An Investigation and Visualisation of the Riemann Hypothesis

Jack Morgan

A-Level Computer Science Coursework

Dr Challoner's Grammar School
Centre Number: 52205
Candidate Number: 7469
April 2022

Contents

1 Analysis

1.1 Introduction

1.1.1 Project Proposal

Throughout this coursework, I will be examining the Riemann Hypothesis; to be able to visualise this important conjecture and make its complex mathematical structures accessible for anyone who is interested in mathematics. This project will be heavily based on how to compute and plot recursive mathematical functions on the complex plane and test whether the Riemann hypothesis is a true statement, as well as how the hypothesis affects mathematics today.

1.1.2 Abstract

During this coursework, I aim to be able to make the Riemann Hypothesis a more accessible mathematical concept to be able to help people to understand it and hopefully inspire them to research further into related mathematical subjects. I will do this in 3 simple ways:

1. By detailing what the Riemann hypothesis is and why it is so important,
2. Investigating the major functions that make up the Riemann Hypothesis
3. By Visualising the significant concepts of the Hypothesis, in order to make them more understandable; and to be able to investigate what they are and how they work.

I will be conducting my investigation by programming the key functions of the Hypothesis in Python 3. This will allow me to:

1. Compute key mathematical functions
2. Plot graphs of data
3. Store data in files and a database
4. Have a graphical user interface

1.1.3 End Users

This project is aimed at people who are interested in mathematics. It is a great way to inspire people to delve deeper into maths and number theory; as well as help those with a more advanced understanding of mathematics. Although the mathematics and understanding behind the problem can be complex at times, I am for this project to be able to be used by anyone who will want to use it. This would require it to be simple for people to understand and navigate. I aim to get input from people with a range of mathematical abilities to give me feedback on the project.

1.2 Research

1.2.1 What is the Riemann Hypothesis?

The Riemann Hypothesis - first proposed by Bernhard Riemann in 1859 - is considered by many to be one the most important unsolved problems in mathematics. To understand why the Riemann hypothesis is so important, it is necessary to understand what the Riemann hypothesis is, and how it is used in many mathematical and scientific fields. The Riemann hypothesis was proposed by Bernhard Riemann in his paper "On the Number of Primes Less Than a Given Magnitude". This paper deeply explores the prime numbers, and the functions used to estimate and count prime numbers. Throughout this paper, Riemann discussed definitions and proofs of multiple theories that all relate to the prime numbers. But most notably, he had taken a function previously mentioned by Leonhard Euler (a famous mathematician who lived around 100 years prior), and formalised this function into the Riemann Zeta Function.

Riemann took Euler's function and used a process called analytic continuation to extend the domain of this function to all numbers - be that real, imaginary or complex. Just like every other function, the Riemann Zeta Function contains what are called 'zeros' or 'roots'. This is where a function's output is equal to zero. These zeros can be described as trivial (where the explanation for why these zeros exist is almost intuitive), or non-trivial (where it can be hard to explain why these zeros occur.) The Riemann Zeta Function has trivial zeros when the input is a negative even integer. I will explore why this is the case later on.

However, the non-trivial zeros of the zeta function can be found when the real part of the function's input is between 0 and 1 (called the critical region). This is a fact that has been proven by Riemann in his paper, and he even proved that there are an infinite amount of non-trivial zeros in the critical region. But Riemann went further than this, he hypothesised that the non-trivial zeros do not only occur when the real part of the input is between 0 and 1; but they all occur in the middle of the critical region when the real part of the input is exactly $\frac{1}{2}$. The Riemann Hypothesis states that 'the real part of every nontrivial zero of the Riemann zeta function is $\frac{1}{2}$'. This is the fundamental part of the Riemann Hypothesis, and although it is widely considered to be true, this conjecture has never actually been proven with conclusive evidence, and that is why these are considered to be non-trivial zeros.

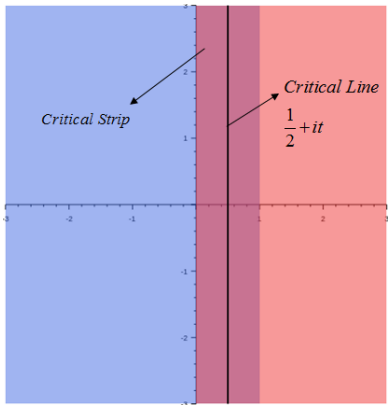


Figure 1: Critical Strip and Line

1.2.2 The Importance of the Riemann Hypothesis

The Riemann Hypothesis and the Riemann Zeta Function have many extraordinary uses. From calculating prime numbers to uses in Quantum Physics, and even cryptography with the RSA algorithm. This is due to the fact that the Zeta function is deeply connected to the prime numbers.

One of the main reasons why the Riemann Hypothesis is significant is because there have been many conjectures and theories that assume the Riemann hypothesis to be true. So if the Riemann hypothesis was proven, this would also prove countless other theories. Some of these theories include :

- The weak Goldbach conjecture - stating that all integers greater than 5 are the sum of three primes
- Mills' constants - numbers that allow you to generate prime numbers
- The theory that there will always be at least one prime between consecutive cubes
- The theory that there is a maximum bound between consecutive prime numbers

All of these theories are, in some way, related to the prime numbers. There is an important reason for this which I'll cover in a later section.

If the Riemann Hypothesis was proven to be true, there would be profound effects in fields such as cryptography. In public-key cryptosystems, like RSA, public keys are created using prime numbers. If someone wanted to try and decode information without knowing what the prime numbers were that created the key, they would have to try and guess what the prime numbers were. What keeps these algorithms secure is the fact that it can take a long time to calculate the prime numbers, especially some of the larger ones. In fact, the largest prime number discovered has over 23 million digits. However, if the Riemann hypothesis were true, then people would be able to calculate the prime numbers much quicker than before. This would make many of the current cryptography algorithms obsolete.

The Riemann Hypothesis does not just influence mathematics and cryptography, it also has great importance in quantum physics. It was discovered in 1996 that the arrangement of the zeta zeros exhibits the same statistical pattern as the spectra of energy levels (that is the possible values of energy of a quantum system) in quantum chaotic systems. Furthermore, it was conjectured in 1999 by Michael Berry and Jonathan Keating, that there will exist a quantum system, where the energy levels will correspond exactly to the non-trivial zeros of the Riemann Zeta Function. If this conjecture is true, it would prove the Riemann Hypothesis.

1.2.3 Complex Numbers and Key Operations

Complex numbers play a key part in the Riemann Hypothesis. When Riemann created the Riemann Zeta Function, he allowed the inputs and outputs to be complex numbers, through his ideas of analytic continuation. It is therefore important to know what complex numbers are, and how to do arithmetic with them.

To first understand complex numbers (denoted by the symbol $\mathbb{C}$), it is necessary to be familiar with the idea of imaginary numbers. There is no real number whose square root is a negative number, so mathematicians decided to invent numbers for this. It is denoted by the symbol i, for the imaginary unit.

Where:

$$i \equiv \sqrt{-1}$$

And thus:

$$i^2 \equiv -1$$

You can multiply the imaginary unit(i) by any real number to create an imaginary number. For example:

$$3i$$

$$2i$$

Which are both imaginary numbers. By combining both real numbers and imaginary numbers, you can create complex numbers. For example:

$$2 + 3i$$

Where the real part of the number is 2, and the imaginary part is $3i$.

$$4 - 7i$$

Where the real part of the number is 4, and the imaginary part is $-7i$. We often represent real numbers on a number line, however, this is impossible to do with complex numbers. Instead, one way to understand them is through a two-dimensional graph, or to give it its formal name, the complex plane. On the complex plane, there are two axes, a real axis and an imaginary axis. Any complex number can be plotted on the complex plane. We can think of a complex number as a set of coordinates that tell us where the number lies on the complex plane, with the real part being the x coordinate and the imaginary part being the y coordinate.

So if we wanted to plot the point $2 + 3i$ on the complex plane, it would look as follows:

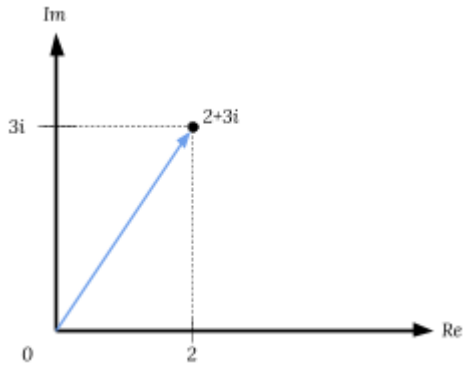


Figure 2: Argand Diagram of $2 + 3i$

This type of diagram is known as an Argand Diagram. And as you can see, we have plotted the point $2 + 3i$ at the coordinates $(2, 3)$ on the complex plane.

We can also do arithmetic with complex numbers - be that addition, subtraction, multiplication, division or exponentiation.

The sum of two complex numbers is the sum of both of the real parts of the numbers, plus the sum of the two complex parts of the numbers.

$$\begin{aligned}(3 + 2i) + (1 + 3i)\\ = (3 + 1) + (2i + 3i)\\ = 4 + 5i\end{aligned}$$

Subtraction is also done using a similar method.

$$\begin{aligned}(3 + 2i) - (1 + 3i)\\ = (3 + 2i) + (-1 - 3i)\\ = (3 - 1) + (2i - 3i)\\ = 2 - i\end{aligned}$$

To multiply two complex numbers, it is equivalent to expanding out two binomial brackets where:

$$\begin{aligned}(a + b)(c + d)\\ = ac + ad + bc + bd\end{aligned}$$

So with two complex numbers this would look like:

$$(3+i)(2+2i)$$
$$= 6 + 6i + 2i + 2i^2$$
$$= 6 + 8i + 2i^2$$

We can then use the identity $i^2 \equiv -1$ to simplify this expression, so that:

$$= 6 + 8i + 2(-1)$$
$$= 4 + 8i$$

But we can speed up this method by using the rule $(a+bi)(c+di) = (ac-bd)+(ad+bc)i$ For example:

$$(3+i)(2+2i)$$
$$= (3 \times 2 - 1 \times 2) + (3 \times 2 + 1 \times 2)i$$
$$= 4 + 8i$$

We can derive this rule, to prove that it works for all complex numbers:

$$RTP : (a+bi)(c+di) = (ac-bd) + (ad+bc)i$$
$$LHS = (a+bi)(c+di)$$

Distributing Terms:

$$= ac + adi + bci + bdi^2$$

Using $i^2 = -1$:

$$= ac + adi + bci - bd$$

Then rearranging:

$$= ac - bd + adi + bci$$
$$= (ac-bd) + (ad+bc)i$$
$$= RHS$$
$$Q.E.D$$

To divide two complex numbers, we create a fraction and then rationalise the denominator, by using the identity $(a+b)(a-b)a^2-b^2$ where $a, b \in \mathbb{C}$ (meaning that a and b are both complex numbers) For example:

$$\begin{aligned}(3-i) \div (2-2i) \\ = \frac{3-i}{2-2i}\end{aligned}$$

We can then rationalise the denominator by multiplying the numerator and denominator of the fraction by $(2+2i)$. This does not change the value of the expression but does allow for it to be simplified.

$$\begin{aligned}= \frac{3-i}{2-2i} \cdot \frac{2+2i}{2+2i} \\ = \frac{(3-i)(2+2i)}{(2-2i)(2+2i)}\end{aligned}$$

We can then distribute terms in the numerator and denominator, using the aforementioned multiplication rule:

$$\begin{aligned}= \frac{(6+2)+(6-2)i}{4-4i^2} \\ = \frac{8+4i}{4-4i^2} \\ = \frac{8+4i}{4-(-4)} \\ = \frac{8+4i}{8} \\ = 1-\frac{1}{2}i\end{aligned}$$

There are also some functions that we can use on complex numbers to talk about the real and imaginary parts separately. The $\Re(z)$ function (also defined as $Re(z)$), outputs the real part of the complex variable z. So for example:

If we let

$$z = a + bi$$

Then

$$\Re(z) = a$$

Similarly, the $\Im(z)$function (also defined as $Im(z)$) outputs the imaginary part of of the complex variable z. For example:

If we let

$$z = a + bi$$

Then

$$\Im(z) = b$$

Although these functions have limited use in equations and mathematical usage, they prove very handy when needing to discuss complex variables. It is also very useful when computing functions, to be able to split up complex numbers into their real and imaginary parts.

We can also express imaginary numbers using the trigonometric functions, sine and cosine.

Where we have the complex number such that:

$$z = a + bi$$

It can be expressed such that:

$$z = r \cdot cis(\phi)$$

Where:

$$\begin{aligned} cis(\phi) &= cos(\phi) + i \cdot sin(\phi) \\ r = |z| &= \sqrt{a^2 + b^2} \\ \phi = arg(z) &= arctan\left(\frac{b}{a}\right) \end{aligned}$$

This is known as the polar form of the complex number.

We can use this form to easily calculate the value when we raise a complex number to a real power, using De Moivre's Theorem. De Moivre's theorem states that for a complex number in the form

$$z = r(cos(\phi) + isin(\phi))$$

If we raise z to the power n then:

$$\begin{aligned} z^n &= (r(cos(\phi) + isin(\phi)))^n \\ &= r^n(cos(n\phi) + isin(n\phi)) \end{aligned}$$

Which is true for

$$r, \phi, n \in \mathbb{R}, z \in \mathbb{C}$$

Which means that this formula does not work when we are raising any number to a complex power.

