)\log\left(1+\frac{x-a}{a}\right)\right].$$

The motivation here is to factor out the terms of the numerator which make the largest contribution, leaving a function $g(x-a)$ whose power series converges very rapidly on L_1.

Without computing its coefficients, consider $g(x-a)$ as a power series $\sum_{n=0}^{\infty} b_n (x-a)^n$. Now make the change of variables $x = u + 2\pi i N$ and simplify. The result is then

$$\begin{aligned} R_{1,N} &= \left(\frac{t}{2\pi}\right)^{-1/4} \left[\frac{e^{i\left(-\frac{1}{48t}-\frac{7}{5760t^3}+\cdots\right)}}{1-ie^{-t\pi}}\right] (-1)^{N-1} \\ &\times e^{i\pi/8} e^{-2\pi i p^2} \frac{1}{2\pi i} \int_{\Gamma_1} \frac{e^{iu^2/(4\pi)} e^{2pu} \sum_{n=0}^{\infty} b_n (u-2\pi i p)^n}{e^u - 1} du \end{aligned} \tag{3.9}$$

where Γ_1 is the contour



Here again, the length of Γ_1 is $|a|$.

3.7 Second Approximation

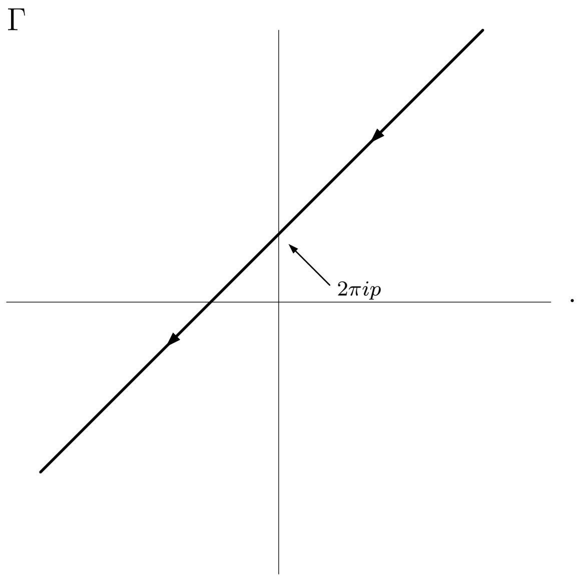
Since the series in (3.9) converges very rapidly, let us consider truncation of the series at, say, the K^{th} term. Then (3.9) becomes

$$\begin{aligned} R_{1,N} \approx R_{2,N} &= \left(\frac{t}{2\pi}\right)^{-1/4} \left[\frac{e^{i\left(-\frac{1}{48t}-\frac{7}{5760t^3}+\cdots\right)}}{1-ie^{-t\pi}}\right] (-1)^{N-1} \\ &\times e^{i\pi/8} e^{-2\pi i p^2} \frac{1}{2\pi i} \int_{\Gamma_1} \frac{e^{iu^2/(4\pi)} e^{2pu} \sum_{n=0}^{K} b_n (u-2\pi i p)^n}{e^u - 1} du \end{aligned} \tag{3.10}$$

We will see later that this single step introduces the dominant error of the approximation method.

3.8 Third Approximation

Next, we would like to evaluate the integral in (3.10). Riemann was not able to compute this integral exactly, but he was able to show that term by term integration was possible if the path Γ_1 in (3.10) was extended to the path Γ as follows:



Then the remainder term becomes

$$R_{2,N} \approx R_{3,N} = (-1)^{(N-1)} \left(\frac{t}{2\pi}\right)^{-1/4} \left[\frac{e^{i\left(-\frac{1}{48t}-\frac{7}{5760t^3}+\cdots\right)}}{1-ie^{-t\pi}}\right] \sum_{n=0}^{K} b_n c_n \tag{3.11}$$

where

$$c_n = e^{i\pi/8} e^{-2\pi i p^2} \frac{1}{2\pi i} \int_\Gamma \frac{e^{iu^2/(4\pi)} e^{2pu} \left(u - 2\pi i p\right)^n}{e^u - 1} \, du \; .$$

Analysis of $g(x-a)$ from Section 3.6 shows $b_0 = 1$. Riemann was able to show that

$$c_0 = \Psi(p) = \frac{\cos\left[2\pi\left(p^2 - p - 1/16\right)\right]}{\cos(2\pi p)} \; .$$

The task now is to approximate $\sum_{n=0}^{K} b_n c_n$.

3.9 Fourth Approximation

Riemann devised a method of approximating $\sum_{n=0}^{K} b_n c_n$ in (3.11) which does not require the computation of the individual b_n's and c_n's. With this method, one obtains an exact expression for $\sum_{n=0}^{\lfloor K/3 \rfloor} b_n c_n$.

Let $\omega = (2\pi/t)^{1/2}$ and show the following:

- Each coefficient b_n is a polynomial in ω of degree n. The coefficients b_0, b_1 and b_2 contain only terms in ω^0, ω^1 and ω^2 respectively. All other b_n contain no terms in ω of degree less than $\lfloor n/3 \rfloor$.

- Each integral c_n is a linear combination of $\Psi^{(k)}(p)$, $k = 0, \ldots, n$, and the coefficient of each $\Psi^{(k)}(p)$ does not involve t. The c_n satisfy the formal relationship
$$e^{2\pi i y^2} \sum_{m=0}^{\infty} \frac{\Psi^{(m)}(p)}{m!} y^m = \sum_{n=0}^{\infty} \frac{(2y)^n}{n!} c_n \; . \tag{3.12}$$

- The truncated series $\sum_{n=0}^{K} b_n c_n$ is therefore a polynomial in ω whose coefficients are linear combinations of $\Psi^{(k)}(p)$ not involving t .

Start by multiplying both side of the formal relationship (3.12) by $\sum_{j=0}^{K} j! b_j \,(2y)^{-j}$ which yields
$$\left(e^{2\pi i y^2} \sum_{j=0}^{K} j! b_j \,(2y)^{-j} \right) \left(\sum_{m=0}^{\infty} \frac{\Psi^{(m)}(p)}{m!} y^m \right) = \left(\sum_{n=0}^{\infty} \frac{(2y)^n}{n!} c_n \right) \left(\sum_{j=0}^{K} j! b_j \,(2y)^{-j} \right) \tag{3.13}$$

The first factor of the left hand side of (3.13) can be considered a polynomial in ω with coefficients which are functions of y. That is,
$$e^{2\pi i y^2} \sum_{j=0}^{K} j! b_j \,(2y)^{-j} \quad = \quad \sum_{j=0}^{K} A_j \,(y)\, \omega^j \quad = \quad G(y) \; .$$

The right hand side of (3.13) is a series in y with constant term $\sum_{n=0}^{K} b_n c_n = \sum_{j=0}^{K} C_j \omega^j$ say.

The b_n can be shown to satisfy a recurrence relation

$$b_{n+1} = \frac{\pi i\,(2n+1)\,b_n - b_{n-2}}{4\pi^2 \omega^{-1}\,(n+1)} \ , \tag{3.14}$$

where $b_n = 0$ for $n < 0$, from which a recurrence relation for the A_j can be derived:

$$A_j = -\frac{1}{2} y A_{j-1} - \frac{1}{32}\pi^{-2} D^2 \left(y^{-1} A_{j-1}\right) \ . \tag{3.15}$$

Note that $A_0 = e^{2\pi i y^2}$.

As a result of our first observation above, $A_0, \ldots, A_{\lfloor K/3 \rfloor}$ are independent of K, so use (3.15) to compute these. Once A_j, $j = 0, \ldots, \lfloor K/3 \rfloor$ are known, compute the constant term of

$$A_j\,(y) \sum_{m=0}^{\infty} \frac{\Psi^{(m)}(p)}{m!} y^m \ .$$

These are our C_j. Finally then,

$$\sum_{n=0}^{K} b_n c_n \approx \sum_{j=0}^{\lfloor K/3 \rfloor} C_j \omega^j$$

This method is labour saving, however it should be noted that the b_n's and c_n's can be computed directly using their recurrence relations. That is, the b_n's (as a function of ω) can be computed from (3.14) and from (3.12) we have

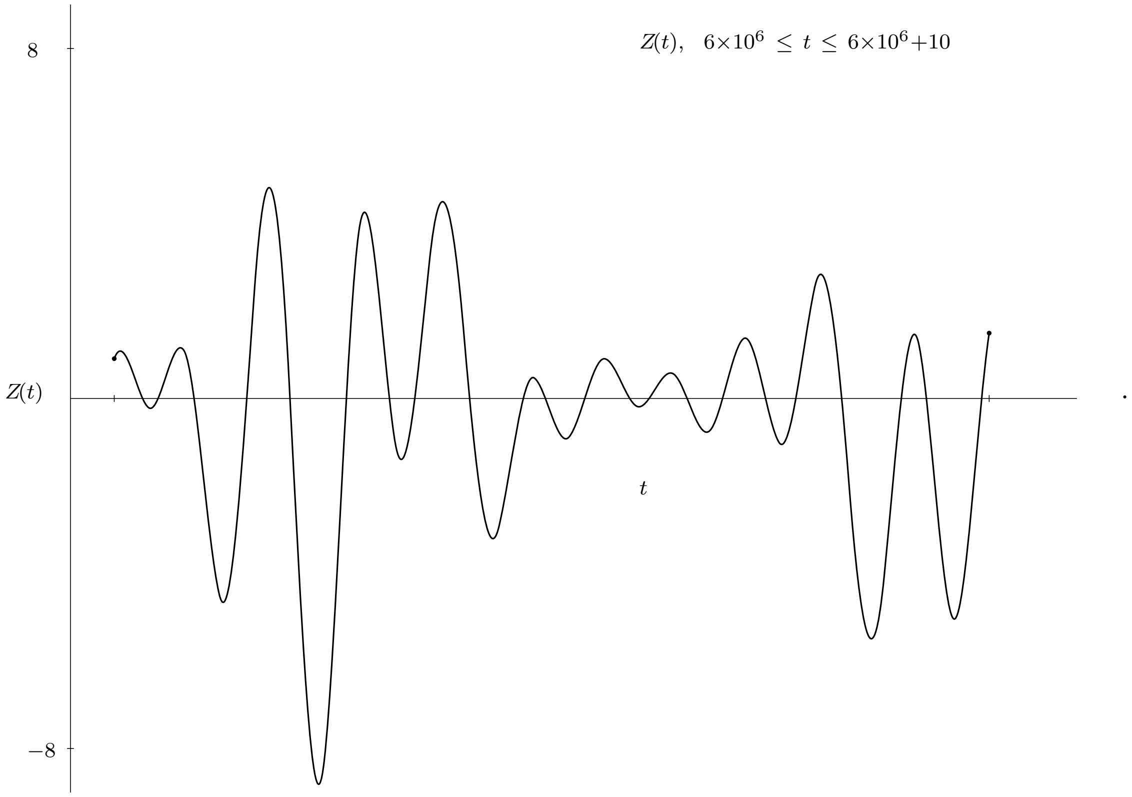
$$c_n = \frac{n!}{2^n} \sum_{j=0}^{\lfloor n/2 \rfloor} \frac{(2\pi i)^j}{2j!} \frac{\Psi^{(n-2j)}(p)}{(n-2j)!} \ .$$

Using a symbolic computation package such as Maple it would not be difficult to compute $\sum_{n=0}^{K} b_n c_n$ as the desired series in increasing powers of ω.

3.10 An Example

Appendix B contains an implementation of the Riemann-Siegel method written in the C programming language. This program allows the user to input the endpoints of the (positive) t domain as well as the number of sample points desired. The output is a list of $(t, Z(t))$ pairs suitable for plotting using a plotting package.

The following is a plot of $(t, Z(t))$ for $6 \times 10^6 \leq t \leq 6 \times 10^6 + 10$ using 10 000 data samples:



3.11 Errors in the Approximation

There are three main sources of error in calculating $Z(t)$ using the Riemann-Siegel formula. The first two of these are due to the approximations made in the derivation of the formula, while the third is inherent in the tools used to compute the value of the approximation at a given point: the computer hardware and software.

The introduction of the first error was in Section 3.5 where the contour of the integral was restricted to a short path through a point on the imaginary axis which was near a saddle point of the integral. As noted in that section, Edwards[1] shows that the error in that approximation is $O\left(e^{-t/11}\right)$.

The second, more significant error is that of Section 3.7 where the series expansion of $g(x-a)$ is truncated. Bounds on the size of this error have been determined by Titchmarsh[6] and more recently by W. Gabcke as noted in Odlyzko's work[5]. Citing a much more complicated result of his own, Titchmarsh notes that if the C_0 term is the only term retained in the Riemann-Siegel formula for $Z(t)$, then the error in the approximation is less than $3/2t^{-3/4}$ for $t > 125$.

According to Odlyzko, Gabcke's result is "essentially optimal" and states that if the C_0 and C_1 terms are the only terms retained in the Riemann-Siegel approximation, then the error is at most $0.053t^{-5/4}$ for $t \geq 200$.

Although Riemann's ingenious method for computing the $\Psi^{(n)}(p)$ functions greatly facilitates the computation of many of the C_n terms, in practice only two or three of these last terms are required to determine the existence of a zero of the ζ function. For large values of t, provided $|Z(t)|$ is sufficiently large between zeros, Gabcke's result ensures sufficient accuracy to identify the existence of a zero.

Finally, the third source of error in the computations is that inherent in the floating point arithmetic used by the computer. To model the real line, compilers use a finite set of discrete points, and the accuracy of the model depends on the data types used for the calculation in question. For the present case, floating point data types were declared `long double` to maintain as much accuracy as possible for the computations of $Z(t)$ at large values of t. The GNU C Compiler used to implement the programs contained in the appendices uses twenty digits for both computation and storage of `long double` data types on Intel-80x86 based systems.

The greatest amount of accuracy is lost in the floating point evaluation of

$$\sum n^{-1/2} \cos(\vartheta(t) - t \log n) \ .$$

Firstly, the argument of the cosine function is $O(t \log t)$, which, when reduced modulo 2π and evaluated by the cosine function, leaves about $20 - \lceil \log_{10} t \rceil$ decimal digits to the right of the decimal point. Secondly, the error of each of these cosine evaluations is accumulated when the $\lfloor \sqrt{t/\,(2\pi)} \rfloor$ terms of the series are summed.

One of the main uses of the Riemann-Siegel formula is not to evaluate $Z(t)$ at a given point t with great precision, but rather to determine sign changes of $Z(t)$ with sufficient certainty to count roots. As such, provided the cumulative errors noted above are not so great as to prevent the determination of the sign of $Z(t)$, the Riemann-Siegel formula is a valuable computational tool despite its practical shortcomings.

This section dealing with errors would not be complete without mentioning that the floating point computation problems noted are inherent in all computations of $\zeta(1/2 + it)$ using fixed precision arithmetic. This limitation can be overcome by using a variable precision computation package such at the GNU `bc` utility, however, so much speed is sacrificed in return for the increased accuracy that computations at large values of t become infeasible.

Chapter 4
Verification of the Riemann Hypothesis

4.1 Introduction

Since the rediscovery of the Riemann-Siegel formula in the early 1930's, this method has been used to verify the correctness of the Riemann Hypothesis up to very large values of t. In this chapter, we examine the key steps involved in the verification process.

Prior to Siegel's work, the main method for computing values of the ζ function was by Euler-Maclaurin summation. Though precise, the amount of work required to compute $\zeta(1/2 + it)$ is of the same order as t. With Riemann-Siegel, the decrease in computation time from $O(t)$ to $O\left(\sqrt{t}\right)$ allowed huge improvements over the previously established limit that the Riemann Hypothesis was correct up to $t \approx 300$. This last result was obtained by J. Hutchinson using the more labour intensive Euler-Maclaurin method. More recent results have shown the Riemann Hypothesis to hold for $t \leq 5 \times 10^8$, while Odlyzko[5] has developed algorithms which verify the correctness of the Riemann Hypothesis in a large region about $t = 10^{20}$.

The key to the verification process is a method of A. Turing which allows one to analytically determine the number of zeros $\sigma + i\tau$ in the region $0 < \sigma < 1$ (the *critical* *strip*) for $0 \leq \tau \leq t$. Once all zeros on the critical line up to height t have been counted using Riemann-Siegel, Turing's method is then used to verify that the zeros found are indeed the only zeros $\sigma + i\tau$ in the critical strip with $\tau \leq t$.

4.2 Gram's Observations

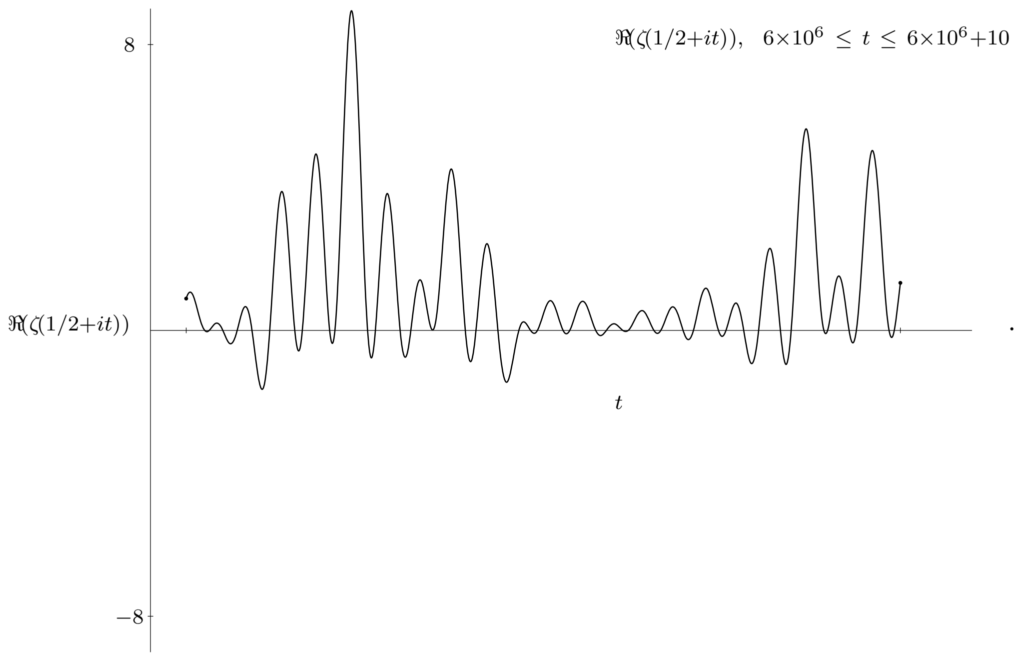
Prior to the rediscovery of the Riemann-Siegel formula, J. Gram computed the first fifteen roots of $\zeta(1/2+it)$ using the Euler-Maclaurin formula. In the course of his work, Gram made the following observations:

1. $\Re(\zeta(1/2+it))$ has a tendency to be positive, while the values of $\Im(\zeta(1/2+it))$ are distributed more evenly between positive and negative values.

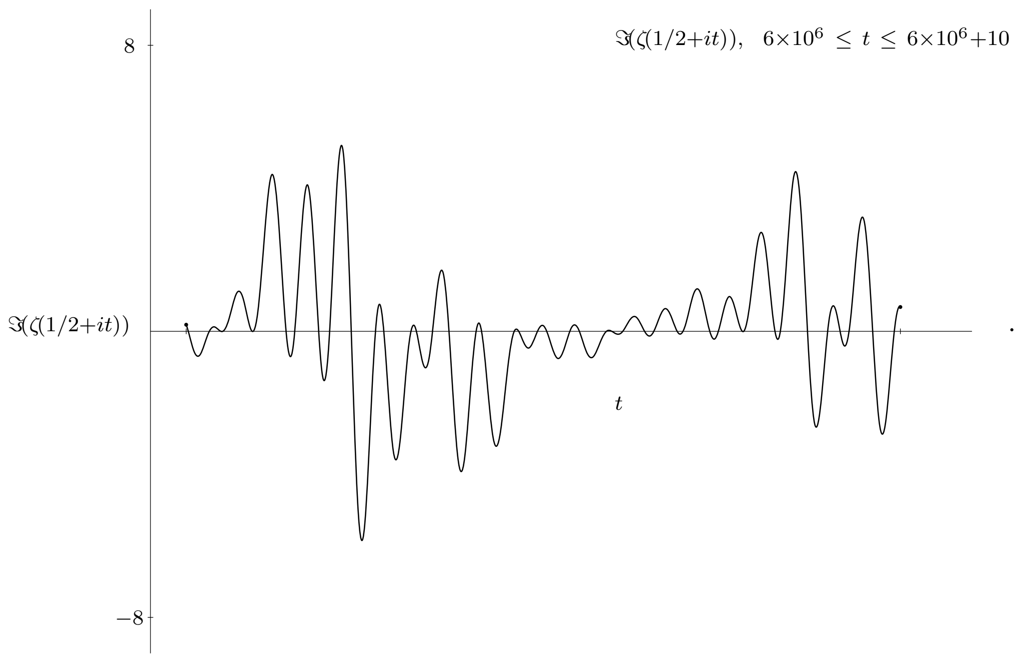
2. The zeros of $Z(t)$ tend to alternate with the zeros of $\sin\vartheta(t)$.

Although these observations are based purely on empirical results at low values of t, in practice they hold true for all ranges of t tested to date. These simple observations are the foundation upon which modern root counting strategies are based.