Say we have two complex numbers z and w and they are of the form $a + bi$. Such that $z = a + bi$ and $w = c + di$.To calculate z^w, the process is best split down into multiple steps.

First we calculate ρ and θ, where ρ is the modulus of z and θ the argument of z

$$\begin{aligned} \rho &= \sqrt{a^2 + b^2} \\ \theta &= \arctan \frac{b}{a} \end{aligned}$$

Then using these values, we can say that:

$$z^w = \rho^c e^{-d\theta}(\cos(d \ln \rho + c\theta) + i \sin(d \ln \rho + c\theta))$$

Which is true for

$$z, w \in \mathbb{C}$$

1.2.4 The Riemann Zeta Function

Previously I have mentioned multiple functions that play a key role in the Riemann Hypothesis, but now I aim to address and examine these functions in detail.

When discussing the Riemann Zeta Function the complex variable s is traditionally used where $s = \sigma + it$.

The Riemann Zeta Functions is defined as $\zeta(s)$ where:

$$\begin{aligned} \zeta(s) &= \sum_{n=1}^{\infty} \frac{1}{n^s} \\ &= \frac{1}{1^s} + \frac{1}{2^s} + \frac{1}{3^s} + \frac{1}{4^s} + \dots \end{aligned}$$

This was the equation that Leonhard Euler first introduced and studied, but it was Bernhard Riemann who defined this function for not just all of the real numbers, but all of the complex numbers. Riemann did this through the method of analytic continuation to expand the domain of the function. But by doing it he had to create a new function that would work for all possible input values. He produced what is known as Riemann's Functional Equation:

$$\zeta(s) = 2^s \pi^{s-1} sin\left(\frac{\pi s}{2}\right) \Gamma(1-s)\zeta(1-s)$$

Where $\Gamma(n)$ is the gamma function, such that:

$$\Gamma(n) = \int_0^\infty n^{s-1} e^{-n} \mathrm{d}n$$

From Riemann's Functional Equation, we can now see why the trivial zeros occur at even negative integers for s. This is all due to the sine function. If we let $s = -2n$ for $n \in \mathbb{N}$, (a negative even integer) then the functional equation becomes:

$$\zeta(-2n) = 2^{-2n} \pi^{-2n-1} sin\left(\frac{\pi(-2n)}{2}\right) \Gamma(1-(-2n))\zeta(1-(-2n))$$
$$\Rightarrow \zeta(-2n) = -2^{-2n} \pi^{-(2n+1)} sin(n\pi)\Gamma(2n+1)\zeta(2n+1)$$

Now we can see that the functional equation includes a factor of $sin(n)$, but due to the nature of the sine function: $sin(n) = 0$ for $n \in \mathbb{Z}$. As shown in this diagram of the function $y = sin(x)$:

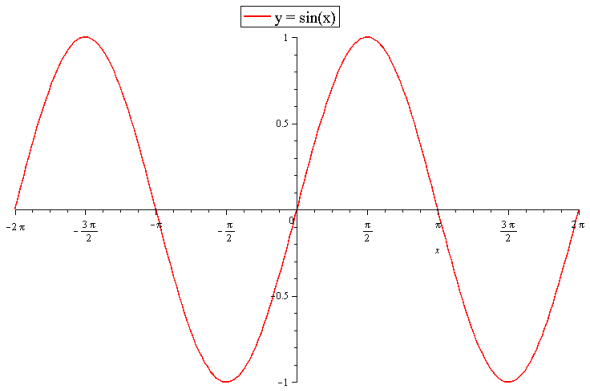


Figure 3: $y = sinx$

And when the sine factor of the functional equation evaluates to zero, this causes the entire function to evaluate to zero, due to the fact that anything

multiplied by zero, is zero. This suggests that when s is an even integer, the zeta function evaluates to zero. But then comes the question, why does the zeta function only have zeros at the negative even integers, but not the positive ones? Well, this is because of the gamma function (Γ). The factor $\Gamma(1-s)$ in the functional equation has simple poles at positive integers, or in other words, when $s \in \mathbb{N}, \Gamma(1-s)$ is undefined. So for $s \in \mathbb{N}$, the sine function would evaluate to 0, but the gamma function would evaluate to an undefined value. These values cancel each other out, which means that the equation does not necessarily evaluate to 0 if $s \in \mathbb{N}$. This is why the trivial zeros are only found at negative even integers.

However, although Riemann's Functional Equation has significant importance; it is long, complex, and would be hard to compute; mainly due to the fact that it is recursive, with the $\zeta(1-s)$ factor. Luckily, there are other equations that we can use to compute the Zeta function more efficiently. An example of this is by using the Dirichlet eta function, denoted by $\eta(n)$, such that:

$$\eta(s) = \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{n^s}$$

This function is also known as the alternating zeta function. It has the special property that:

$$\eta(s) = (1 - 2^{1-s}) \cdot \zeta(s)$$

We can now do arithmetic and rearranging of this function so that we can use it to compute $\zeta(s)$.

So rearranging for $\zeta(s)$:

$$\zeta(s) = \frac{1}{1 - 2^{1-s}} \cdot \eta(s)$$

Now, if we let $\phi_n = log(n)$, where $log(n)$ is the natural logarithm of n, then:

$$\begin{aligned} n^{it} &= e^{log\left(n^{it}\right)} \\ &= e^{it\phi_n} \\ &= cos(t \cdot \phi_n) + isin(t \cdot \phi_n) \end{aligned}$$

Then if we express $\zeta(s)$ using $\eta(s)$ in its summation form, and split the variable s into $\sigma + it$, we derive that:

$$\zeta(\sigma + it) = \frac{2^{it}}{2^{it} - 2^{1-\sigma}} \cdot \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{n^{\sigma}} \cdot [cos(t\phi_n) - isin(t\phi_n)]$$

Now, this equation is very long and complex, but it seems to be relatively simple to compute. However, due to the fact that the Dirichlet eta function is only defined for values where the real part of sis greater than one, it means that this form of the zeta function is only defined when $\Re(s) > 0$.

Similarly, we can manipulate the earlier function to find it in another form that will be easy to compute as part of a program.

If we take the equation from earlier that:

$$\zeta(s) = \frac{1}{1 - 2^{1-s}} \cdot \eta(s)$$

And substitute in the eta function where

$$\eta(s) = \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{n^s}$$

Then we can say that

$$\zeta(s) = \frac{1}{1 - 2^{1-s}} \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{n^s}$$

Then, by performing analytic continuation by using Hankel Functions (essentially very complicated mathematics that is beyond the scope of this project), it can be derived that:

$$\zeta(s) = \frac{1}{1 - 2^{1-s}} \sum_{n=0}^{\infty} \frac{1}{2^{n+1}} \sum_{k=0}^{n} (-1)^k \binom{n}{k} (k+1)^{-s}$$

Where $\binom{n}{k}$ is a binomial coefficient where:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Just like the previous function we derived, this does look very complicated, however for a computer, it is relatively simple to compute.

1.2.5 The Riemann Hypothesis and Prime Numbers

The Riemann Hypothesis is deeply connected to the prime numbers. If the Riemann Hypothesis was proven to be true, then the effect this would have on mathematics would be monumental.

Leonhard Euler derived a very nice equation from the Riemann Zeta Function, that involves the prime numbers. This function is known as Euler's Product Formula.

We already know that the zeta function can be defined by:

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

Where:

$$\zeta(s) = \frac{1}{1^s} + \frac{1}{2^s} + \frac{1}{3^s} + \frac{1}{4^s} + \frac{1}{5^s} + \dots \tag{1}$$

Now if we multiply both sides by the second term: $\frac{1}{2^s}$

$$\frac{1}{2^s} \cdot \zeta(s) = \frac{1}{2^s} + \frac{1}{4^s} + \frac{1}{6^s} + \frac{1}{8^s} + \frac{1}{10^s} + \dots \tag{2}$$

Subtracting equation 2 from equation 1 removes all elements with a factor of 2:

$$\left(1 - \frac{1}{2^s}\right) \cdot \zeta(s) = 1 + \frac{1}{3^s} + \frac{1}{5^s} + \frac{1}{7^s} + \frac{1}{9^s} + \dots \tag{3}$$

Then multiplying again by the second term, $\frac{1}{3^s}$

$$\frac{1}{3^s}\left(1 - \frac{1}{2^s}\right)\zeta(s) = \frac{1}{3^s} + \frac{1}{9^s} + \frac{1}{15^s} + \frac{1}{21^s} + \frac{1}{27^s} + \dots \tag{4}$$

Subtracting equation 4 from equation 3 removes all elements with a factor of 3 as well as those with a factor of 2 from the right-hand side.

$$\left(1 - \frac{1}{3^s}\right)\left(1 - \frac{1}{2^s}\right)\zeta(s) = \frac{1}{5^s} + \frac{1}{7^s} + \frac{1}{11^s} + \frac{1}{13^s} + \frac{1}{17^s} + \dots$$

We can see that the right-hand side of the equation is being sieved. If we repeat this process infinitely many times by multiplying by $\frac{1}{p^s}$ where p is a prime number, and then subtracting. We end up with:

$$\dots\left(1 - \frac{1}{11^s}\right)\left(1 - \frac{1}{7^s}\right)\left(1 - \frac{1}{5^s}\right)\left(1 - \frac{1}{3^s}\right)\left(1 - \frac{1}{2^s}\right)\zeta(s) = 1$$

Then rearranging for $\zeta(s)$:

$$\zeta(s) = \frac{1}{\left(1 - \frac{1}{2^s}\right)\left(1 - \frac{1}{3^s}\right)\left(1 - \frac{1}{5^s}\right)\left(1 - \frac{1}{7^s}\right)\left(1 - \frac{1}{11^s}\right)\dots}$$

We can then write this as an infinite product over all primes, p:

$$\zeta(s) = \prod_{p,prime} \frac{1}{1 - p^{-s}}$$

This function is only defined when $\Re(s) > 0$, due to the fact that negative numbers can not be prime. Furthermore, it would be hard to compute, due to the fact that you would be required to calculate the prime numbers first.

As well as Euler's Product Formula, there are many other functions that involve the prime numbers that are all related to the zeta function.

The prime number theorem was a theorem thought of by Carl Friedrich Gauss near the end of the 18th century. This theorem describes the distribution of the prime numbers. It formalises the intuitive idea that as numbers get larger, the prime numbers are less common, by precisely quantifying the rate at which this occurs. One way this theorem was modelled was through the prime counting function (denoted $\pi(N)$). Where $\pi(N)$ gives the number of primes that are less than or equal to N. Given this we can say that as $N \to \infty$ then $\frac{\pi(N)}{log(N)} \to 1$, where $log(N)$ is the natural logarithm of N This, therefore, means that:

$$\pi(N) \sim \frac{N}{log(N)}$$

This means we can approximate the numbers of primes less than or equal to N, by calculating $\frac{N}{log(N)}$ For example, if we wanted to approximate the number of primes less than 100: Then our approximation would be:

$$\frac{100}{log(100)} = 22$$

But the true value:

$$\pi(100) = 25$$

So there is some error in our approximation. However, the larger the number N, the better our approximation for $\pi(N)$. For example:

$$\pi(1,000,000) = 50,847,534$$

and

$$\frac{1,000,000}{log(1,000,000)} = 48,254,942$$

where proportionally, the error is a lot s maller

However, Peter Dirichlet and Carl Friedrich Gauss came up this a much better approximation for $\pi(N)$. They said that:

$$\pi(N) \sim Li(N)$$

Where $Li(N)$ is the logarithmic integral of N such that:

$$Li(N) = \int_0^N \frac{1}{log(t)} \mathrm{d}t$$

Where $\log(t)$ is the natural logarithm of t. Now if we wanted to use the logarithmic integral function to approximate $\pi(1,000,000)$, we get that:

$$Li(1,000,000) = 50,849,234$$

Which is only 1700 away from the true value.

As well as the prime counting function, we also have a similar function called the prime power function (denoted by $\Pi(N)$). In the prime counting function, you would get 1 point per prime number (less than or equal to N). But in the prime power function, you get 1 point per prime $+\frac{1}{2}$ point per prime squared $+\frac{1}{3}$ point per prime cubed and so on. As an equation this would be:

$$\begin{aligned} \Pi(N) &= \pi(N) + \frac{1}{2}\pi(N^{\frac{1}{2}}) + \frac{1}{3}\pi(N^{\frac{1}{3}}) + \dots \\ &= \sum_{r=1}^{\lfloor log_2 N \rfloor} \pi(N^{\frac{1}{r}}) \end{aligned}$$

But how is this related to the Riemann Hypothesis? It turns out that the prime number theorem was proved by using the Riemann Zeta Function.

Riemann came up with a function for the Zeta Function where he described it in terms of the non-trivial zeros. He stated:

$$\zeta(s) = \left(\frac{\pi^{\frac{s}{2}}}{2\,(s-1)\,\Gamma\left(\frac{s}{2}-1\right)} \right) \cdot \prod_{\rho} \left(1 - \frac{1}{\rho}\right)$$

Where ρ are the non-trivial zeros of the Riemann Zeta Function.

Then through complex analysis and advanced calculus, it was derived that:

$$\Pi(x) - Li(x) = \sum_{\rho} Li(N^{\rho}) - log(2) + \int_N^{\infty} \frac{1}{t(t^2-1)log(t)} \mathrm{d}t$$

Which tells us the difference or 'error term' between the prime counting function, and the prime power function. The main and most influential part of this error term is the sum of the logarithmic integrals of N^{ρ}.

Because ρ are the non-trivial zeros, the non-trivial zeros of the Riemann Zeta function are really controlling the size of this error term. If the Riemann Hypothesis were true and all of the non-trivial zeros lie on the critical strip, then this error term will be as small as possible.

Mathematicians were finally able to prove the prime number theorem by using the Riemann zeta function, by showing that the Riemann zeta function has no zeros on the line $\Re(s) = 1$. However, this error term is controlled by the non-trivial zeros of the Riemann zeta function, and it is still unproven whether all of the non-trivial zeros lie on the critical strip. But if the Riemann Hypothesis were true, then this error term is bounded by:

$$|\pi(N) - Li(N)| \leq c\sqrt{N} \cdot log(N)$$

Where c is some constant

This means that there would also be a similar bound between consecutive prime numbers, such that:

$$p_{n+1} - p_n \leq c\sqrt{P_n} \cdot log(p_n)$$

There are also many other consequences if the Riemann Hypothesis was proved to be true. For example, the Weak Goldbach Conjecture would also be proven true, stating that 'All odd integers greater than 5 are the sum of three primes'. Furthermore, if the Riemann Hypothesis was true then there will always be a prime number between consecutive cubes, such that:

$$x^3 < p < (x+1)^3$$

This would mean that you could define a constant θ such that $\lfloor \theta^{3^n} \rfloor$is prime for all n. This is called a Mills' Constant.

All of these theorems and conjectures highlight the importance of the Riemann hypothesis, and that if it was to be proven, it would lead to several major breakthroughs, not only in mathematics but quantum physics and computer science. The fact that the non-trivial zeros of the Riemann zeta function have such an importance in countless other fields just shows how influential the Riemann Hypothesis is.

1.2.6 Visualisations of the Riemann Hypothesis

There are numerous ways of visualising functions on graphs and plots. Throughout this next section, I will explore the different types of data visualisation techniques, as well as their positives and negatives.

The most common form of visualisation for the Riemann Hypothesis is showing how the output of the Riemann zeta function varies as you move along the critical strip. It plots the values of $\zeta(s)$ for $s = \frac{1}{2} + it$, where it is some imaginary number.

This graph shows us how the Zeta function varies along the critical strip and allows for us to identify the non-trivial zeros of the function.

Whereas most graphs show the input to the function along a horizontal axis and the output along a vertical axis, we cannot do this with the plot of the Riemann Zeta function because we are required to use both axes to represent complex numbers on the complex plane.

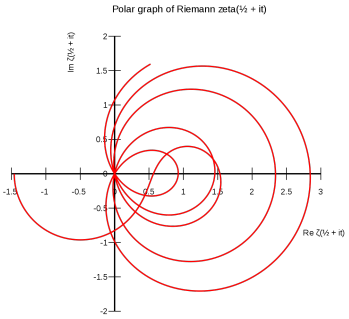


Figure 4:
Polar Graph of the Riemann Zeta Function

Some positives about this graph are that it is simple and easy to understand. However, it would require an input value for t, which is not shown on the graph, for the data to have any practical use.

However, we can expand on this graph and use it to help us to create a dynamic graph. A dynamic graph is one that changes with time. To plot a dynamic graph of the Riemann Zeta function, we could plot values of $\zeta(s)$ along the critical strip so that $\zeta(s) = \frac{1}{2} + it$. Where t would be our independent variable.

However, we would be changing t over time, where we would need a relationship between the time of the graph and t, which could be in the form of an equation.

If for example, we had it so that for every second, the value of t increases by 1, this would give us a dynamic graph that would plot the points of $\zeta(\frac{1}{2} + it)$ while the user is watching it. The graph would then be changing every second.

If we made it so that for every 0.1 seconds then t increased by 0.1, this would give us a much smoother plot and would be much more appealing to look at.

Although the graph at any given point in time would look the same as the previous graph, the fact that this graph is changing for the user to see makes it significantly more visually appealing. Furthermore, by changing the value of t over time, it allows for the user to get a much deeper understanding of how the

value of t affects the values of the Riemann Zeta Function.

Another way of representing functions that involve complex numbers is through domain colouring. This technique for visualising complex functions assigns different colours to each point on the complex plane. Not only does domain colouring look very appealing and intriguing it is also very useful to be able to plot functions that require four dimensions.

To graph a function with real inputs and outputs, two dimensions are required: one for the input and one for the output. But complex numbers are made up of two variables (the real and imaginary parts) and therefore two dimensions.

This means that in order to plot both the input and output of a complex function it requires four dimensions. The simplest way of doing this is by using the method of domain colouring.

In order to represent a function by using domain colouring, it is necessary to determine how the colours of the graph will be chosen.

There are multiple ways of doing this, but one of the most common ways is by taking the output of the function and using this to determine the hue and saturation of the colour.

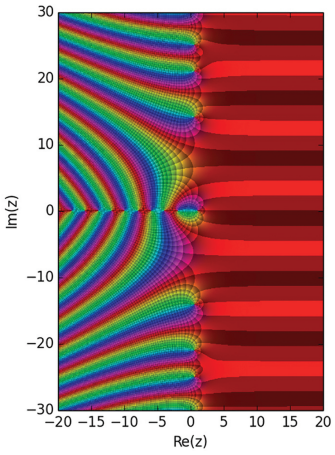


Figure 5: The Riemann Zeta Function Plotted Using Domain Colouring

For example, if we had an argand diagram for the complex number $x + iy$:

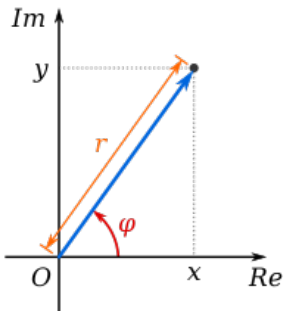


Figure 6: An argand diagram for the complex number $x + iy$

Then we could use the value of ϕ to determine the hue of the colour, and r to determine the brightness of the colour. If we did this for all outputs of a complex function, and then plotted them on a graph, then we would have a domain coloured graph.

Domain coloured graphs have multiple advantages: they look very interesting; they allow complex functions to be plotted using four dimensions; and they are useful to see the general trend of a complex function. However, these graphs are not ideal if you want to read off specific points.

Another method used to illustrate this data is through an interactive graph. An example of this could be where the user selects a complex number on the complex plane. This number is then input into the Riemann Zeta function and an animation is then displayed to show how that number gets translated by the Riemann Zeta Function. The graph would then display another data point that shows the output from the Riemann Zeta Function.

This type of graph is able to show the user how the Riemann Zeta function works and could even give insight into where the zeta zeros could be. Moreover, the interactive feature makes the graph a lot more engaging for the user.

We could also use interactive graphs to show how computers can approximate values for the Riemann Zeta function. Computers are unable to calculate the exact value of the output from the Riemann Zeta function, the reason of which I discuss later on, but it simply comes down to the fact that computers are unable to calculate the infinite series (where an infinite amount of numbers are added together) that are present in the functions. To get around this, you can compute these series to a high number that is not quite infinity, and still get a very accurate approximation.

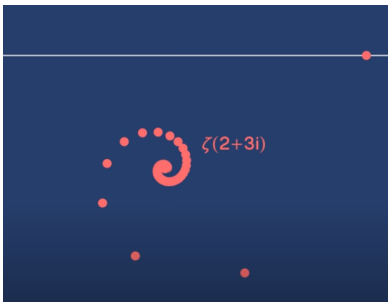


Figure 7: The approximation of $\zeta(2+3i)$

As each term in the series gets calculated, the approximation gets closer and closer to the true value. We can use this fact and display it on a graph. If a user selects a point on the complex plane, we can display the approximation for $\zeta(s)$ for 1 term, 2 terms ... all the way up until n terms (where n is a high enough number so that the approximation is extremely close to the true value). The resulting graph will show a spiral of data points that converge to one point; that is the output of the zeta function. This graph allows the user to understand how the infinite series works in the zeta function, as well as how computers are able to compute the zeta function.

1.2.7 Data Storage

As part of this program, I aim to be able to store the values of the $\zeta(s)$ for specific values of s. There are multiple ways to store data sets such as different file types and databases. I will be exploring these data storage methods and finding which method is best suited for my program.

The simplest way of storing data is by storing it in a text (.txt) file. These files store plain text that can be read from and written to. There is limited functionality when it comes to storing data in text files; as you can only read from and write to them. This could be used to store relatively simple and small

amounts of data, but to store anything larger and more complex, text files won't be the best choice.

Another way of storing data is by using comma-separated values (CSV) files. These files consist of numerous records and fields, where each line is a record and the fields are separated by delimiters - often commas.

```
~ ❯ cat example-csv-file.csv
Name,ID,Attendance
Alice,1,Present
Bob,2,Present
Charlie,3,Absent
Dave,4,Present
~ ❯
```

Figure 8: An example CSV file

This type of storage for data has a lot more functionality than using plain text files. The use of field headers allows entire fields to be read or written to with ease and data can easily be sorted, grouped, and processed. However, some downsides of using the CSV data format are that only basic data can be stored. Complex tables and configurations cannot be used with CSV files. Furthermore, there is no distinction between text and numerical values which can make processing the data difficult.

Undoubtedly, one of the most efficient ways of storing large data sets is by using databases. More specifically, by using Structured Query Language (SQL).

SQL is a declarative programming language, making it very efficient to use. Databases allow for large amounts of data to be written and read very quickly.

Moreover, processing this data can be very simple with packages such as SQLite that allows SQL databases to be accessed with Python. SQL databases are efficient and can store large amounts of data, making them ideal for a project like this.

1.2.8 Programming Languages

There are multiple programming languages that I could - relatively easily - use for this project, but it is important to choose a language that is suited for the task at hand. Every language has its advantages and disadvantages. Throughout this section I will evaluate various languages and whether or not they would be suitable for this project. I will then give my overall verdict on which language I wish to use.

C++ is a fairly low-level compiled programming language. Its use of classes and memory manipulation allows for an extensive range of functionality. Furthermore, with countless libraries and support, there is almost no limit to what can be done in C++. This is one of the main benefits of this language, alongside the fact that it can run incredibly fast compared to some other interpreted languages, and that it can run on almost any machine. However, the actual coding in C++ is long and tedious. Especially for a project like mine where memory management is not too much of an issue, it seems unnecessary to have to code in C++.

Javascript is also another programming language that could be used for this project. Unlike C++, Javascript is used for more high-level projects, predominantly for web applications. Making a web app has many advantages, such as high accessibility for multiple users. However, this would require an internet connection to run, and the speed of the program would be significantly reduced compared to that of a compiled application,

Python 3 is a high-level interpreted programming language, with a wide range of libraries and packages to support it. The main advantage of Python is that the code has high readability. This allows for it to be understood by people, even if they are not familiar with the language. Although Python is an interpreted language, there are ways to compile the code into an executable file that can be run on machines that do not have the interpreter installed. As well as this, Python has wide support for user interfaces and the plotting of graphs.

Overall, I think that Python 3 will be the programming language that is most suited for my project. A web application is not what my users are looking for, so javascript would be impractical to use. The high readability and high-level features are what makes Python the optimal programming language for this project, over Javascript and C++.

1.2.9 Product Research

Throughout this section, I will be taking a look at other programs that have similar features to mine. I will talk about the advantages and disadvantages of each program and how I can incorporate some of their features into my project.

One of the most major programs I will look at is the web application WolframAlpha. This is a mathematical computation site, where users can input a maths query into a search bar. The answer to the query is then returned, along with information about the solution.

In this example, I input zeta(10+5i). The program then outputs a decimal approximation of the answer to my query, as well as a diagram showing the output on the complex plane. Further down the page, the program lists alternative ways that I could've written my input.

This application works very well for calculating specific values of the zeta function, but it does little to show the mathematics of what's happening to calculate the answer.

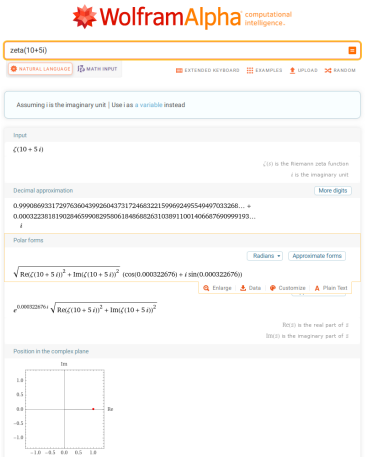


Figure 9: Wolfram Alpha

Another web application by Wolfram Research is the Riemann Zeta Function Page on Wolfram Mathworld.