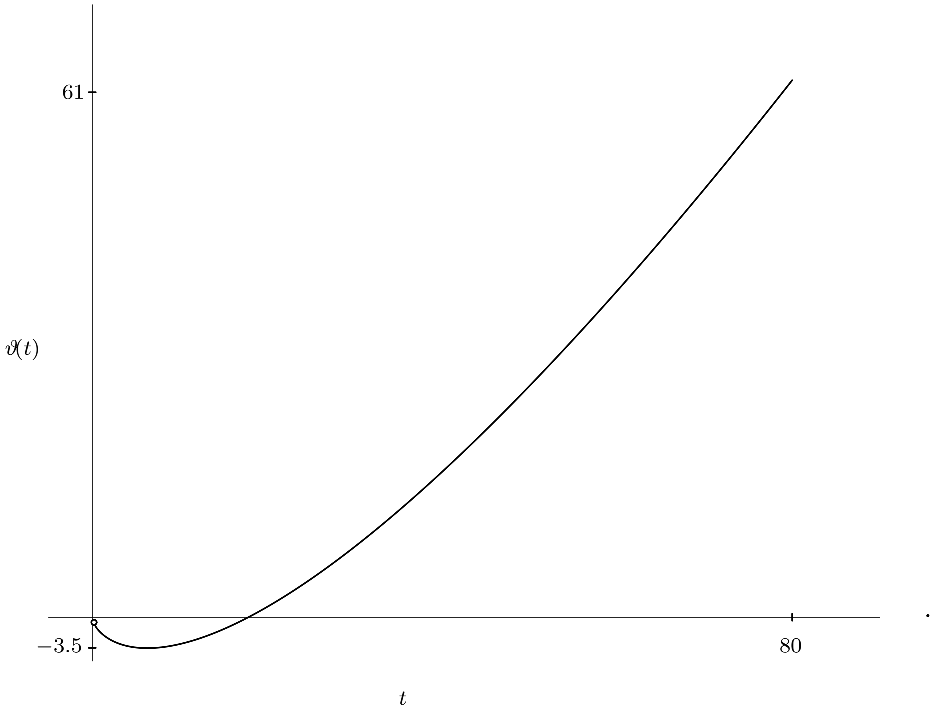
To illustrate the first observation above, consider the graph of the real part of $\zeta(1/2+it)$ for $6 \times 10^6 \leq t \leq 6 \times 10^6 + 10$:



Compare this to a plot of $\Im(\zeta(1/2+it))$ over the same domain:



To discuss the second observation, first examine the function $\vartheta(t)$. From (3.1) we have that $\vartheta'(t) \approx 1/2 \log(t/(2\pi))$, and for $t > 7$, $\vartheta(t)$ is increasing. Further, this rate of increase is nearly constant over short intervals. The graph of $\vartheta(t)$ looks like:



For each $n \geq -1$, define the point $t > 7$ which satisfies $\vartheta(t) = n\pi$ to be the n^{th} *Gram point*, and denote it g_n. For example, the first five Gram points are approximately as follows:

n	g_n
-1	9.666908077
0	17.84559954
1	23.17028270
2	27.67018222
3	31.71797995

Returning to the second observation above, assume for the moment that the tendency for $\Re(\zeta(1/2+it))$ to be positive continues indefinitely, and recall that $\zeta(1/2+it) = Z(t)\, e^{-i\vartheta(t)} = Z(t)\cos\vartheta(t) - iZ(t)\sin\vartheta(t)$. Under the assumption that the real part of $\zeta(1/2+it)$ has a tendency to be positive, the signs of $Z(t)$ and $\cos\vartheta(t)$ must have a tendency to be the same. So as $\cos\vartheta(t)$ changes sign as $\vartheta(t)$ increases almost linearly between integer multiples of π, one would expect a corresponding sign change in $Z(t)$ indicating at least one root of $Z(t)$. Further, if the sign of $Z(t)$ follows that of $\cos\vartheta(t)$, this sign change should occur approximately midway between Gram points, supporting Gram's second observation. Finally, because of the positive tendency of $\Re(\zeta(1/2+it))$, one would expect that this root is the only root in the interval, and that it has order one.

Gram's empirical observation that zeros of $Z(t)$ tend to alternate with the zeros of $\sin\vartheta(t)$ is known as *Gram's Law*, and is stated as the inequality

$$(-1)^n\, Z(g_n) > 0\ .$$

This "law" does fail eventually, and infinitely often in fact, but if true most of the time, reduces the verification of the Riemann Hypothesis to the examination of the more infrequent cases where Gram's Law fails to hold. (In fact, using the methods of the next section, one can show that if Gram's Law held for all Gram points g_n, then the Riemann Hypothesis would follow).

The question remains as to whether the assumption about the positive tendency of $\Re(\zeta(1/2+it))$ is valid. This is equivalent to the question of whether Gram's Law continues to hold for most g_n. Empirically, Gram's Law holds for about 70% of tested cases, and depends on the tested range. One argument in support of Gram's Law is that at Gram points g_n, the first term of the main sum in the Riemann-Siegel formula is always $2\,(-1)^n$, and it is unlikely that the subsequent terms which rapidly decrease in absolute value will reinforce one another to overcome the strong sign tendency of the first term. These arguments are plausible, but are by no means rigorous. At present, Gram's Law is still a purely empirical result which serves well for the purpose root counting.

It must be noted that Gram's Law provides only a lower bound on the number of roots on the critical line in a given range. That is, if Gram's Law is satisfied at two successive Gram points, then there is certainly at least one root of $Z(t)$ between the points, but there is no guarantee that there is only one root. In fact, there are cases at very large values of t where more than one root occurs between Gram points, though these cases are rare—see[5]. Once we have applied the methods of the next section to determine and upper bound on the number of roots in our range, we may then compare with the roots found following Gram's Law to see if indeed we have located all roots in the range.

Following the terminology used by Edwards, call a Gram point g_n *good* if it satisfies Gram's Law, and *bad* otherwise. Define a *Gram block* to be an interval $[g_n, g_{n+k})$ such that g_n and g_{n+k} are good Gram points but all Gram points strictly inside the interval are bad. The process of counting roots on the critical line is then to simply count the Gram points where Gram's Law is satisfied, and to count the number of zeros of $Z(t)$ in each Gram block.

4.3 The Number of Roots to Height T

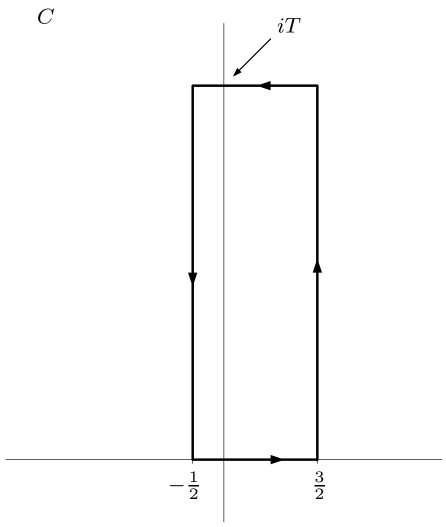
Once we have counted all roots on the critical line up to some height T, we must be able to determine whether these are all the roots in the critical strip up to this height.

Let $N(T)$ denote the number of roots of $\zeta(\sigma + i\tau)$ in the region $0 < \sigma < 1$ and $0 \leq \tau \leq T$. We would like to have a method to easily evaluate $N(T)$ in order to verify the result of our root counting algorithm based on Gram's Law. The method currently used was developed by A. Turing and is based on results of J. Littlewood and R. Backlund.

Riemann noted that

$$N(T) = \frac{1}{2\pi i}\int_C \frac{\xi'(s)}{\xi(s)}\,ds$$

where C is the contour

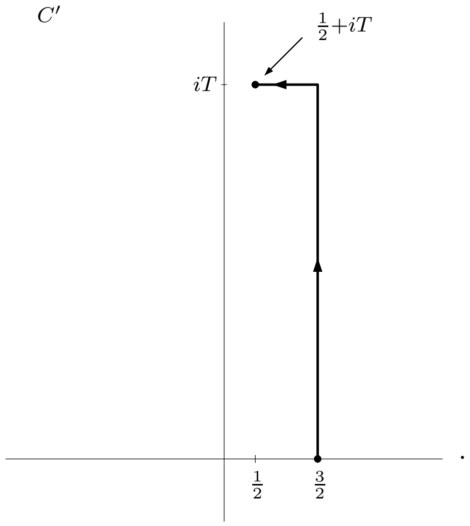


as long as it is assumed that there are no zeros of ξ on the contour itself. This result can be shown using the residue theorem and the fact that ξ is an entire function. (In fact, it is known that there are no zeros of ξ on the two vertical sides of the contour and on the portion of the contour on the real axis, so the necessary assumption is that there are no zeros on the fourth remaining side.)

Using properties of $\xi(s)$ and the functional equation, this result may be written

$$N(T) = \frac{1}{\pi}\vartheta(t) + 1 + \frac{1}{\pi}\Im\left(\int_{C'} \frac{\zeta'(s)}{\zeta(s)}\,ds\right) \tag{4.1}$$

where the contour C' is



Provided $\Re(\zeta)$ is non-zero on the contour C', and since $\zeta(3/2)$ is real and positive, the integrand in (4.1) has an anti-derivative which may be bounded to yield

$$\left|\frac{1}{\pi}\Im\left(\int_{C'}\frac{\zeta'(s)}{\zeta(s)}\,ds\right)\right| < \frac{1}{2}\ .$$

Backlund used this argument to show that under the condition that $\Re(\zeta)$ is non-zero on the contour C', $N(T)$ is the integer nearest $\pi^{-1}\vartheta(T)+1$.

This method is a useful tool, but for each determination of $N(T)$, one must prove that no zeros of ζ lie on the contour C'. Consider instead the function

$$S(T) = N(T) - \frac{\vartheta(T)}{\pi} - 1\ .$$

Littlewood proved that $\int_0^T S(t)\ dt = O(T)$. From Littlewood's proof of this result, Turing was able to show that

$$\left|\int_{t_1}^{t_2} S(t)\ dt\right| \leq 2.30 + 0.128\log\left(\frac{t_2}{2\pi}\right)$$

for $168\pi < t_1 < t_2$. For certain Gram points g_n, this expression will be used to bound $S(g_n)$ and prove that $N(g_n) = n+1$. Turing reasoned as follows: Suppose that Gram's Law is satisfied at the Gram point g_m. That is, at $t = g_m$ we have

$$(-1)^m Z(g_m) > 0 \ . \tag{4.2}$$

Then at g_m we must have

$$\begin{aligned} S(g_m) &= N(g_m) - \frac{\vartheta(g_m)}{\pi} - 1 \\ &= N(g_m) - \frac{m\pi}{\pi} - 1 \\ &= N(g_m) - m - 1 \end{aligned}$$

and if we can conclude that $S(g_m) = 0$ then we will know exactly how many roots lie in the critical strip up to height g_m. Let E denote the number of even order roots (counting multiplicities) on the critical line up to height g_m. Similarly, let D denote the number of odd order roots. Finally, let F denote the number of roots in the critical strip but not on the critical line. Then $N(g_m) = D + E + F$, and noting that $Z(g_{-1}) < 0$, the sign of $Z(g_m)$ is then $(-1)\,(-1)^E\,(-1)^D = (-1)^{D+E+1} = (-1)^{D+E-1}$. But this must equal $(-1)^m$ by (4.2), so $D+E-1$ and m are either both even integers or both odd. Further, F must be even since roots not on the critical line occur in pairs by virtue of the functional equation $\xi(s) = \xi(1-s)$. Thus