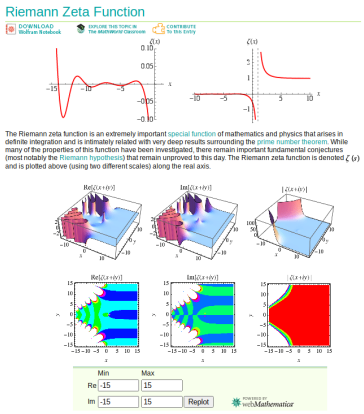


Figure 10: Wolfram Mathworld

This site explains the Riemann Hypothesis in great detail, giving the user a very thorough understanding of the subject. There are also multiple three-dimensional and domain coloured graphs that plot the Riemann Zeta Function. The user is able to change the scale of the plots to see how the zeta function changes, depending on the input. The web page contains a variety of different graphs and lots of maths along with explanations as to what is going on.

However, the maths on this page is very complex and it would be unlikely that people would understand this page if they didn't have some significant background knowledge into the Riemann Zeta function, or advanced maths in general. The page is also very linear starting from top to bottom. Overall, I like the way that the graphs are presented and explained here, however, the large quantity of maths displayed can seem confusing and unrelated.

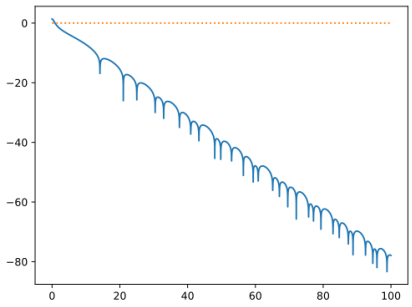


Figure 11: Verifying the Riemann Hypothesis by Aaron Meurer

The next program that I will look at is Verifying The Riemann hypothesis by Aaron Meurer.

He creates a program that finds the non-trivial zeros of the zeta function and displays this data on some very simple and understandable graphs. I like this program a lot because it contains a lot of relevant information.

Some positives of this program are that the graphs in it are presented and explained well so that anyone could understand what is going on. The only drawback is that this isn't a very extensive program. Only 2 graphs are calculated and there isn't a user interface.

Overall this program contains some appealing and useful information and graphs but could be more refined.

1.3 Project Objectives

My overall aim of this project is to be able to visualise and examine the Riemann Hypothesis, to make the complicated maths more accessible and to test whether the Hypothesis actually is true. I will be doing this by completing the following key objectives.

1. The program will be interactive and engaging for the user
   (a) There will be various questions throughout the program that the user can answer
   (b) The user will be able to make notes on any of the content in the program
   (c) The user will be able to choose what they do and where they go in the program, not forced along a single route
   (d) The user will be able to input their own values into various functions and graphs
2. The user will be able to save their progress through the program via a login system
   (a) The login system should allow the user to:
      i. Sign in to their account
      ii. Create a new account
      iii. Reset their password if they want to change it
      iv. Reset their password if they have forgotten it
   (b) Multiple users will be able to make separate accounts on the system
   (c) The user's data such as passwords must be saved securely
   (d) The user must be able to see how they have progressed throughout the program
3. The user will be given some background information about the Riemann Hypothesis such that almost anyone could feel comfortable using the program
   (a) The user should be able to learn the historical background of the Riemann Hypothesis
   (b) The user should be able to learn what imaginary and complex numbers are
   (c) The user should be able to learn what the Riemann Hypothesis states
   (d) The user should be able to learn about the practical applications of the Riemann Hypothesis

4. The program will allow the user to plot various graphs in order to develop their understanding of the Riemann Zeta function and allow them to learn about it
    (a) A polar graph of the Riemann Zeta Function
    (b) A graph of the prime counting function, and other related functions used to estimate it
    (c) A graph of the zeroes of the Riemann Zeta Function
    (d) A visualisation of the convergence of the infinite series in the Riemann Zeta Function
5. The user will be able to calculate specific values of the Riemann Zeta Function
    (a) A single calculator, where the user inputs a complex input and receives an output
    (b) A table calculator, where the user can calculate the zeta function for a range of input values
    (c) A leaderboard showing how many values of the zeta function, each user has computed
    (d) The program will allow the user to store these value(s) to a database and to a csv file
6. The user will be able to calculate the zeroes of the Riemann Zeta Function
    (a) The user will input how many zeroes they want to calculate
    (b) The program will calculate these zeroes
    (c) The zeroes will be displayed to the user in a table
7. The user will be able to store the data that they have collected
    (a) The program will include a database of multiple values and zeroes of the zeta function
        i. The user will be able to store values of the zeta function
        ii. The user will be able to store the non-trivial zeta zeros
        iii. The user program will retrieve data from the database and display its contents suitably to the user
8. The Program will have a graphical user interface

1.4 Third-Party Input

Throughout the following section, I will be discussing the consultation and guidance that I have received from various third parties. This section will cover the significance of third party input, who my third parties are, how and why I will be conducting my research, a transcript of the actual interviews themselves and my conclusions and takeaways from them.

1.4.1 The Significance of Third Party Input

Collecting third party input is vital in creating a successful project. It will allow me to assess the progress that I have made so far in my project, and check that I am heading in the right direction. Having input from various different people allows for people to view the project in different ways, each giving their own advice towards it. Third-party input and feedback is extremely important because it can help me develop and improve my project.

1.4.2 Who are my Third Parties?

There are benefits and drawbacks to each person you have as a third party. These depend on their relationship with me, their experience with creating projects like these and their willingness to help; just to name a few factors.

For my third parties, I am using friends who are computer science and further maths students. The benefits of this are that they have good availability, and I can talk to them regularly about the project. Furthermore, they are very supportive and encouraging. Because they are further maths and computer science students, they are also very knowledgeable about the subject area. However, the drawbacks of having friends as third parties are that they are potentially biased towards my project and could be unwilling to criticise it. It could be hard for them to tell me what is wrong, or what could be improved.

1.4.3 How and Why I will be Conducting My Research

I will be conducting my third party research by conducting interviews with my third parties. By creating a list of questions, I am able to ask specific questions that are useful to me. I have tailored the questions such that I can use the answers to help improve my project.

The interviews start with me asking questions to the third parties and taking answers from them. At the end of the interview, the third parties have an opportunity to ask me any questions that they have about the project.

1.4.4 Interview Transcript - Student 1

Question: What will an end-user be able to gain from using this project, how would it be helpful to them?

Answer: As a maths and further maths student, I want to be able to extend my knowledge of these subjects. This project looks like a good resource that will allow someone to understand the Riemann Hypothesis.

Question: What features could be implemented into the project such that the end-users can accomplish what was mentioned in the previous question?

Answer: It would be very beneficial to the user to have an explanation of the theory behind the Riemann Hypothesis at the beginning of the program. I'd like to do a couple of example steps, so I can try and understand the program before I start using it. You could have an example graph and pause the graphing animation and explain at each point what is happening. As well as this, I would like the program to be interactive so that I can really control what is happening.

Question: How accessible would this project be for someone with little or no knowledge of the subject area?

Answer: Due to the nature of the project, it is not extremely accessible. However, for someone with an interest in mathematics, they may be more willing to research around the topic to gain an understanding

Question: What could be done such that people who have less experience with the subject matter are still able to understand the program? How could it still be relevant to them?

Answer: For people with less experience, you could add some of the background information and story about the Riemann Hypothesis and the investigation. Mention why it's so important, and how it came to be. Not just the maths but the history behind it.

Question: Which parts of the project interest you; what would you like to research further?

Answer: The links to cryptography interest me. The fact it's such a modern-day feature of life, actually using the function that was theorised so long ago. I am interested in the practical applications of the Riemann Hypothesis and how it actually relates to me.

Question: How would someone be able to use this project to extend their knowledge of this subject area?

Answer: You would be able to use the interactive interface to manipulate and break down the theory of the hypothesis, in order to gain a deeper understanding of it

Question: Which features of the project do you like, and why?

Answer: Visualising is helpful for exploring the project. I like the graphs, the fact it has a visual element is very appealing. It is easier to appreciate the graphs than just looking at mathematical equations.

Question: Which features of the project do you think could be improved?

Answer: The accessibility to the project could be improved; in terms of the mathematics knowledge needed to understand it as well as how the program can be physically accessed and run. For example, as a downloaded piece of software or as a web application.

Question: Have you used any similar programs to this? What were they like?

Answer: A program that seems similar to this that I have used is the practical investigation software Focus eLearning. I used this to carry out practical investigations and consolidate learning. I liked that it was accessible over the internet and that it gave me explanations of the theory, with a visual exploration of the theory it was trying to explain, in the form of the interactive practical experiment. However, I felt that the user interface was difficult to navigate.

The question they asked me: How will you refine the access of the project so that everyone can use it?

My answer: I will include a detailed description and explanation of the Riemann Hypothesis as part of the program to make sure that whatever your mathematical ability, you will still be able to use the program.

1.4.5 Interview Transcript - Student 2

Question: What will an end-user be able to gain from using this project, how would it be helpful to them?

Answer: The Riemann Hypothesis itself is used in so many areas such as in cryptography and quantum physics, talking about the practical uses of the hypothesis would be helpful for the user to gain an insight into why the hypothesis is so important. I am interested in the practical usage of theory in computing and physics. Being able to understand the Riemann Hypothesis visually is also very helpful. Being able to graphically plot the many functions will give a deeper understanding of the project. Furthermore, being able to change the input values to the functions would help so that the user can expand their knowledge of the function

Question: What features could be implemented into the project such that the end-users can accomplish what was mentioned in the previous question?

Answer: It would be very good to make the graphing program interactive. This can be done by changing the function's input values, but also being able to traverse the plot by scrolling around and zooming in and out by changing the scale. It would also be useful to have multiple graphs displayed on the same set of axes to compare values.

Following along with this interactive idea, I think that you could questions for the user to answer about the hypothesis, to help solidify their understanding of it. It could be beneficial to add sections where the user and write down their observations and any conclusions that they have drawn from the investigation.

To implement this, you could create a login system where a user creates a username and password, so that they would be able to save their progress in the program.

Question: How accessible would this project be for someone with little or no knowledge of the subject area?

Answer: I believe this project will be relatively accessible because it mainly involves just displaying maths functions, where it takes an input and gives an output. It's relatively easy for someone to use it without knowing the details of how it works. That is, to be able to use the program, but maybe not fully understand it. But then to get a deeper understanding of the project, it takes more in-depth knowledge.

Question: What could be done such that people who have less experience with the subject matter are still able to understand the program? How could it still be relevant to them?

Answer: Start off by just exploring and explaining the hypothesis, how it works and how the Riemann zeta function works. If the user wants to understand it further, you can go through the maths of how it works; explaining the Riemann Hypothesis in detail. You could graph the individual mathematical functions in order to break down the problem for the user to understand. You can use 3D graphs and different ways of representing the data. This could make it easier for the user to understand.

Question: Which parts of the project interest you; what would you like to research further?

Answer: I am very interested in the cryptography application of the Riemann hypothesis. You could showcase as part of this project, a cryptography program that utilises the Riemann hypothesis in a practical way.

Question: How would someone be able to use this project to extend their knowledge of this subject area?

Answer: By utilising the multiple mathematical functions of the Riemann Hypothesis, the user could investigate how changing the input values changes the output. This will help the user learn about the individual functions and their use cases. When combining these functions together, it is important for you to be able to show how changing one piece of data can entirely change the output of a function, especially if the functions are all linked together. You should create the program such that the users could use it to investigate the domains and ranges of the functions.

Question: Which features of the project do you like, and why?

Answer: I like the idea of having an introduction where you explain the practical uses of the Riemann Hypothesis, and where you can give an explanation of how to use the Riemann Hypothesis and Riemann Zeta Function in other programs such as in the RSA cryptosystem. I like how the user would be able to change and manipulate the graphs to their liking and how easy it is to compare values of separate functions. It's also a good idea to have a conclusion/summary to help the user fully understand the maths.

Question: Which features of the project do you think could be improved?

Answer: It would be good for the user to be able to draw their own custom graph functions on the same axes as the functions in the program because currently, you would not be able to compare the functions in the program to the user's own custom functions. A drawback of the project is that the maths could be too specific and intricate for the user to gain a proper understanding of the project. It would require a lot of research to fully understand the project and to be able to make the most of it.

Question: Have you used any similar programs to this? What were they like?

Answer: A program like this that I have used is desmos. Desmos, a graphing software, allows you to plot many graphs on top of each other, you can easily change the axes as well as any input values. What's bad about desmos is that it doesn't allow you to plot in 3D, it would be nice to see this in your project. Another program similar to this is Graphing Calculator 3D. This allows you to do what desmos does, but it plots in 3D. However, its downsides are that It's not very user friendly and the menus are not easy to navigate, so it can be hard for new users to understand how the software works.

Questions asked to me:

Question: What are the necessary system requirements?

Answer: The necessary system requirements for this program are not very high. The functions are very memory and time efficient so the specifications of the systems that the program is run on should not be an issue.

Question: What would help the software be more useful to the user?

Answer: If that program gave an in-depth explanation of the Riemann Hypothesis, the user can gain a full understanding of what is going on. Furthermore, examples of the practical applications of the Riemann Hypothesis, make the program more relevant to the user and makes this program practical use instead of just being theory.

Question: With such complicated functions, how long would it take to compute them?

Answer: The functions I have written are very memory and time-efficient, it will

not take an extensive amount of time to compute the required mathematical functions.