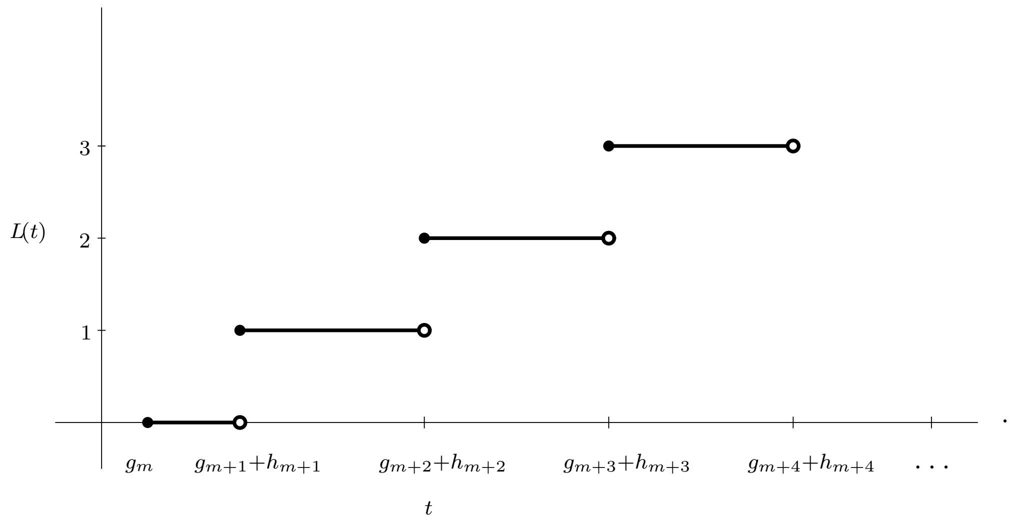
$$\begin{aligned} S(g_m) \pmod 2 &= N(g_m) - m - 1 && \pmod 2 \\ &= D + E + F - m - 1 && \pmod 2 \\ &= (D + E - 1 - m) + F && \pmod 2 \\ &= 0 && \pmod 2 \ , \end{aligned}$$

which shows that $S(g_m)$ must be an even integer. It remains now to show only that $|S(g_m)| < 2$, and this Turing did as follows.

Suppose g_m and g_{m+k} are good Gram points (that is, Gram's Law is satisfied at these points). For the Gram points g_j, $j = m+1, ..., m+k-1$ in between, compile a separate list of points h_j of corrections to the g_j which force

$$(-1)^j \, Z(g_j + h_j) > 0, \qquad j = m+1, ..., k-1 \; .$$

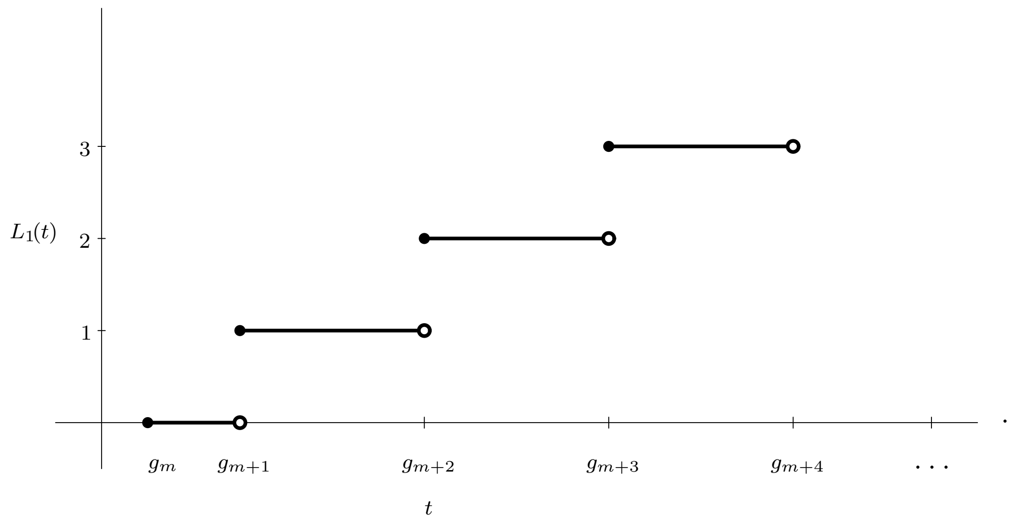
If g_j is a Gram point which satisfies Gram's Law, then $h_j = 0$. For g_j where Gram's Law is not satisfied, the h_j should be chosen to be as small as possible and such that the sequence $g_j + h_j$ is strictly increasing. Now define the function $L(t)$ as



So each jump of $L(t)$ at a point $g_j + h_j$ indicates at least one root of $Z(t)$ in the interval $g_{j-1} + h_{j-1} < t < g_j + h_j$, so

$$N(t) \geq N(g_m) + L(t) \, . \tag{4.3}$$

Next, define $L_1(t)$ as



Here $L_1(t)$ is the Gram point counting function for Gram points larger than g_m. Now $\pi^{-1}\vartheta(t)$ is an increasing function of t which coincides with $\pi^{-1}\vartheta(g_m) + L_1(t)$ at Gram points g_{m+j}, $j > 0$. Further, the following relation between these two functions holds for $t \geq g_m$:

$$\pi^{-1}\vartheta(g_m) + L_1(t) \quad \leq \quad \pi^{-1}\vartheta(t) \quad \leq \quad \pi^{-1}\vartheta(g_m) + L_1(t) + 1 \ . \tag{4.4}$$

Combining the results of (4.3) and (4.4) then gives:

$$\begin{aligned} S(t) &= N(t) - \pi^{-1}\vartheta(t) - 1 \\ &= N(t) - (\pi^{-1}\vartheta(t) + 1) \\ &\geq N(g_m) + L(t) - (\pi^{-1}\vartheta(t) + 1) && \text{from (4.3)} \\ &\geq N(g_m) + L(t) - (\pi^{-1}\vartheta(g_m) + L_1(t) + 1 + 1) && \text{from (4.4)} \\ &= S(g_m) + L(t) - L_1(t) - 1 \ , \end{aligned}$$

whence

$$S(g_m) \leq S(t) + 1 - (L(t) - L_1(t)) \ . \tag{4.5}$$

In terms of indicator functions, we may consider

$$L(t) - L_1(t) = \sum_{j=1}^{k-1} \left(I_{[g_{m+j}+h_{m+j}, g_{m+j})} - I_{[g_{m+j}, g_{m+j}+h_{m+j})} \right) ,$$

so that integrating both sides of (4.5) from g_m to g_{m+k} yields

$$\begin{aligned}
\int_{g_m}^{g_{m+k}} S(g_m)\; dt &\leq \int_{g_m}^{g_{m+k}} S(t)\; dt + \int_{g_m}^{g_{m+k}} 1\, dt + \int_{g_m}^{g_{m+k}} \left(L(t) - L_1(t)\right)\, dt \\
\Rightarrow S(g_m)\,(g_{m+k} - g_m) &\leq 2.30 + 0.128 \log\Big(\frac{g_{m+k}}{2\pi}\Big) + (g_{m+k} - g_m) + \sum_{j=1}^{k-1} h_{m+j} \\
\Rightarrow S(g_m) &\leq 1 + (g_{m+k} - g_m)^{-1} \left(2.30 + 0.128 \log\Big(\frac{g_{m+k}}{2\pi}\Big) + \sum_{j=1}^{k-1} h_{m+j} \right) .
\end{aligned}$$

Note the use of Turing's bound in the second line above. As k increases, the second term of this last expression becomes less than one as long as the sum of the correction terms h_{m+j} does not become too large.

If we are able to find a k and correction terms $h_{m+1}, \ldots, h_{m+k}$ so that $S(g_m) < 2$, it will follow that $S(g_m) \leq 0$, whence $N(g_m) \leq \pi^{-1}\vartheta(g_m) + 1 = m + 1$. For our purposes, this will be enough, for as long as our root counting algorithm produces at least $m+1$ roots, we will have succeeded in verifying the Riemann Hypothesis up to $t = g_m$.

It should be noted that a construction similar to that of (4.3) and (4.4) may be used to compute a lower bound for $S(g_m)$. As such, the number of zeros up to height $t = g_m$ can be determined using only a set of appropriately chosen points $g_{m+j} + h_{m+j}$. This is the true beauty of Turing's method — a small set of easily computable quantities is sufficient for one to determine with absolute certainty the number of zeros of the ζ function in the critical strip, and further, not a single evaluation of the ζ function itself is necessary.

We now have all of the machinery required to implement our Riemann Hypothesis verification program. We have an efficient computation method for identifying roots of the ζ function, and we have have a means of ensuring that we have located all roots in a given region. It remains now to implement our program and that will be the subject of the next chapter.

Chapter 5
Riemann-Siegel At Work

5.1 Introduction

In this chapter we discuss the results of our implementation of a Riemann Hypothesis verification program. This verification program combines into practical application the theory and methods of the previous chapters. We first present our raw findings and interpret the results, and we then discuss the computer programs used in the computation, including error considerations.

5.2 Results

The following are the results obtained by running the computer programs contained in Appendix C and Appendix D, and following are the conclusions which follow from these results:

1. $g_{12\,193\,873} = 6\,000\,000.485\,999$
2. There are at least 12 193 874 zeros of $\zeta(1/2 + it)$ for $0 \le t \le g_{12\,193\,873}$.
3. There are 1 518 045 Gram blocks containing 3 317 645 of these roots.
4. The longest Gram block in the range $0 \le t \le g_{12\,193\,873}$ is $[g_{1\,181\,229},\ g_{1\,181\,235}] = [698\,899.370\,788,\ 698\,902.615\,289]$.

5. The data for twenty Gram points used to apply Turing's method (see Section 4.3) is as follows:

m	g_m	h_m	$g_m + h_m$	$Z(g_m + h_m)$
12 193 873	6 000 000.486	0.000	6 000 000.486	−0.110
12 193 874	6 000 000.942	−0.100	6 000 000.842	0.856
12 193 875	6 000 001.399	0.000	6 000 001.399	−3.262
12 193 876	6 000 001.855	0.000	6 000 001.855	4.634
12 193 877	6 000 002.311	0.000	6 000 002.311	−8.947
12 193 878	6 000 002.768	0.000	6 000 002.768	3.350
12 193 879	6 000 003.224	0.000	6 000 003.224	−1.161
12 193 880	6 000 003.680	0.000	6 000 003.680	4.335
12 193 881	6 000 004.137	0.000	6 000 004.137	−1.663
12 193 882	6 000 004.593	0.100	6 000 004.693	0.245
12 193 883	6 000 005.049	0.000	6 000 005.049	−0.721
12 193 884	6 000 005.505	0.000	6 000 005.505	0.732
12 193 885	6 000 005.962	0.000	6 000 005.962	−0.172
12 193 886	6 000 006.418	0.000	6 000 006.418	0.513
12 193 887	6 000 006.874	0.000	6 000 006.874	−0.510
12 193 888	6 000 007.331	0.000	6 000 007.331	0.997
12 193 889	6 000 007.787	0.000	6 000 007.787	−0.225
12 193 890	6 000 008.243	0.000	6 000 008.243	1.625
12 193 891	6 000 008.700	0.000	6 000 008.700	−5.559
12 193 892	6 000 009.156	0.000	6 000 009.156	1.503

Now recall from Section 4.3 that if we define the function

$$S(T) = N(T) - \frac{\vartheta(T)}{\pi} - 1 \ ,$$

then S takes on even integer values at Gram points g_m. Further, we saw that if g_m and g_{m+k} are good Gram points, then for suitably chosen adjustments $h_{m+1}, \ldots, h_{m+k-1}$,

$$S(g_m) \leq 1 + (g_{m+k} - g_m)^{-1} \left(2.30 + 0.128 \log\left(\frac{g_{m+k}}{2\pi}\right) + \sum_{j=1}^{k-1} h_{m+j} \right) < 2$$

from which $N(g_m) \leq m + 1$. For our particular case, $m = 12\,193\,873$, and we have chosen $k = 19$. We have also found values of h_m which satisfy the required conditions, namely that for $k = 0, \ldots, 19$, the sequence h_{m+k} is strictly increasing, and $(-1)^{m+k} Z(g_{m+k} + h_{m+k}) > 0$. For these h_m we have $\sum_{j=1}^{k-1} h_{m+j} = 0.000$, so that

$$\begin{aligned} S(g_m) &\leq 1 + (g_{m+k} - g_m)^{-1} \left(2.30 + 0.128 \log\left(\frac{g_{m+k}}{2\pi}\right) + \sum_{j=1}^{k-1} h_{m+j} \right) \\ &= 1 + (6\,000\,009.156 - 6\,000\,000.486)^{-1} \left(2.30 + 0.128 \log\left(\frac{6\,000\,009.156}{2\pi}\right)\right) + 0.000 \\ &\doteq 1.469 \end{aligned} \tag{5.1}$$

That is, $S(g_m) < 2$, and so

$$\begin{aligned} S(g_m) &= N(g_m) - \frac{\vartheta(g_m)}{\pi} - 1 \leq 0 \\ \Rightarrow N(g_m) &\leq \frac{\vartheta(g_m)}{\pi} + 1 \\ \Rightarrow N(g_m) &\leq m + 1 \end{aligned}$$

Since we have located $m + 1$ zeros on the critical line, we may conclude that there are exactly $12\,193\,874$ zeros of $\zeta(s)$ for $0 < \Re(s) < 1$, $0 \leq \Im(s) \leq 6\,000\,000.485\,999$, and all of these are simple zeros which lie on the critical line.

5.3 Computation

The results presented in Section 5.2 were computed using the programs contained in the appendices. The root and Gram block counting routines are contained in Appendix C, while the program to produce the data needed for Turing's method is that of Appendix D.

The programs were written in the C programming language and compiled with the GNU gcc compiler. The hardware used was an Intel Pentium based PC with 16 MB RAM running at 200 MHz. The operating system was Red Hat Linux release 4.2 (Biltmore) Kernel 2.0.30.

The programs were compiled with all optimization switches turned on, however it should be noted that the computer code is by no means optimized. The code sacrificed some speed for readability and logical flow. In particular, parts of the routines could be accelerated by reducing the number of repeated function calls, or reducing the precision in steps where high precision is not necessary, such as Gram point calculation.

On Intel 80x86 based machines, the `long double` data type carries twenty digits of floating point precision in both computation and storage. The `int` data type will store integers of size up to $2^{31} - 1 = 2\,147\,483\,647$. Data underflow and overflow are discussed in later sections.

The main root counting computation took 24 hours and 24 minutes of dedicated CPU time. The Turing method calculation takes only seconds to compute the data for the selected range of 20 points.

5.4 Programs

In this section we will examine some of the practical points considered when coding the programs and functions which appear in the Appendices.

Each program and function has a complete description of its use in the header, and the code is heavily commented throughout to permit the user to trace through the logic.

5.4.1 The Function `C(n,z)`

To begin, we discuss the coefficient function of the remainder terms of the Riemann-Siegel formula, `C(n,z)`, where `n` is an integer from 0 to 4 and `z` is a `long double` variable between -1 and 1.

`C(n,z)` was coded by beginning with the function

$$\Psi(p) = \frac{\cos\left[2\pi\left(p^2 - p - 1/16\right)\right]}{\cos\left(2\pi p\right)}, \qquad 0 \le p < 1$$

from Section 3.2 and making the substitution $p = (z+1)/2$. Then

$$\Psi(p) \quad = \quad \Phi(z) \quad = \quad \frac{\cos\left[\left(z^2+3\right)\pi/8\right]}{\cos\left(\pi z\right)}$$

and $\Psi^{(n)}(p) = 2^n \Phi^{(n)}(z)$. These were then substituted into the expressions for C_n, $n = 0, \ldots, 4$, of Section 3.2, and the Taylor series of each of the C_n (as a function of z now) were computed using the symbolic computation package Maple. The Taylor series coefficients of the C_n were computed using thirty digit precision, and each of the series was truncated once the absolute value of the coefficients dropped below 10^{-20}. It is these coefficients which appear in `C(n,z)`.

5.4.2 The Function `Z(t,n)`

The next function we discuss is `Z(t,n)`, the main result of the Riemann-Siegel formula. In this implementation, `Z(t,n)` takes a `long double` argument `t` and an integer argument `n`. `t` is simply the real number at which Z is to be evaluated. `n` is the number of remainder terms to use in the computation. It is particularly useful to be able to vary `n` since in most cases, only the C_0 and C_1 term of the remainder are necessary to identify a root, and since `Z(t,n)` is the core of the main root counting program, reducing its computation time is an important consideration.

5.4.3 The Function `gram(n)`

Let us now move on to the function `gram(n)`, a function which locates the n^{th} Gram point using Newton's method. The n^{th} Gram point is defined to be the solution to the equation $\vartheta(t) = n\pi$, where $n \geq -1$ and $t > 7$. Recall that Newton's method will converge to a root of a function very quickly provided the function satisfies certain regularity conditions and the initial approximation to the root is reasonably accurate. In this case, from (3.1) we have for large t,

$$\vartheta(t) \approx \frac{t}{2}\left(\log\left(\frac{t}{2\pi}\right) - 1\right) - \frac{\pi}{8} \; .$$

Setting $\log(t/\left(2\pi\right)) - 1 = k$ and solving $kt/2 - \pi/8 = n\pi$ for t yields

$$t = \frac{2\pi}{k}n + \frac{\pi}{4k} \; . \tag{5.2}$$

Now for $1\,000\,000 \leq t \leq 6\,000\,000$, k ranges from approximately 11 to 13. Choosing an average of 12 for k and ignoring the $\pi/\left(4k\right)$ term of (5.2) then gives us the starting approximation $t \approx n/2$. The resulting Newton's method routine turns out to be very fast, converging to eight decimal places in four steps or less in most cases.

5.4.4 The Function `gramblock(n,m,FILE)`

The last function which is of interest is `gramblock(n,m,FILE)`, a function which searches for the expected number of roots in the Gram block $[g_n, g_m]$. This function uses what is essentially a bisection algorithm to count sign changes of $Z(t)$. The algorithm begins with a coarse partition of the Gram block, searches for sign changes, and then partitions and searches again until either the expected number of roots are finally found, or the limit on the number of partitions is reached. If the partition depth is reached without finding the expected number of roots $(m - n)$, a warning message is written to FILE indicating that the Gram block in question should be analyzed more carefully.

The algorithm actually proceeds in two stages. The first uses evaluations of $Z(t)$ with only the zeroth and first remainder terms. This first stage proceeds to no more than a depth of ten partitions before exiting if the expected number of roots is not found. If the root search in the first stage is unsuccessful, the second stage starts the search anew with more accurate $Z(t)$ values in which all five remainder terms are used. This second stage proceeds to at most a depth of sixteen partitions before exiting and writing the warning to the output file.

This function is our first encounter with error tolerance considerations. Namely, when searching for sign changes, it is not enough to verify that $Z(t_1) > 0$ at one point t_1 and $Z(t_2) < 0$ at its neighbour (in the partition) t_2, or vice versa. Rather, one must ensure that $Z(t_1) > \epsilon$ and $Z(t_2) < -\epsilon$ (or vice versa) where ϵ is the error in the approximation of Z. In `gramblock(n,m,FILE)`, the error tolerance used was that of Odlyzko[5] which is based on results of Gabcke: $\epsilon(t) \leq 0.053t^{-5/4}$ if $Z(t)$ is computed using the zeroth and first remainder terms.

5.4.5 The Main Root Counting Program

The program in Appendix C combines all of the functions discussed so far into the main root counting algorithm. The algorithm is quite straight-forward: successive Gram points are located, and Gram's Law is tested at these points. If Gram's Law is satisfied, then the root count is augmented by one. Otherwise, the next Gram point at which Gram's Law is satisfied is found, thus defining a Gram block, and this Gram block is passed to `gramblock(n,m,FILE)` for root counting. Once all roots in the input range have been counted, the program exits with a report of the number of roots found, the longest Gram block found, the number of Gram blocks found, and the number of roots in these Gram blocks.

The testing of Gram's Law at each Gram point is a two stage process. The first stage is a rough calculation of $Z(t)$ using only the zeroth and first remainder terms. A second more accurate calculation of $Z(t)$ is carried out only if Gram's Law in the form $(-1)^n Z(g_n) > \epsilon(g_n)$ fails at the less accurate value of $Z(t)$ from the first stage. The $\epsilon(t)$ function used here is the same as that used in `gramblock(n,m,FILE)`.

5.4.6 The Turing Method Program

This last program is used to determine the adjustments h_m used in Turing's method for bounding the function $S(t)$ (see Section 4.3). This program is a type of bisection algorithm which locates the adjustment terms near Gram points by stepping to the left or right of the Gram point until the h_m satisfying the required conditions are found. If a required h_m in the sequence cannot be found, the search interval is partitioned into smaller steps and the process begins again. In the current implementation, all required h_m for $m = 12\,193\,873, 12\,193\,874, \ldots, 12\,193\,892$ were found in the first partition level of step size 0.100.

5.5 Effective Domain of the Algorithms

With any numerical calculation algorithm, one must be able to state the range of input which will result in meaningful output, and the Riemann-Siegel approximation formula is no different in this respect. In the current implementation, if calculations could be performed in infinite precision, (and consequently the Taylor series in `C(n,z)` left untruncated), the results obtained for any search range would be absolutely correct. That is, the roots counted on the critical line would most definitely be roots. That is not to say, however, that all roots on the critical line would be found—it could happen that $Z(t)$ just barely crosses the t axis before changing direction and crossing back over again, and these zeros may not be detectable if $Z(t) < \epsilon(t)$ between the zeros. (Here $\epsilon(t)$ is the error tolerance discussed in Section 5.4.4).