Question: To what degree of accuracy will the functions be plotted?

Answer: The user will have an option where they can increase or decrease the accuracy of the functions. However, this will come with the disadvantage that it will take longer to compute.

1.4.6 Interview Conclusion

There are multiple improvement points from these interviews that I have used to help develop the plan for my project.

One of my main improvement points is with the design and layout of the project. The program will have 3 main sections: the introduction, the investigation, and the summary.

The introduction section will let the user be able to read and learn about the Riemann Hypothesis, so they can get an understanding of it, before using the main program. This introduction aims at making the program more accessible to people, even if they have less experience with maths. It will also allow people to explore the Riemann Hypothesis' links to cryptography and other practical applications so that they can explore it in more detail.

After the introduction will be the actual practical investigation, this is where the user can plot graphs and really investigate the Riemann Hypothesis. They will be able to plot multiple graphs on the same axes in order to be able to compare values. The user will also be able to easily change the input values to many of these graphs so that they can investigate how this changes the outputs.

The final part of the program will be a summary; showcasing the results from the investigation that they have conducted, and detailing some of the key mathematics behind the investigation.

Breaking down the project into these three sections will allow for the User Interface of the program to be simple, such that the user can access all of the relevant parts of the program easily.

Another part of the project spoken about in these interviews was a login system to help save a user's progress through the program. There would be multiple points in the program where the user can answer questions about the hypothesis to check understanding, or for the user to just write down any conclusions that they make about the investigation and the Riemann Hypothesis. Thus, creating a login system would allow a user to save their progress in the program, and any of their answers and data can be stored in a database.

1.5 Modelling the Problem

In this section, I will report any modelling of my project that will inform the design stage. I will be covering how to compute certain mathematical functions and also try some prototypes for what the project will look like.

1.5.1 Prototypes

Throughout this next section, I will be prototyping some mathematical functions in Python that will allow me to compute the Zeta Function. I will then test each of the functions to make sure that they work as expected.

The majority of functions that I intend to use involve an infinite sum, which is something that we are unable to compute because it would be impossible to add up an infinite amount of numbers. Therefore we must approximate the value of the function by calculating it for a certain number of terms. For example, when calculating the zeta function where:

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

We can approximate the value but calculating the sum to such a high number, that the error difference is not significant. If we want to approximate the zeta function, we could compute:

$$\zeta(s) = \sum_{n=1}^{1\times10^{6}} n^{-s}$$

Which would not give us the exact value, but one close enough that we can still use.

First of all, is a function that computes the zeta function by its definition. $\zeta(s) = \sum_{n=1}^{\infty} n^{-s}$. This function is defined for $\Re(s) > 1$ and could look as follows in Python 3:

```
# calculates zeta(s) for any complex number where Re(s) > 1
# where \zeta(s)=\sum _{n=0}^{\infty }\frac{1}{n^s}
def zeta(s: complex) -> complex:
    TERMS = 5 * 10**3 # the number of terms that we wish to compute.
    result = 0
    # computes an approximation for the infinite sum
    for n in range(1, TERMS+1):
        result += 1/(n**s)
    # output the final result
    return result
```

First, we define the function *zeta(s)*, where s is a complex variable. Then the constant *TERMS* is defined then defined to be a large integer. This number determines how many times we calculate $\frac{1}{n^s}$ and therefore determines how accurate our approximation will be. Then in the for loop, we calculate n^{-s} for *TERMS* many times and sum it to our result. We then return the complex variable res in order to output from the function.

Here we can test the function to show if and when it works as expected, and also how accurate it is to the true value.

No.	**Type**	**Input**	**Expected Output**	**Output**	**Result**
1	Invalid	-1	ValueError	12502500.0	Fail
Comment		This output is neither the output we were expecting nor the correct output of $\zeta(-1)$. I will need to fix this by validating the input to check that it is within the defined range of the function			
2	Invalid	0	ValueError	5000.0	Fail
Comment		This is the same situation as test number 1			
3	Boundary	1	ValueError	9.095	Fail
Comment		This is the same situation as the previous tests			
4	Valid	1.5	2.61	2.584	Fail
Comment		Although this gives an output that is close to the expected output, it is not accurate enough. I need to increase the number of terms in order to get a more accurate output			
5	Valid	5 $+10i$	1.025 $-0.015i$	1.025 $-0.015i$	Pass
Comment		This test gives a value that is extremely close to the true value. It see,s that out approximation becomes more accurate as the input value increases.			

Based on the testing, I have made some adjustments to the function as seen here:

```
# calculates zeta(s) for any complex number where Re(s) > 1
# where \zeta(s)=\sum _{n=0}^{\infty }\frac{1}{n^s}
def zeta(s: complex) -> complex:
    # validate that Re(s) > 1
    if not s.real > 1:
        raise ValueError('Real part of input must be greater than 1')
    TERMS = 1 * 10**6 # the number of terms that we wish to compute.
    result = 0
    # computes an approximation for the infinite sum
    for n in range(1, TERMS+1):
        result += 1/(n**s)
    # output the final result
    return result
```

The first change I made was validating the input. Now if the real part of the input is less than 1, then a *ValueError* is raised. Secondly, I have increased the number of *TERMS* that are computed in order to make the program more accurate. Although this does make the program take more time, the time difference is hardly noticeable.

Now here are the testing results for the improved function.

No.	**Type**	**Input**	**Expected Output**	**Output**	**Result**
1	Invalid	-1	ValueError	ValueError	Pass
Comment		This works as expected; the error is raised			
2	Invalid	0	ValueError	ValueError	Pass
Comment		This works as expected; the error is raised			
3	Boundary	1	ValueError	ValueError	Pass
Comment		This works as expected; the error is raised			
4	Valid	1.5	2.61	2.61	Pass
Comment		This functions output is very close to the true value			
5	Valid	$5+10i$	$1.025-0.015i$	$1.025-0.015i$	Pass
Comment		This functions output is very close to the true value			

Overall, this function is able to compute values of $\zeta(s)$ where $\Re(s) > 1$. It throws an error at incorrect inputs (which is good) and calculates $\zeta(s)$ relatively accurately. However, this approximation could be made more accurate by increasing the number of *TERMS*, although this could slow the program down significantly. Using this function to calculate $\zeta(s)$ to any useful degree of accuracy would take too much time to compute. Furthermore, this function only allows us to calculate values where $\Re(s) > 1$.

2 Documented Design

2.1 Structure Diagrams

The program will be made up of 4 main parts; that is the Login System, the Introduction, the Investigation, and the Summary.

The login system will allow users to either sign into an account or create an account with a username and password. All user information will be encrypted using hashing algorithms for security reasons, and the data will be stored in a database. The purpose of the login system is to allow the users to save their progress throughout the program. There will be various opportunities throughout the program for the user to write down answers to questions, or to record any observations that they make. This data will be stored specifically to their username such that when they next log in, they can continue from where they left off.

The introduction will be a visual section of the project that will demonstrate to the user how the program will work, as well as give an overview of what the riemann hypothesis is, why it's important, and how it is used in science and technology. The aim of this section is to provide the user with sufficient information and knowledge such that they will be able to effectively be able to use this program. Because quite a lot of mathematical knowledge is required to be able to properly use this program to its full potential, this section will try and provide some basic information of the project area for the user to feel more comfortable using the program.

The investigation will be the main bulk of the program. It will allow the user to plot graphs, find zeta zeros using a variety of methods, and investigate how prime numbers are related to the zeta function. The aim of this section is to get the user to investigate the hypothesis, collecting their own data that they can draw conclusions from.

Finally, the summary will be the last section of the program. Its purpose is to allow the user to reflect on their results, and see what conclusions they are able to draw from them. The summary will be split into 4 sections; a quick recap of the theory from the introduction section; followed by an overview of the user's results from the investigation. There will then be sections on whether their data proves/ disproves the Riemann hypothesis, how reliable the information from this program will be, and the impact on maths and computer science if the hypothesis is proven to be true.

2.1.1 Program Overview Structure Diagram

The following structure diagram for the program gives an overview of the main sections in the project:

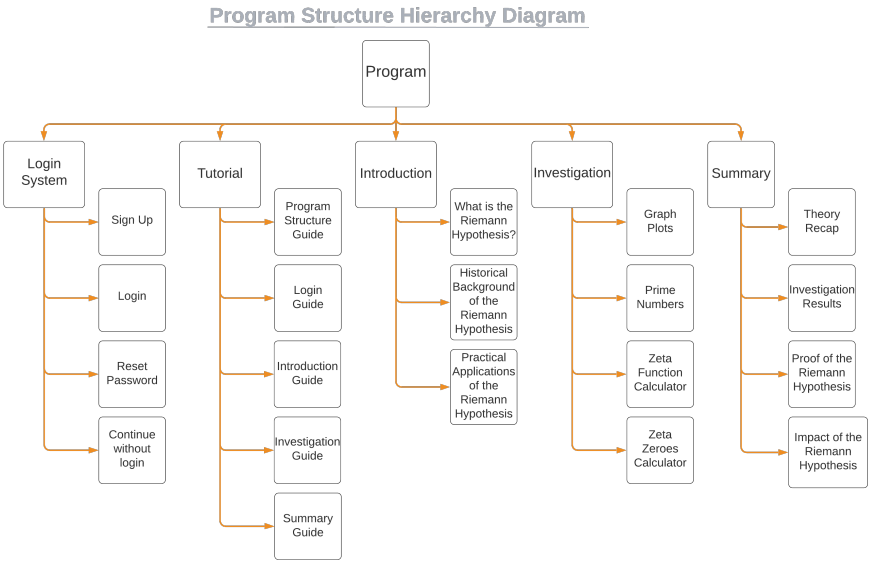


Figure 12: Structure Diagram for the program

As we can see from this diagram; the main program is split up into its 5 main sections: the Login System, the Tutorial, the Introduction, Investigation, and the Summary. Each of these main sections is then further broken down into its relevant subsections.

We can then break down each section into its subsections, which will in turn, be broken down into more sections.

2.1.2 Login System Structure Diagram

The login system's aim is to be able to allow a user to sign into an account, such that their progress with the program has been kept.

It will not be necessary for a user to login, however, if they do not log in, then their progress will not be saved.

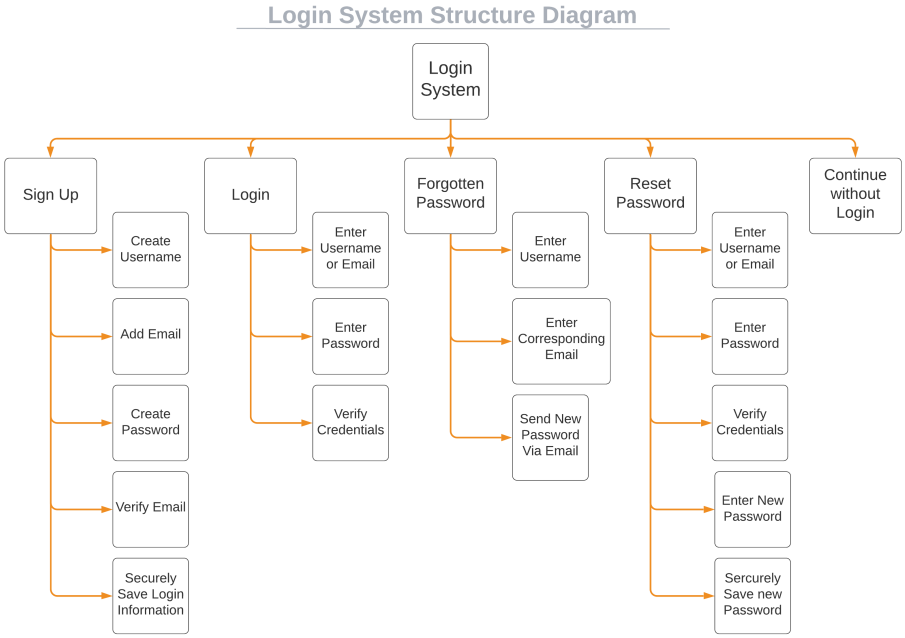


Figure 13: Login System Structure Diagram

There will be 5 sections to the login system that I create. The first of which allows the user to sign up and create an account. This is for first-time users of the programme, who want to be able to use it to its full functionality. The sign up process involves the user creating a unique username, adding a unique email to the account (used for when the user forgets their password), and creating a strong and complicated password. They will then be sent an email to the address that they entered, with a verification code. Once this code has been entered in the program, the user's login details will be encrypted and stored in a database. They will then advance to the next part of the program: the tutorial.

The login section will allow the user to sign in to an account that has already been created. They will be required to enter either their username or email, and their password. The program will check whether the username or email is valid (if it is in the database), and if the password associated to that account is the same as the one entered by the user. If the user gets the password wrong 3 times, then they will have to wait 1 minute before trying again. This is to stop

any attacks to try and guess user's passwords. Once a user has signed in, they will be taken to the tutorial section.

If a user has forgotten their password, then they will be able to reset it in the forgotten password section. The user will be required to enter either their username and email for that account. These credentials will need to be verified, and then if they are correct, a new password will be sent to the user by email.The user will then be sent straight to the Reset Password screen so that they can change the password to one that they will remember.