The algorithms break down once the size of t becomes so large that the floating point calculations produce meaningless results. In the present case, a floating point representation of an input value to $Z(t)$ will have $20 - \lceil \log_{10} t \rceil$ digits to the right of the decimal point, and an error in the representation of $O(10^{-20}t)$. When this error is used in the evaluation of each term of the main cosine sum of $Z(t)$, the error remains about $O(10^{-20}t)$, and each of these $\lfloor \sqrt{t/(2\pi)} \rfloor$ cosine terms are added, giving an error of $O\big(10^{-20}t^{3/2}\big)$. So as t increases, the floating point accuracy of $Z(t)$ drops off rapidly, and assuming the error actually equals $10^{-20}t^{3/2}$, the floating point error matches the error tolerance $\epsilon(t)$ at $t \approx 6.5 \times 10^6$. Of course, $\epsilon(t)$ is based on using only two of the five remainder terms in the Riemann-Siegel formula, and is likely much smaller if determined for the case when all five remainder terms are used. Such an error tolerance was not determined in this case, and the more conservative estimate was used throughout.

Chapter 6
Conclusion and Future Work

From Riemann's early studies of the zeta function to the modern verification algorithms of Odlyzko and Schönhage[3], the Riemann-Siegel formula has been the basis upon which these studies have been founded. Though lacking in its ability to allow for arbitrary precision in calculations, this shortcoming in Riemann-Siegel is seldom a problem for verification purposes, and the sheer efficiency makes it the clear choice over other zeta function evaluation methods.

Despite the efficiency of the Riemann-Siegel formula, one eventually reaches the limits of computation space and time. In the current implementation, the results (for root counting) using twenty digit floating point precision become questionable once t reaches the top of the tested range, around $t = 6\,000\,000$. Even the calculations of Odlyzko[5] using twenty eight digit precision begin reaching their limit around the 10^{20}th zero of the zeta function where $t \approx 1.5 \times 10^{19}$.

The single most computationally expensive step in the computation of $Z(t)$ is in the evaluation of the $\lfloor \sqrt{t/(2\pi)} \rfloor$ terms of the main cosine sum $\sum_{n=1}^{N} n^{-1/2} \cos(\vartheta(t) - t \log n)$. Odlyzko's algorithms use a technique which splits this sum into a principal part which is evaluated term by term, and an approximated part which is evaluated using sophisticated approximation algorithms. This last method works well for computing many values of $Z(t)$, but requires a precomputation stage in order to perform the approximations of the main cosine sum. As noted in [3] , these last algorithms are no more efficient for computing a single value of $Z(t)$ than simple term by term evaluation of the Riemann-Siegel formula.

What is of particular interest for single evaluations of $Z(t)$, and hence $\zeta(1/2+it)$, is that

$$\sum_{n=1}^{N} n^{-1/2}\cos(\vartheta(t) - t\log n) \quad = \quad \Re\left(e^{i\vartheta(t)}\sum_{n=1}^{N}\left(\frac{1}{n}\right)^{1/2+it}\right) \, .$$

The $e^{i\vartheta(t)}$ term is straightforward to compute. The sum of the $(1/n)^{1/2+it}$ terms, on the other hand, is labour intensive as before. If this sum could be approximated with an error of the same order as that of the Riemann-Siegel formula itself, then this would present an extension of great practical importance to our zeta function evaluation and root counting algorithms. Further, the increased efficiency derived from such an advance would make calculations using extended precision software feasible, and hence extend rigorous analysis of the zeta function to heights beyond current limits.