The reset password section will allow the user to change their password if they don't like the one they currently have, or if they have had to reset their password. They will have to enter their username and password, to confirm their identity. They will then enter a new strong and complicated password, and this will be the new password associated to their account.

The final section is the continue without login section. Without a login, the user will still be able to use all parts of the program, however, their data will not be saved.

2.1.3 Tutorial Structure Diagram

The tutorial is one of the key sections of the program. The tutorial will give the user a guide on how to use the program.

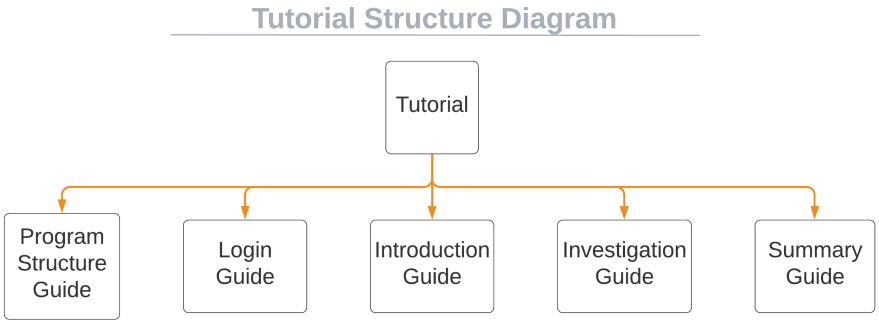


Figure 14: Tutorial Structure Diagram

The tutorial itself will be split up into 4 main parts; that is a guide on the structure of the program, and how to use each of the individual sections of the program.

Overall, the tutorial will be relatively simple, and will just give the user an understanding of how to use the program

The Program Structure Guide will inform the user of the layout of the project, and how each section is related to another. It will tell the user how to navigate pages, and how to use key features of the program that will require user input.

Next in the tutorial, will be the Login Guide. This will essentially just be a quick note to the user, about how to sign up and create a strong and complicated password, what to do if you forget your password, or want to change it; as well as explain why the user may or may not want to create an account, and how the program differs whether the user is signed in or not.

As for the Introduction Guide, it will show an example of what the introduction section will look like, and have labels to show how to navigate pages as well as what everything means.

Probably the most complex part of the tutorial section will be the Investigation section. This will inform the user of how to use, understand, and interpret the different graphs and plots. This section should give the user a simple understanding of how they can use the program to investigate the Riemann Hypothesis

The final section in the tutorial will be a Summary Guide, that is how to use the summary section of the program. This will be very similar to the Introduction Guide, but will have an emphasis on the user trying to draw conclusions from the data they have collected, instead of the just being able to understand what is going on.

The tutorial section will not have a lot of functionality, but its purpose is just to briefly describe to the user, how the program is intended to be used.

2.1.4 Introduction Structure Diagram

In the Introduction section of the program, the user will be able to develop their understanding of what the Riemann Hypothesis is and why its so important.

I will portray this information by splitting the Introduction into three smaller sections. You can see how this section will be split up in the Structure Diagram below:

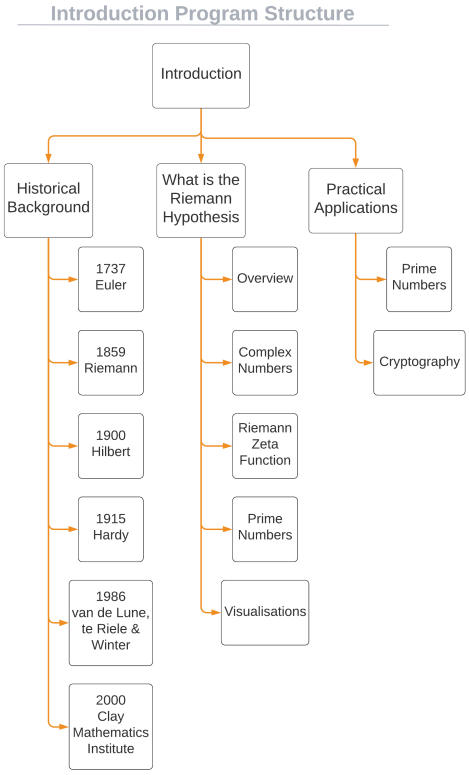


Figure 15: Introduction Structure Diagram

First of all is the historical background section. This shows how the Riemann Hypothesis has developed over time and showcases the context of the problem

Next is the section describing the Riemann Hypothesis. This section will include a lot of maths, but will also be written such that people will a limited mathematical understanding will still be able to understand what is being said.

The final part of the introduction is the Practical Applications section. This will inform the user why the Riemann Hypothesis affects us today and how it can actually be used.

2.1.5 Investigation Structure Diagram

The Investigation is the main part of the program and will allow the user to conduct their own research into the Riemann Hypothesis.

It will have 4 main section; The graphs, Prime Numbers, a zeta function calculator, and a zeta zeros calculator.

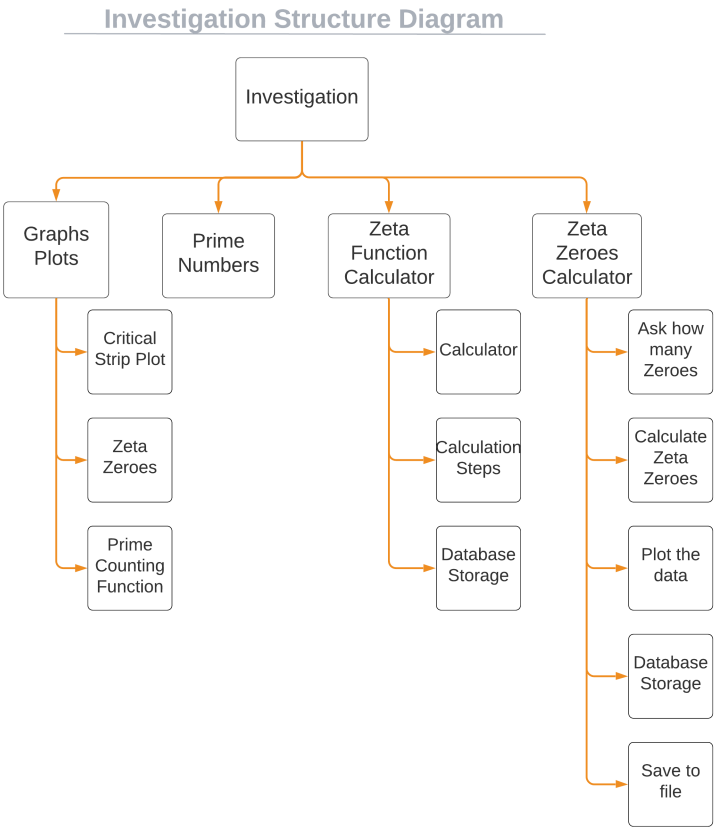


Figure 16: Investigation Structure Diagram

The first section will be the Graph Plot section. The aim of this section is to display to the user various graph plots that are related to the zeta function. This user will be able to change given inputs into the graph plots and be able to see the output visually.

The prime number section will show the user the correlation between the Riemann Zeta Function and the prime numbers. This section will showcase various graphs and mathematical functions for the user to explore.

Next is the zeta function calculator. This will be a relatively simple part of the investigation section. It will allow the user to input a number (or range of

numbers)into the calculator. It then calculates these values and will allow the user to store the data, either in a file or database.

Finally is the zeta zeroes calculator. This will calculate all of the zeta zeroes between two points given by the user, and with accuracy determined by the user. Similarly to the zeta function calculator, the user will also be able to store this data to a database or file, or even display the data in a graph plot.

2.1.6 Summary Structure Diagram

The final main section in the program will be the summary. It will be a recap of what the user should have learnt from using the program, and what to take away from it. It will be split up into a theory recap section, a results section, a conclusion and evaluation section and a section on the impact of the Riemann Hypothesis.

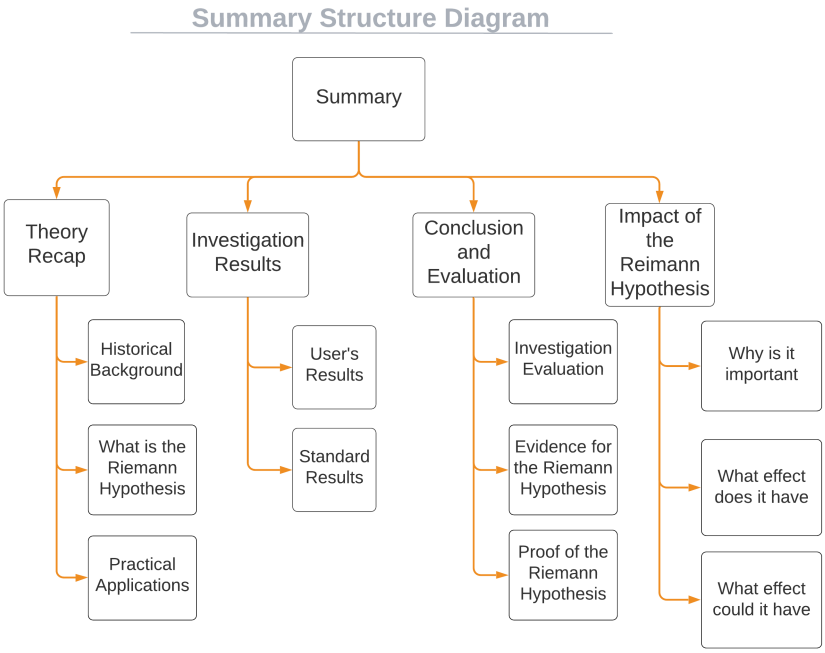


Figure 17: Summary Structure Diagram

The first part of the summary; the theory recap will essentially be a very cut down version of the introduction section. It will provide an overview of the problem such that the user can refresh their mind as to what the Riemann Hypothesis actually is.

As for the investigation results, this page aims to provide the results of the user's investigation. It will provide raw data as well as some visual aids such as

graphs. It will also compare the data collected by the user with the expected data.

Next is the conclusion and evaluation section. This section will detail to the user how accurate their results were to the true values, as well as any evidence and proof of the Riemann Hypothesis.

Finally, is the Impact of the Riemann Hypothesis. The aim of this section is to explain to the user why the Riemann Hypothesis is so important by explaining how it affects us.

Overall, I feel as if I have sufficiently broken down my project into a number of more manageable parts that will comes together neatly to form the whole system.

2.2 Object-Oriented Design

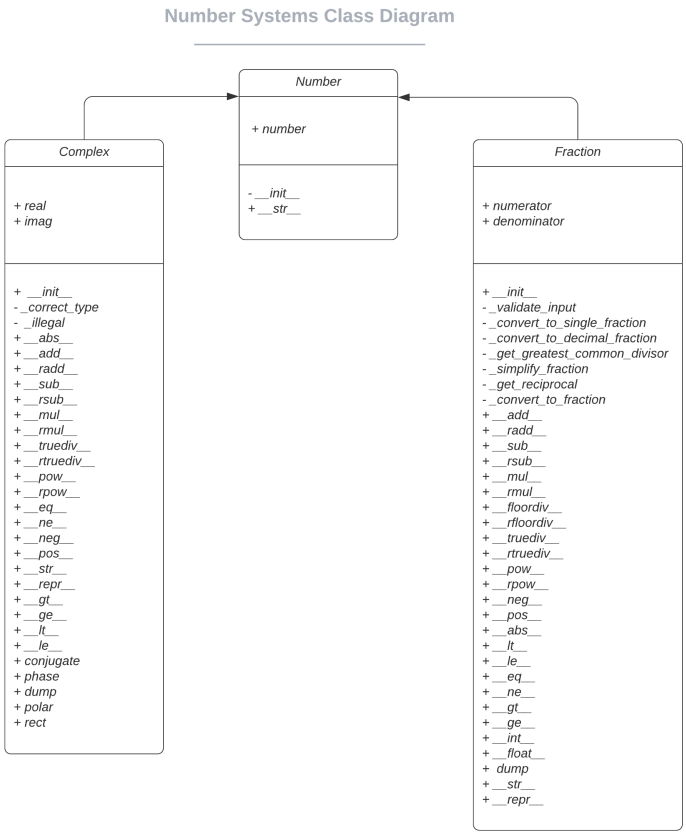


Figure 18: Number Systems Class Diagram

As part of this project I will need to be able to represent many types of numbers namely, fractions and complex numbers. The best way to do this is to use Classes and Object-Oriented Programming. I can create a data structure in python, such that they will be able to store all of the necessary data, and perform all the operations that I need.

All of the data types will inherit attributes and methods from a main *'Number'* Class. This number class will have basic attributes and methods, allowing a number to be defined and printed.

The complex number class is a lot more substantial than the number class. It will allow for inputs in Rect form, or Modulus Argument Form. The methods in this class will allow for the user to add, subtract, multiply, divide, and use exponentials with complex numbers. There will also be a set of private functions in the class in order to change between cartesian and polar form.

As for the fraction class, it will take inputs of a numerator and denominator, and will allow the user to use public functions in order to perform operations and calculations involving fractions.

These classes will be able to let me sufficiently perform precise calculations with numbers. The fraction class removes any mathematical floating point errors from calculations, and the complex class will allow me to utilise imaginary and complex numbers in Python.

Some points to note; I have tailored this class diagram and the following pseudocode for a Python programme. Conventionally, when calling a function prefix notation is used. For example, if we have a simple function as follows:

Algorithm 1 Prefix add function

```
function ADD(a, b)
    return a + b
end function
```

Then we can see that it takes two inputs, a and b, and will return their sum.

To call this function, we will use prefix, notation with the function name at the beginning, followed by the two parameters. Such as:

Algorithm 2 Calling the prefix add function

```
ADD(a, b)
```

However, in Python, we are able to create mathematical functions that allow for infix notation. This is a more natural way of calling maths functions. To allow for infix notation, we add two underscores to the beginning and end of the function name when we define it. So for some example python code:

```
# Python infix add function
def __add__(a, b):
    return a + b
```

We can run it by calling

```
# Python call infix add function
    a = 1
    b = 2
    c = a + b
```

Which is infix notation. Now in this example, it may seem almost obsolete to create a function that already mimics the behaviour of what we are trying to achieve, however using this notation we are able to overwrite the existing infix

notation in python, and even create new notation of any data types that we choose to create.

A simple example of this is with the fraction class. There is no inbuilt class for handling fractions in python, so we can create one and allow it to use infix notation.

If we have a fraction class that takes a numerator and denominator as an input, and have this function defined in the class, it would look as follows:

```
# Python __add__ as part of the Fraction Class
def __add__(self, other):
        """ self + other """
        other = self._convert_to_fraction(other, 'addition')
        resulting_numerator = self.numerator * other.denominator +
                              self.denominator * other.numerator
        resulting_denominator = self.denominator * other.denominator
        return (Fraction(resulting_numerator, resulting_denominator))
```

This function finds a common denominator for the fractions and then adds the numerators together.

To call this function to add $\frac{3}{4}$ and $\frac{1}{2}$, we would use infix notation as follows:

```
# Python calling __add__ for two instances of the class Fraction
Fraction(3, 4) + Fraction(1, 2)
```

Which would return the correct value of $\frac{5}{4}$.

As we can see from this example, we have created a python function that can be called using infix notation, rather than the standard postfix notation.

One other point to note is that python does not handle infix notation for user defined data types very well. If we want to compute the user defined *Fraction* + the Python Integer 3, this is okay. But due to the way python has been created, we would be unable to do Integer 3 + *Fraction*. For this we need to define another function called *__radd__*, short for right add. This allows us to still compute the infix addition, even when the user defined variable is on the right hand side of the addition sign. The *__radd__* function would use the already created *__add__* function, but would have its arguments swapped and the code would look as follows:

```
# Python infix radd function
def __radd__(self, other):
        """ other + self """
        return self.__add__(other)
```

This would swap the arguments around so that the *__add__* function is then

called with the parameters in the correct position.

2.3 Database Design

Throughout this next section, I will be explaining the structure of the database I will be using, creating data tables for each of the tables in the database, create designs for the key SQL queries that will be used, and I will create entity-relationship diagrams for the database.

2.3.1 Table Designs

In this subsection, I will be using data tables to represent the tables that will be present in my database.

Within my database, there will be 8 tables, which are:

1. Users Table
2. Questions Table
3. Answers Table
4. User Answer Table
5. Notes Table
6. Notes Responses Table
7. Zeta Table
8. User Zeta Table

In each data table, I will show the table name and the name of each of the fields in the table. For each of the fields, I will list whether it is a key of the table, it's datatype, any necessary validation using regular expressions, and some notes on the field.

With the regular expression used for validation, an item in the table will only be considered valid if the entire item itself is a match to the regex. For strings, I am limiting their size to 140 characters, such that the user will be unable to enter strings that are too long and potentially use up too much storage in the database.

Users Table

The purpose of the Users Table will be to store the login credentials for each user registered to the program. The table will store the username, email, and hashed password of each of the users. This is the table that will be referenced when a user creates an account, or goes to login to an existing account.

Table: Users				
Field	**Key**	**Data Type**	**Validation**	**Notes**
User_ID	Primary	Integer		
Username		String	\w{1,20}	Unique Username for each user
Email		String	.+@.+\..+	
Password		String		A hashed version of the user's password

Table 1: Data table for the Users Table

Questions Table

Another table in the database is the Questions Table. At points throughout the program, the user will be asked questions. These are questions that the user can either get right or wrong and they are used to check that the user is able to use the program in the right way.

The table will store, for each question, the question number the actual question itself, and the correct answer to the question.

Table: Questions				
Field	**Key**	**Data Type**	**Validation**	**Notes**
Queston_ID	Primary	Integer		
Queston_No		Integer	\d+	The number of the question
Question		String	.{1,140}	Questions that is asked to the user
Answer		String	.{1,140}	The correct answer to the question

Table 2: Data table for the Questions Table

Answers Table

Users will be expected to give answers to the questions, throughout the program. The answers of each user will be stored in the Answers table.

Every answer will have an associated Answer_ID, which will be used in the User Answer table to determine answer corresponds to which user. The answer field will be used to record the various different answers said by the users, and the values in Answer, will be in the form of a set, such that two members in the Answer field will not be the same. The Question_ID field is then used to match each answer up with the question that it was answering.

Table: Answers				
Field	**Key**	**Data Type**	**Validation**	**Notes**
Answer_ID	Primary	Integer		
Answer		String	.{1,140}	
Question_ID		Integer		

Table 3: Data table for the Answers Table

User Answer Table

The User Answer table will link the user's answers to them. Formatting the table this way allows for minimal data to be stored. If the answer to one of the questions is 2, and the vast majority of users answer 2, then the value 2 will only need to be stored once in the answers table, we can then use the User Answers table to link the User_ID of everyone who answered 2, to the Answer_ID of 2, rather than saving an answer over and over again.

This table will only have two fields, the User_ID, and the Answer_ID.

Table: User Answer				
Field	**Key**	**Data Type**	**Validation**	**Notes**
User_ID		String		
Answer_ID		String		

Table 4: Data table for the User Answer Table

Notes Table

Similarly to the questions and answers that occur in the program, the user will also be able to make notes and comments on their findings.

The difference between notes and questions is that there are no wrong or right answers (or responses as I name them) for the notes, whereas the answer to a question will either be right or wrong.

Table: Notes				
Field	**Key**	**Data Type**	**Validation**	**Notes**
Notes_ID	Primary	Integer		
Note_No		Integer	\d+	The number of the note question
Question		String	.{1,140}	Questions that is asked to the user

Table 5: Data table for the Questions Table

Responses Table

The responses table will act to the Notes table in a similar way to how the Answers table has a relationship with the Question Table.

Whoever, there will of course be differences in how the tables relate to each other, how they are used, and differences in the information stored in each of the tables.

Unlike with the Questions and Answers, the Responses to the Notes will most likely by completely unique from user to user. Therefore, creating a User Responses table, similar to the User Answer table will not be necessary.

In the table will be stored each response, the Response_ID, which will be a unique identifier for each response, the Note_ID, which describes which note the response is for, and the User_ID, showing which user said the response.

Table: Responses				
Field	**Key**	**Data Type**	**Validation**	**Notes**
Response_ID	Primary	Integer		
Note_ID		Integer		
Response		String	.{1,140}	The user's response to the posed question
User_ID		Integer		

Table 6: Data table for the Questions Table

Zeta Table

The zeta table will be used to store values of the zeta function. The aim of this table is to be able to store possibly hundreds of input, output pairs for the zeta function.

These input and output values will be complex numbers in the form $a + bi$, where a and b are two integers

The Zeta_ID Field, will be used in the User Zeta table, to determine which users have computed which values of the zeta function.

Table: Zeta				
Field	**Key**	**Data Type**	**Validation**	**Notes**
Zeta_ID	Primary	String		
Input		String	\d+\+\d+i	
Output		String	\d+\+\d+i	

Table 7: Data table for the Zeta Table

User Zeta Table

The last table in the database will be the User Zeta table. Its fields will be Zeta_ID and User_ID and will be used to match up users and input-output pairs that have been calculated for the zeta function.

This table will be used to calculated which users have computed which values of the riemann zeta function.

Table: User Zeta				
Field	**Key**	**Data Type**	**Validation**	**Notes**
Zeta_ID		String		
User_ID		String		

Table 8: Data table for the User Zeta Table

2.3.2 Third Normal Form

2.3.3 Database-Program Connections

2.3.4 Entity-Relationship Diagrams

2.4 Key Algorithms Design

Throughout this section, I will be modelling some of the key algorithms used throughout this project. For each algorithm, I will describe how it works using structured English, represent it by using a flowchart, and then write the algorithm in Pseudocode and Python.

2.4.1 Euclidean Algorithm

The Euclidean Algorithm is a method for computing the greatest common divisor of two integers. The greatest common divisor is the largest number that divides into both of the numbers without a remainder. This algorithm is recursive.

It works by, first, taking in two integers a and b. These are the numbers for which the greatest common divisor will be found. I will explain how this algorithm works through the use of an example.

Say we want to find the greatest common divisor of 45 and 10. We would first take the largest of the two numbers, and set that equal to some multiple of the smaller number, plus some remainder.

$$45 = 10 \cdot n + r$$

Where n is the multiple of the smaller number, and r is the remainder. We can then compute what the values of n and r are. For this example, $n = 4$ and $r = 5$.

$$45 = 10 \cdot 4 + 5$$

Say we then use this as a general equation

$$a = b \cdot n + r$$

The next step (and every other step) of the algorithm is to take the value of b and move it to the left hand side of the equation, replacing the value of a. We then do a similar thing by replacing the value of n with the value of r. This process is repeated until $r = 0$.

Completing this step would look as follows:

$$10 = 5 \cdot n + r$$

Then working out the values of n and r

$$10 = 5 \cdot 2 + 0$$

And now the repeating process stops because $r = 0$. We then output the value of b, and that is the answer for the greatest common divisor of the two numbers a and b. So for this example, the greatest common divisor of 45 and 10 is 5.

Now, lets see how this algorithm works on a larger example, where we want to work out the greatest common divisor of 300 and 245.

$$300 = 245 \cdot n + r$$

$$300 = 245 \cdot 1 + 55$$

$$245 = 55 \cdot n + r$$

$$245 = 55 \cdot 4 + 25$$

$$25 = 5 \cdot n + r$$

$$25 = 5 \cdot 5 + 0$$

Then our answer is the value at position b in $a = b \cdot n + r$, which is 5. So the greatest common divisor of 300 and 245 is 5.

Now, earlier I mentioned that we start by taking the larger of the two inputs, and place that on the left hand side of the equation, but really, the algorithm works no matter which input starts on the right of the equation. Say instead of calculating the greatest common divisor of 300 and 245, we want to calculate the greatest common divisor of 245 and 300. We would start by forming the equation:

$$245 = 300 \cdot n + r$$

And then working out the values of n and r:

$$245 = 300 \cdot 0 + 245$$

For any set of values where the smaller input is at position a, we will always be left with $n = 0$ and $r = a$.

Following through with the algorithm, the next step would be:

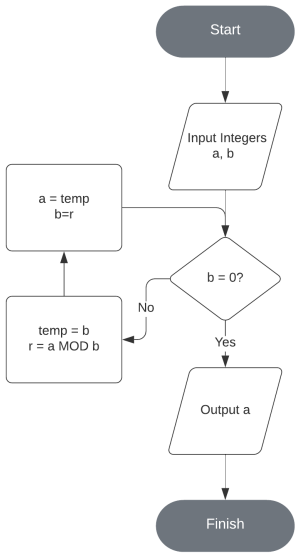
$$300 = 245 \cdot n + r$$

This is exactly how we started the previous example. This shows that it doesn't matter which way round the two input numbers go in the algorithm, it will always compute the same number.

In other words, the greatest common divisor of a and b, is the same as the greatest common divisor of b and a, which intuitively makes sense.

Now using flowcharts, we can write the algorithm as follows:

Figure 19: Euclidean Algorithm Flowchart



By representing the algorithm as a flowchart, we can see that it starts of by receiving two integers a and b as an input, and then keeps on finding the values of n and r from the examples, until the value of b is 0. Implementing this as code, this looping could be done using a standard `WHILE` loop, but I think that implementing this algorithm as a recursive function would be more suitable.

Implementing this algorithm in pseudocode, it would look as follows:

Algorithm 3 Euclidean Algorithm Pseudocode

```
function GREATEST_COMMON_DIVISOR(a, b)
    if b = 0 then
        return a
    else
        return GREATEST_COMMON_DIVISOR(b, a MOD b)   ▷ recursively call the function until b = 0
    end if
end function
```

We can see from this pseudocode, the recursive nature of this algorithm.

Then writing the Euclidean Algorithm in Python:

```
# Euclidean Algorithm
def greatest_common_divisor(a: int, b: int) -> int:
    if b == 0:
        return a
    else:
        # recursively call the function until b = 0
        return greatest_common_divisor(b, a % b)
```

This implementation of the Euclidean Algorithm in Python, is very efficient. Although it is recursive, and that can in some circumstances lead to a large amount of memory use - especially in Python - it would be relatively simple to unwind the stack at the end of the computation, because this function does not call any other functions and is relatively simple.