Appendix A
C Functions Used in Main Programs

This appendix lists the code for functions used by the main programs in the appendices which follow.

```
int even(int n)
/*************************************************************************
*                                                                        *
* function which returns -1 if argument is odd and +1 if argument is even*
*                                                                        *
*************************************************************************/
{
  if (n%2 == 0)
    return(1);
  else
    return(-1);
}
```

```
long double theta(long double t)
/*************************************************************************
*                                                                        *
* Approximation to theta(t)=Im{log[Pi(it/2-3/4)]}-t/2*log(pi)            *
*                                                                        *
*************************************************************************/
{
  const long double pi = 3.1415926535897932385L;
  return(t/2.0L*logl(t/2.0L/pi) - t/2.0L - pi/8.0L
         + 1.0L/48.0L/t + 7.0L/5760.0L/t/t/t);
}
```

```
long double gram(int n)
/**********************************************************************
*                                                                     *
* Function which locates n^th Gram point using Newton's Method.      *
*                                                                     *
**********************************************************************/
{
  long double t_n;                                 /* n^th approximation */
  long double t_n1;                                /* (n+1)^st approx.   */

  const long double pi = 3.1415926535897932385L;
  t_n=0.0L;
  t_n1=0.5*n+20.0L;

  while(fabsl(t_n-t_n1) > 0.00000001L)
    {
      t_n=t_n1;
      t_n1 = t_n-(t_n*log(t_n/pi/2.0L)/2.0L-t_n/2.0L-
                  pi/8.0L+1.0L/t_n/48.0L+7.0L/5760.0L/(t_n*t_n*t_n)-
                  ((long double) n)*pi)
                 /(log(t_n/pi/2.0L)/2.0L-1.0L/(t_n*t_n)/48.0L-
                  7.0L/1920.0L/(t_n*t_n*t_n*t_n));
    }
  return(t_n1);
}
```

```
long double C(int n, long double z)
/**********************************************************************
*                                                                     *
* Coefficients of remainder terms; n can range from 0 to 4.           *
*                                                                     *
**********************************************************************/
{
  if (n==0)
    return(.38268343236508977173L * powl(z, 0.0L)
          +.43724046807752044936L * powl(z, 2.0L)
          +.13237657548034352332L * powl(z, 4.0L)
          -.01360502604767418865L * powl(z, 6.0L)
          -.01356762197010358089L * powl(z, 8.0L)
          -.00162372532314446528L * powl(z,10.0L)
          +.00029705353733379691L * powl(z,12.0L)
          +.00007943300879521470L * powl(z,14.0L)
          +.00000046556124614505L * powl(z,16.0L)
          -.00000143272516309551L * powl(z,18.0L)
          -.00000010354847112313L * powl(z,20.0L)
          +.00000001235792708386L * powl(z,22.0L)
          +.00000000178810838580L * powl(z,24.0L)
          -.00000000003391414390L * powl(z,26.0L)
          -.00000000001632663390L * powl(z,28.0L)
          -.00000000000037851093L * powl(z,30.0L)
          +.00000000000009327423L * powl(z,32.0L)
          +.00000000000000522184L * powl(z,34.0L)
          -.00000000000000033507L * powl(z,36.0L)
          -.00000000000000003412L * powl(z,38.0L)
          +.00000000000000000058L * powl(z,40.0L)
          +.00000000000000000015L * powl(z,42.0L));
  else if (n==1)
    return(-.02682510262837534703L * powl(z, 1.0L)
           +.01378477342635185305L * powl(z, 3.0L)
           +.03849125048223508223L * powl(z, 5.0L)
           +.00987106629906207647L * powl(z, 7.0L)
           -.00331075976085840433L * powl(z, 9.0L)
           -.00146478085779541508L * powl(z,11.0L)
           -.00001320794062487696L * powl(z,13.0L)
           +.00005922748701847141L * powl(z,15.0L)
           +.00000598024258537345L * powl(z,17.0L)
           -.00000096413224561698L * powl(z,19.0L)
           -.00000018334733722714L * powl(z,21.0L)
           +.00000000446708756272L * powl(z,23.0L)
           +.00000000270963508218L * powl(z,25.0L)
           +.00000000007785288654L * powl(z,27.0L)
```

```
              -.00000000002343762601L * powl(z,29.0L)
              -.00000000000158301728L * powl(z,31.0L)
              +.00000000000012119942L * powl(z,33.0L)
              +.00000000000001458378L * powl(z,35.0L)
              -.00000000000000028786L * powl(z,37.0L)
              -.00000000000000008663L * powl(z,39.0L)
              -.00000000000000000084L * powl(z,41.0L)
              +.00000000000000000036L * powl(z,43.0L)
              +.00000000000000000001L * powl(z,45.0L));
  else if (n==2)
    return(+.00518854283029316849L * powl(z, 0.0L)
              +.00030946583880634746L * powl(z, 2.0L)
              -.01133594107822937338L * powl(z, 4.0L)
              +.00223304574195814477L * powl(z, 6.0L)
              +.00519663740886233021L * powl(z, 8.0L)
              +.00034399144076208337L * powl(z,10.0L)
              -.00059106484274705828L * powl(z,12.0L)
              -.00010229972547935857L * powl(z,14.0L)
              +.00002088839221699276L * powl(z,16.0L)
              +.00000592766549309654L * powl(z,18.0L)
              -.00000016423838362436L * powl(z,20.0L)
              -.00000015161199700941L * powl(z,22.0L)
              -.00000000590780369821L * powl(z,24.0L)
              +.00000000209115148595L * powl(z,26.0L)
              +.00000000017815649583L * powl(z,28.0L)
              -.00000000001616407246L * powl(z,30.0L)
              -.00000000000238069625L * powl(z,32.0L)
              +.00000000000005398265L * powl(z,34.0L)
              +.00000000000001975014L * powl(z,36.0L)
              +.00000000000000023333L * powl(z,38.0L)
              -.00000000000000011188L * powl(z,40.0L)
              -.00000000000000000416L * powl(z,42.0L)
              +.00000000000000000044L * powl(z,44.0L)
              +.00000000000000000003L * powl(z,46.0L));
  else if (n==3)
    return(-.00133971609071945690L * powl(z, 1.0L)
              +.00374421513637939370L * powl(z, 3.0L)
              -.00133031789193214681L * powl(z, 5.0L)
              -.00226546607654717871L * powl(z, 7.0L)
              +.00095484999985067304L * powl(z, 9.0L)
              +.00060100384589636039L * powl(z,11.0L)
              -.00010128858286776622L * powl(z,13.0L)
              -.00006865733449299826L * powl(z,15.0L)
              +.00000059853667915386L * powl(z,17.0L)
              +.00000333165985123995L * powl(z,19.0L)
```

```
              +.00000021919289102435L * powl(z,21.0L)
              -.00000007890884245681L * powl(z,23.0L)
              -.00000000941468508130L * powl(z,25.0L)
              +.00000000095701162109L * powl(z,27.0L)
              +.00000000018763137453L * powl(z,29.0L)
              -.00000000000443783768L * powl(z,31.0L)
              -.00000000000224267385L * powl(z,33.0L)
              -.00000000000003627687L * powl(z,35.0L)
              +.00000000000001763981L * powl(z,37.0L)
              +.00000000000000079608L * powl(z,39.0L)
              -.00000000000000009420L * powl(z,41.0L)
              -.00000000000000000713L * powl(z,43.0L)
              +.00000000000000000033L * powl(z,45.0L)
              +.00000000000000000004L * powl(z,47.0L));
  else
    return(+.00046483389361763382L * powl(z, 0.0L)
              -.00100566073653404708L * powl(z, 2.0L)
              +.00024044856573725793L * powl(z, 4.0L)
              +.00102830861497023219L * powl(z, 6.0L)
              -.00076578610717556442L * powl(z, 8.0L)
              -.00020365286803084818L * powl(z,10.0L)
              +.00023212290491068728L * powl(z,12.0L)
              +.00003260214424386520L * powl(z,14.0L)
              -.00002557906251794953L * powl(z,16.0L)
              -.00000410746443891574L * powl(z,18.0L)
              +.00000117811136403713L * powl(z,20.0L)
              +.00000024456561422485L * powl(z,22.0L)
              -.00000002391582476734L * powl(z,24.0L)
              -.00000000750521420704L * powl(z,26.0L)
              +.00000000013312279416L * powl(z,28.0L)
              +.00000000013440626754L * powl(z,30.0L)
              +.00000000000351377004L * powl(z,32.0L)
              -.00000000000151915445L * powl(z,34.0L)
              -.00000000000008915418L * powl(z,36.0L)
              +.00000000000001119589L * powl(z,38.0L)
              +.00000000000000105160L * powl(z,40.0L)
              -.00000000000000005179L * powl(z,42.0L)
              -.00000000000000000807L * powl(z,44.0L)
              +.00000000000000000011L * powl(z,46.0L)
              +.00000000000000000004L * powl(z,48.0L));
}
```

```
long double Z(long double t, int n)
/***********************************************************************
*                                                                      *
* The Z(t) function from the Riemann-Siegel formula. This functions takes*
* an additional integer argument which is the number of terms to use in *
* the remainder. This integer argument can vary from 0 to 4 which       *
* corresponds to the first through fifth remainder terms.              *
*                                                                      *
***********************************************************************/
{
  long double ZZ;                 /* the result                       */
  long double tt;                 /* theta(t)                         */
  long double p;                  /* fractional part of sqrt(t/(2.0*pi))*/
  long double theta(long double); /* theta function                   */
  long double C(int,long double); /* coefficient of (2*pi/t)^(k*0.5)  */
  long double R;                  /* remainder term                   */
  int even(int);                  /* -1,+1 parity function            */
  int j;                          /* summation index for Z(t) function */
  int k;                          /* summation index for remainder term */
  int N;                          /* integer part of sqrt(t/(2.0*pi))  */

  const long double pi = 3.1415926535897932385L;  /* initializations... */
  ZZ    =  0.0L;                                  /*                   */
  R     =  0.0L;                                  /*                   */
  j     =  1;                                     /*                   */
  k     =  0;                                     /*                   */
  N     =  sqrtl(t/(2.0L * pi));                  /*                   */
  p     =  sqrtl(t/(2.0L * pi)) - N;              /*                   */
  tt    =  theta(t);                              /*                   */

  while (j <= N)                                  /* add up terms of   */
    {                                             /* main series...    */
      ZZ  = ZZ + 1.0L/sqrtl((long double) j )     /*                   */
               * cosl(fmodl(tt                    /*                   */
                      -t*logl((long double) j),   /*                   */
                      2.0L*pi));                  /*                   */
      ++j;                                        /*                   */
    }                                             /*                   */
  ZZ = 2.0L * ZZ;                                 /*                   */
```

```
  while (k <= n)                                   /* add up terms of    */
    {                                              /* remainder...       */
      R = R + C(k,2.0L*p-1.0L)                     /*                    */
            * powl(2.0L*pi/t,                      /*                    */
                   ((long double) k)*0.5L);        /*                    */
      ++k;                                         /*                    */
    }                                              /*                    */
  R = even(N-1) * powl(2.0L * pi / t,0.25L) * R;  /*                    */

  return(ZZ + R);
}
```

```
int gramblock(int n, int m, FILE *outfile)
/**********************************************************************
*                                                                     *
* Function which searches for the expected number of roots between Gram  *
* points n and m. Warnings of possible missed roots along with the    *
* Gram block are written to outfile for further analysis             *
*                                                                     *
**********************************************************************/
{
  int num;                          /* root counter                       */
  int p;                            /* partitions between gram points     */
  int j;                            /* index of current Gram point        */
  int k;                            /* index of current partition point   */

  long double Z(long double,int); /* Riemann-Siegel Z function          */
  long double gram(int);          /* Gram point evaluation function     */
  long double gp1;                /* Gram interval partition point      */
  long double gp2;                /* Gram interval partition point      */

  p   = 1;
  num = 0;

  while (num < (m - n) && p < 1024)              /* while no. roots    */
    {                                            /* less than          */
                                                 /* expected...        */

      num = 0;                                   /* refine inter-      */
      p = 2*p;                                   /* Gram point         */
      j = n;                                     /* partition; go to   */
      gp2 = graham(j);                           /* beginning of       */
                                                 /* Gram block...

      while (j <= (m-1))                         /* for each Gram pt.  */
        {                                        /* except the last... */

          k = 0;                                 /* step through       */
          while (k <= (p-1))                     /* successive         */
            {                                    /* partition points...*/

              gp1 =  gp2;                        /* determine          */
                                                 /* coordinates of     */
              gp2 =  gram(j)                     /* adjacent partition */
                   * (1.0L-((long double) (k+1)) /* points...          */
                   / ((long double) p))          /*                    */
                   +                             /*                    */
```

```
                 gram(j+1)                      /*                   */
               * ((long double) (k+1))          /*                   */
               / ((long double) p);             /*                   */

          if (Z(gp1,1)                          /* roughly evaluate  */
            > 0.053L/powl(gp1,1.25L)            /* Z(t) at adjacent  */
            &&                                  /* partition points...*/
              Z(gp2,1)                          /*                   */
            < -0.053L/powl(gp2,1.25L))          /* if the signs      */
            ++num;                              /* differ increment  */
          else if (Z(gp1,1)                     /* the root count... */
                 < -0.053L/powl(gp1,1.25L)      /*                   */
                 &&                             /*                   */
                   Z(gp2,1)                     /*                   */
                 > 0.053L/powl(gp2,1.25L))      /*                   */
            ++num;                              /*                   */

          ++k;                                  /* next partition pt. */
        }
      ++j;                                      /* next Gram point   */
    }
  }

p = 1;                                          /* reset partition   */
                                                /* size...
while (num < (m - n))                           /* if no. roots still */
  {                                             /* < expected, start */
                                                /* over using more   */
                                                /* accurate Z(t)...  */

    num = 0;                                    /* refine inter-     */
    p = 2*p;                                    /* Gram point        */
    j = n;                                      /* partition; go to  */
    gp2 = graham(j);                            /* beginning of      */
                                                /* Gram block        */

    while (j <= (m-1))                          /* for each Gram pt. */
      {                                         /* except the last... */

        k = 0;                                  /* step through      */
        while (k <= (p-1))                      /* successive        */
          {                                     /* partition points...*/

            gp1 =  gp2;                         /* determine         */
```

```
                                               /* coordinates of     */
          gp2 =  gram(j)                       /* adjacent partition */
              * (1.0L-((long double) (k+1))  /* points...          */
              / ((long double) p))           /*                    */
              +                              /*                    */
                gram(j+1)                    /*                    */
              * ((long double) (k+1))        /*                    */
              / ((long double) p);           /*                    */

          if (Z(gp1,4)                         /* accurately         */
            > 0.053L/powl(gp1,1.25L)           /* evaluate Z(t) at   */
            &&                                 /* adjacent partition */
              Z(gp2,4)                         /* points...          */
            < -0.053L/powl(gp2,1.25L))         /*                    */
            ++num;                             /* if the signs       */
          else if (Z(gp1,4)                    /* differ, increment  */
                < -0.053L/powl(gp1,1.25L)      /* the root count...  */
                &&                             /*                    */
                  Z(gp2,4)                     /*                    */
                > 0.053L/powl(gp2,1.25L))      /*                    */
            ++num;                             /*                    */

          ++k;                                 /* next partition pt. */
        }
      ++j;                                     /* next Gram point    */
    }
    if (p == 65536) break;                     /* break if partition */
  }                                            /* depth getting too  */
                                               /* deep               */

 if (num < (m - n))                            /* if unable to find  */
   {                                           /* expected number of */
     fprintf(outfile,                          /* roots, write       */
             "Partition depth reached in"      /* warning to output  */
             " Gram block [%d,%d],\n"          /* file               */
             " possible violation of"          /*                    */
             " Rossers rule\n",n,m);           /*                    */
   }
 return(num);
}
```

Appendix B

C Program for Simple Evaluation of $Z(t)$

The following program produces $(t, Z(t))$ pairs appropriate for input to a plotting utility.

```
/**********************************************************************
*                                                                     *
* Program to implement the Riemann-Siegel formula to compute          *
* approximations to Z(t).                                             *
*                                                                     *
* Given the input LOWER, UPPER and NUMSAMPLES, the program computes    *
* NUMSAMPLES coordinate pairs (t,Z(t)) for LOWER <= t <= UPPER and     *
* writes the results to the standard output.                          *
*                                                                     *
* Five of the error terms of the approximation are used for all       *
* calculations.                                                       *
*                                                                     *
**********************************************************************/

#include <stdio.h>
#include <math.h>

#define LOWER      6000000.0L     /* lower bound of t domain            */
#define UPPER      6000010.0L     /* upper bound of t domain            */
#define NUMSAMPLES      10000     /* number of samples to compute       */

/**********************************************************************/
```

```
int main()
{
  long double t1   ;              /* start of plotted interval          */
  long double t2   ;              /* end of plotted interval            */
  long double t    ;              /* sample to compute                  */
  long double Z(long double,int); /* Riemann-Siegel Z(t) function       */


  int samples    ;                /* number samples to compute          */
  int sampleindex;                /* index of sample being computed     */

  t1          = LOWER         ;
  t2          = UPPER         ;
  samples     = NUMSAMPLES - 1;
  sampleindex = 0             ;

  while (sampleindex <= samples)
    {
      t = t1 + 1.0L*sampleindex/samples*(t2 - t1);
      printf("%Lf\t%16.12Lf\n",t,Z(t,4));
      ++sampleindex;
    }

  return(0);
}
```

Appendix C

C Program for Counting Roots of $Z(t)$

The following root counting program uses the Riemann-Siegel formula to establish the existence of roots of $Z(t)$ in the given input range.

```
/**************************************************************************
*                                                                         *
* Program to verify the Riemann hypothesis up to Gram point END. The     *
* main routine uses the Riemann-Siegel formula for fast evaluation of    *
* the zeta function.                                                      *
*                                                                         *
* This program counts the number of zeros of the Riemann zeta function   *
* between Gram points BEGIN and END. The program establishes the         *
* existence of a zero with complete certainty, and thus produces a lower *
* bound on the number of zeros in the search range. There is the         *
* possibility that a zero will be missed in the case where |Z(t)| is     *
* less than the error tolerance, and hence the sign of Z(t) cannot be    *
* determined with certainty. In these situations, a warning is written   *
* to the RESULTFILE indicating that the questionable region should be    *
* examined with more accuracy.                                            *
*                                                                         *
* Output to the RESULTFILE consists of the count of roots found, the     *
* number of Gram blocks found, the total number of roots within these    *
* Gram blocks, and the longest and shortest Gram blocks found.           *
*                                                                         *
**************************************************************************/

#include <stdio.h>
#include <math.h>
```

```
#define BEGIN                  -1  /* initial Gram point              */
#define END              12193873  /* final Gram point                */
#define RESULTFILE  "results.txt"  /* output file for results         */

/*********************************************************************/

int main()
{
  FILE *results_file;              /* output file for results         */

  long double Z(long double,int);  /* Riemann-Siegel Z(t) function    */
  long double gram(int);           /* Gram point location function    */
  int gramblock(int,int,FILE *);   /* Gram block root count function  */
  int even(int);                   /* -1,+1 parity function           */

  int gpt_final;                   /* index of final Gram point       */
  int gpt_indx;                    /* index of current Gram point     */
  int tot_roots;                   /* root counter                    */

  int gblock_start;                /* current Gram block lower index  */
  int gblock_end;                  /* current Gram block upper index  */
  int gblock_roots;                /* roots in current Gram block     */
  int tot_gblocks;                 /* total Gram blocks found         */
  int tot_gblock_roots;            /* total Gram block roots found    */
  int l_gblock_start;              /* longest Gram block left index   */
  int l_gblock_end;                /* longest Gram block right index  */

  gpt_indx         = BEGIN;                       /* initializations  */
  gpt_final        = END;                         /*                  */
  tot_roots        = 0;                           /*                  */
  gblock_roots     = 0;                           /*                  */
  tot_gblocks      = 0;                           /*                  */
  tot_gblock_roots = 0;                           /*                  */
  l_gblock_start   = -1;                          /*                  */
  l_gblock_end     = -1;                          /*                  */

  results_file = fopen(RESULTFILE, "w");          /* open output file */

  while (gpt_indx < gpt_final)                    /* while not at last */
    {                                             /* Gram point...    */

      ++gpt_indx;                                 /* go to next one   */

      if (even(gpt_indx)*Z(gram(gpt_indx),1)      /* if Gram's Law    */
```

```
    > 0.053L/powl(gram(gpt_indx),1.25L))      /* satified,          */
    ++tot_roots;                              /* increment total    */
                                              /* number of roots... */

  else                                        /* otherwise, try     */
    if (even(gpt_indx)*Z(gram(gpt_indx),4)    /* more precise       */
      > 0.053L/powl(gram(gpt_indx),1.25L))    /* calculation of     */
    ++tot_roots;                              /* Z(t)...            */

  else                                        /* still not          */
    {                                         /* satisfied? then we */
                                              /* have a Gram        */
                                              /* block...           */

      gblock_start = gpt_indx - 1;            /* identify left      */
                                              /* index of block...  */

      while (even(gpt_indx)                   /* test succeeding    */
             *Z(gram(gpt_indx),1)             /* Gram points        */
           <=                                 /* until Gram's Law   */
             0.053L/powl(gram(gpt_indx),      /* once again         */
                         1.25L))              /* satisfied...       */
        ++gpt_indx;                           /*                    */

      gblock_end = gpt_indx;                  /* identify right     */
                                              /* index of block     */

      gblock_roots =                          /* determine number   */
        gramblock(gblock_start,gblock_end,    /* of roots inside    */
                  results_file);              /* the Gram block...  */

      ++tot_gblocks;                          /* increment root and */
      tot_gblock_roots += gblock_roots;       /* Gram block         */
      tot_roots += gblock_roots;              /* counters           */

      if (gblock_end-gblock_start             /* determine if the   */
        > l_gblock_end-l_gblock_start)        /* new Gram block     */
        {                                     /* is longest so far  */
          l_gblock_start = gblock_start;      /*                    */
          l_gblock_end = gblock_end;          /*                    */
        }                                     /*                    */
    }
  if (gpt_indx%1000 == 0)                     /* write progress     */
    printf("%d roots located to Gram point"   /* message to screen  */
           " %d\n",                           /*                    */
```

```
              tot_roots,gpt_indx);              /*                    */
    }
  fprintf(results_file,"Number of roots to"       /* write results to   */
          " Gram point N = %d is %d\n",           /* output file:       */
          gpt_indx,tot_roots);                    /* ...total roots,    */
  fprintf(results_file,"Number of Gram"           /*                    */
          " blocks found is %d\n",                /* ...total Gram      */
          tot_gblocks);                           /*    blocks,         */
  fprintf(results_file,"Total number of roots"    /*                    */
          " in these %d Gram blocks is %d\n",     /* ...total roots in  */
          tot_gblocks,tot_gblock_roots);          /*    Gram blocks,    */
  fprintf(results_file,"Longest Gram block"       /*                    */
          " found is [%d, %d]\n",                 /* ...longest Gram    */
          l_gblock_start,l_gblock_end);           /*    block           */

  fclose(results_file);                           /* close output file  */

  return(0);
}
```

Appendix D
C Program for Locating Turing Adjustments

This program searches for the adjustments h_m used in Turing's method for determining the number of roots in the critical strip up to height T.

```
/**********************************************************************
*                                                                     *
* Program to find the adjustments h_m used in Turing's method for     *
* bounding S(g_n)= N(g_n) - theta(g_n) / pi - 1.                      *
*                                                                     *
* For g_END, the program finds values of h_(END-SAMPLES+1), ..., h_END  *
* such that                                                           *
*                                                                     *
* 1.)  g_(END-SAMPLES+1)+h_(END-SAMPLES+1), ..., g_END+h_END is       *
*      strictly increasing, and                                       *
*                                                                     *
* 2.)  (-1)^n*Z(g_n+h_n) > 0 for n = END-SAMPLES+1, ..., END.         *
*                                                                     *
* Output to the RESULTFILE consists of n, g_n, h_n, g_n+h_n, Z(g_n+h_n), *
* for n = END-SAMPLES+1, ..., END.                                    *
*                                                                     *
**********************************************************************/

#include <stdio.h>
#include <math.h>

#define END             12193892  /* final Gram point                 */
#define SAMPLES               20  /* number of samples to compute     */
#define RESULTFILE   "turing.txt"  /* output file for results          */
```

```
/*******************************************************************/

int main()
{
  FILE *results_file;            /* output file for results           */

  long double Z(long double,int); /* Riemann-Siegel Z(t) function      */
  long double gram(int);          /* Gram point location function      */
  long double g[SAMPLES];         /* list of Gram points               */
  long double h[SAMPLES];         /* list of adjustment terms          */
  long double h_total;            /* total of adjustment terms         */
  long double step;               /* step size for adjustment terms    */
  int g_initial;                  /* index of first Gram point sample  */
  int even(int);                  /* -1,1 parity function              */
  int num_samples;                /* number of samples for test        */
  int j;                          /* loop index                        */
  int k;                          /* loop index                        */
  int n;                          /* loop index                        */
  int m;                          /* loop index                        */

  num_samples = SAMPLES;                          /*                   */
  j = 0;                                          /*                   */
  k = 0;                                          /*                   */
  m = 0;                                          /* initializations... */
  g_initial = END - num_samples + 1;              /*                   */
  step      = 0.1L;                               /*                   */
  h_total   = 0.0L;                               /*                   */

  results_file = fopen(RESULTFILE, "w");          /* open output file...*/

  while (j < num_samples)                         /* evaluate Gram     */
    {                                             /* points in range;  */
      g[j] = gram(g_initial+j);                   /* initialize        */
      h[j] = 0.0L;                                /* adjustment        */
      ++j;                                        /* vector...         */
    }                                             /*                   */

  while (k < num_samples)                         /* for each Gram     */
    {                                             /* point ...         */
      n = 0;                                      /* set number of     */
                                                  /* steps to 0...     */

      while (even(g_initial+k)*Z(g[k]-n*step,4)   /* keep stepping one  */
             < 0.053L/powl(g[k]-n*step,1.25L)     /* step to both the   */
          &&                                      /* left and right of  */
```

```
            even(g_initial+k)*Z(g[k]+n*step,4)   /* g_n until          */
            < 0.053L/powl(g[k]+n*step,1.25L))    /* (-1)^n*Z(g_n+h_n)>0*/
        {                                        /* is satisfied,      */
          ++n;                                   /* where              */
        }                                        /* h_n = +/- n*step...*/

      if (even(g_initial+k)*Z(g[k]-n*step,4)     /* store the minimal  */
          > 0.053L/powl(g[k]-n*step,1.25L))      /* adjustment found   */
        h[k] = -n*step;                          /* in the h[] vector  */
      else                                       /*                    */
        h[k] = n*step;                           /*                    */

      h_total += h[k];                           /* increase adj. total*/

      if ((k > 0) && (g[k]+h[k]<=g[k-1]+h[k-1])) /* if the sequence    */
          {                                      /* g_k+h_k so far is  */
            h_total = 0.0L;                      /* not strictly       */
            step = step/2.0L;                    /* increasing, reduce */
            k = 0;                               /* the step size and  */
          }                                      /* start over,        */
      else ++k;                                  /* otherwise go to    */
                                                 /* the next Gram pt   */
    }
  while (m < num_samples)
    {                                            /* write out the      */
      fprintf(results_file,                      /* results            */
              "g[%d]=%14.6Lf,"                   /*                    */
              "h[%d]=%14.6Lf,"                   /*                    */
              "g[%d]+h[%d]=%14.6Lf,"             /*                    */
              "Z(g[%d]+h[%d])=%14.6Lf\n",        /*                    */
              g_initial+m,g[m],                  /*                    */
              g_initial+m,h[m],                  /*                    */
              g_initial+m,g_initial+m,           /*                    */
              g[m]+h[m],                         /*                    */
              g_initial+m,g_initial+m,           /*                    */
              Z(g[m]+h[m],4));                   /*                    */
      ++m;                                       /*                    */
    }                                            /*                    */
  fprintf(results_file,"h_total = %14.6Lf\n",    /*                    */
          h_total);                              /*                    */

  fclose(results_file);                          /* close output file  */

  return(0);
}
```

Bibliography

[1] H. M. Edwards, *Riemann's Zeta Function*, Academic Press, 1974.

[2] J. Van de Lune, H.J.J. te Riele, and D.T. Winter, On the Zeros of the Riemann Zeta Function in the Critical Strip. IV., *Math. Comp.* 46. (1986), 667-681.

[3] A.M. Odlyzko and A.Schönhage, Fast Algorithms for Multiple Evaluations of the Riemann Zeta Function, *Trans. Amer. Math. Soc. 309* (1988), 797-809.

[4] A. M. Odlyzko, Analytic computations in number theory, *Mathematics of Computation 1943-1993: A Half-Century of Computational Mathematics*, W. Gautschi (ed.), Amer. Math. Soc., Proc. Symp. Appl. Math. #48 (1994), pp. 451-463.

[5] A.M. Odlyzko, *The* 10^{20}*-th Zero of the Riemann Zeta Function and 175 Million of its Neighbors*, Manuscript in Preparation (see `http://www.research.att.com/~amo/unpublished/index.html`.)

[6] E.C. Titchmarsh, *The Theory of the Riemann Zeta-function*, 2nd ed., Oxford University Press, 1986.