2.4.2 Riemann Zeta Function

One of the most important mathematical functions in this project is the Riemann Zeta Function. As mentioned in my analysis section, it can be defined as:

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

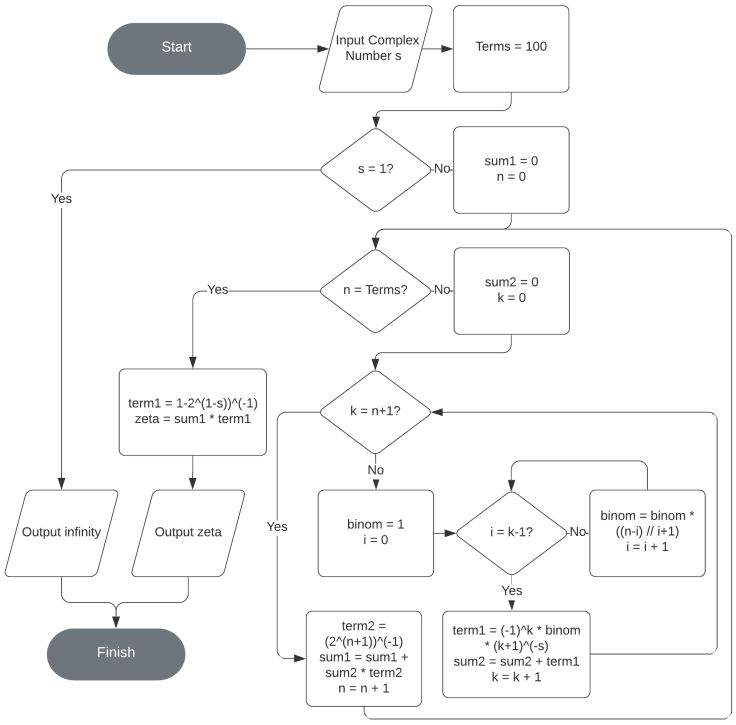
Which can be rewritten as:

$$\zeta(s) = \frac{1}{1 - 2^{1-s}} \sum_{n=0}^{\infty} \frac{1}{2^{n+1}} \sum_{k=0}^{n} (-1)^k \binom{n}{k} (k+1)^{-s}$$

This is just one of many of the ways that $\zeta(s)$ can be defined. The main advantages of this definition are that it is will allow inputs for any complex number s (apart from when $s = 1$ for which $\zeta(s)$ is undefined), and that it is fast to compute.

Using flowcharts, we can write this function as follows:

Figure 20: Zeta Function Flowchart



Unfortunately, with any computational implementation of the zeta function, it would be impossible to calculate an exact output for any given input. This is due to the fact that the zeta function, be definition is the sum of an infinite series.

It would be intractable to compute every single element in the series to be able to sum it. Because the infinite sum is convergent (except for $\zeta(1)$ which cannot be calculated), we can approximate the sum by calculating a very large number of terms in the sum, instead of an infinite amount. The number of terms we calculate in the sum is a trade-off between the time it takes to compute the function, and the accuracy of our answer.

This is what the variable `Terms`, in the flowchart, is used for. Its aim is to replace the ∞ in the infinite loop, such that we can still compute $\zeta(s)$ to a high degree of accuracy, with the function still being relatively time efficient.

In a sense, we are changing our definition of the zeta function, so the function that we are actually calculating is this:

$$\zeta(s) \approx \frac{1}{1-2^{1-s}} \sum_{n=0}^{\texttt{Terms}} \frac{1}{2^{n+1}} \sum_{k=0}^{n} (-1)^k \binom{n}{k} (k+1)^{-s}$$

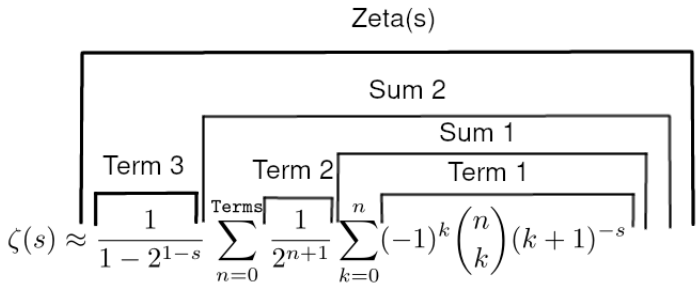
Which is a less accurate version of the zeta function, but is actually computable.

From the flowchart, we can see how the situation where $s = 1$ is handled. $\zeta(1)$ is undefined, so we would be unable to calculate this. Therefore, we have to check this special case, and if $s = 1$, then the function will `OUTPUT Infinity`, otherwise, it will just continue with the rest of the function.

The rest of the function, is essentially just creating two sequences and summing them. The flowchart represents this well, it allows us to visualise the looping nature of the sequences.

We can break down our formula for the approximation of $\zeta(s)$, which will make it clear how the flowchart is designed to work.

Figure 21: Decomposed Zeta Function



From this image, we can see that:

$$\zeta(s) = \texttt{Term3} \cdot \texttt{Sum2}$$

Where:

$$\texttt{Sum2} = \sum \texttt{Term2} \cdot \texttt{Sum1}$$

Where:

$$\texttt{Sum1} = \sum \texttt{Term1}$$

Then, all that has to be done is replace the values of `Term1`, `Term2`, and `Term3` with their respective values, and that is essentially what is being calculated.

By applying decomposition to the function, we can now easily see how the flowchart works, and now how to implement this function by using pseudocode.

Algorithm 4 Zeta Function Pseudocode

function ZETA(s)
  $TERMS \leftarrow 100$
  if $s = 1$ **then**
    return ∞
  else
    for $n \leftarrow 0$ **to** $TERMS$ **do**
      $sum1 \leftarrow 0$
      for $k \leftarrow 0$ **to** $n + 1$ **do**
        $sum2 \leftarrow sum2 + (-1)^k \cdot \binom{n}{k} \cdot (k+1)^{-s}$
      end for
      $sum1 \leftarrow sum1 + sum2 \cdot \frac{1}{2^{n+1}}$
    end for
    $constant \leftarrow 1 - 2^{1-s}$
    return $\frac{sum1}{constant}$
  end if
end function

For sake of readability, I have omitted creating separate variables for `Term1`, `Term2`, and `Term3`. Instead, these can just be seen by the mathematical equations present in the Pseudocode.

I much prefer this representation of the function, as it easily allows you to see how the sums are implemented alongside the terms. The use of `for` loops makes it easy to see how we have a sum being calculated inside a sum.

With pseudocode, I have just been able to write the mathematical statements for readability sake, however, when implementing this as a programming language, it could look rather messy with long maths having to be written out. One area that would need addressing when converting this pseudocode to an actual language is with the binomial coefficient, that is $\binom{n}{k}$. I have previously mentioned this in my analysis, however:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Where:

$$\begin{aligned} x! &= \prod_{a=1}^{x} a \\ &= x \times (x-1) \times (x-2) \times (x-3) \times \cdots \times 3 \times 2 \times 1 \end{aligned}$$

Which is itself, the product of another series.

So when calculating the zeta function in this form, we are actually having to compute, the product of 3 series , within a series, within a series. This may not

seem at all time or space efficient, but these sums are a lot easier for a computer to compute than evaluating $\zeta(s)$ by its definition that is:

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

To implement the binomial coefficient effectively, we would not be able to use it in its factorial form - this would take too long to compute. Instead, we can define the binomial coefficient another way.

If we take our equation for the definition of the binomial coefficient, and focus on the fraction of the right hand side:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

We can divide the numerator and denominator by $(n-k)!$, then we have that:

$$\binom{n}{k} = \frac{\frac{n!}{(n-k)!}}{k!}$$

Then expanding the numerator:

$$\binom{n}{k} = \frac{\frac{n(n-1)(n-2)...(n-k+1)(n-k)(n-k-1)(n-k-2)...(2)(1)}{(n-k)(n-k-1)(n-k-2)...(2)(1)}}{k!}$$

Notice how the denominator of the numerator exactly repeats what is above it, this means that all of these terms can cancel, such that we are left with:

$$\binom{n}{k} = \frac{n(n-1)(n-2)\dots(n-k+2)(n-k+1)}{k!}$$

Which can be rewritten as:

$$\binom{n}{k} = \frac{n^{\underline{k}}}{k!}$$

Where $n^{\underline{k}}$ is known as a falling factorial power and is defined as:

$$\begin{aligned} n^{\underline{k}} &= n(n-1)(n-2)\dots(n-k+1) \\ &= \prod_{x=1}^{k} (n-(x-1)) \\ &= \prod_{x=1}^{k-1} (n-x) \end{aligned}$$

So looking back at how to use this to calculate the binomial coefficient, we can say that:

$$\begin{aligned}\binom{n}{k} &= \frac{n!}{k!(n-k)!} \\ &= \frac{n^{\underline{k}}}{k!} \\ &= \frac{n(n-1)(n-2)\dots(n-(k-1))}{k(k-1)(k-2)\dots 1} \\ &= \prod_{i=1}^{k} \frac{n+1-i}{i} \\ &= \prod_{i=0}^{k-1} \frac{n-i}{i+1}\end{aligned}$$

We now have a function, that we can be called from inside the zeta function, in order to calculate the appropriate binomial coefficient.

Implementing the binomial coefficient function as pseudocode would look as follows:

Algorithm 5 Binomial Coefficient Pseudocode

```
function BINOM(n, k)
    b ← 1
    for i ← 0 to k do
        b = b × (n−i)/(i+1)
    end for
    return b
end function
```

Rewriting our definition of the zeta function, with this new way of calculating the binomial coefficient, we can now say that:

$$\zeta(s) \approx \frac{1}{1-2^{1-s}} \sum_{n=0}^{\texttt{Terms}} \left(\frac{1}{2^{n+1}} \sum_{k=0}^{n} \left((-1)^k \left(\prod_{i=0}^{k-1} \frac{n-i}{i+1} \right) (k+1)^{-s} \right) \right)$$

We are then able to convert both the Zeta Function Pseudocode and the Binomial Coefficient Pseudocode into Python code

```
# Binomial Coefficient
# \binom{n}{k} = \prod^{k-1}_{i=0} \frac{n-i}{i+1}
def binom(n, k):
    v = 1
    for i in range(k):
        v *= (n - i) / (i + 1)
    return v

# Global Zeta Function
# \zeta(s) \approx \frac{1}{1-2^{1-s}} \sum_{n=0}^{\texttt{TERMS}}
    \left( \frac{1}{2^{n+1}} \sum_{k=0}^{n} \left( (-1)^k
    \left(\prod^{k-1}_{i=0} \frac{n-i}{i+1} \right) (k+1)^{-s} \right)
    \right)
def zeta(s, TERMS=100):
    if s == 1:
        return float('inf')
    # \sum_{n=0}^{\texttt{TERMS}} \left( \frac{1}{2^{n+1}}
        \sum_{k=0}^{n} \left( (-1)^k \left(\prod^{k-1}_{i=0}
        \frac{n-i}{i+1} \right) (k+1)^{-s} \right) \right)
    sum1 = 0
    for n in range(TERMS):
        # \sum_{k=0}^{n} \left( (-1)^k \binom{n}{k} (k+1)^{-s} \right)
        sum2 = 0
        for k in range(n + 1):
            sum2 += (-1) ** k * binom(n, k) * (k + 1) ** (-s)
        sum1 += sum2 * (1 / (2 ** (n+1)))
    return sum1 * (1 / (1 - 2 ** (1 - s)))
```

Now we have a very efficient program that can be used to calculate any value of $\zeta(s)$.

This function will be used extensively throughout my technical solution to plot graphs, find data points and to try to help provide some proof for the Riemann Hypothesis.

2.4.3 Circular Queue

For parts of my project, I will require a FIFO Queue Data Structure. I will be implementing this using a circular queue. This data structure will be a class, that has a set of methods that manipulate the data in the queue. The first of these methods will be the procedure that is called to initialise the queue. When an instance of the Queue class is created, the Queue class will require an input_queue parameter, which will have the beginning items of the queue in it. Potentially also a max_size parameter could be used if a static data structure is wanted by the user rather than a dynamic one. When the class is initialised, a

size variable will need to be created to keep track of the current length of the queue, as well as a max_size. The queue will then have to be created with the input queue at the front, and any remaining spaces left in the queue will have to be filled in with some filler character. Variables for the front and rear pointers of the queue will then need to be created.

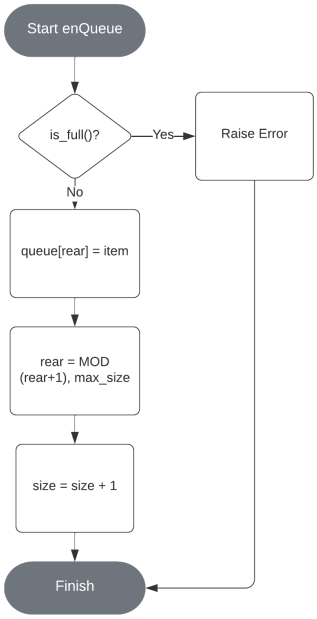
One of the methods used in this implementation of a Circular Queue will be the is_full function. If the size of the queue is equal to it's maximum size, then the function will return true, otherwise it will return false.

Another method is the function is_empty. If the size of the queue is 0, then the function will return true, otherwise it will return false.

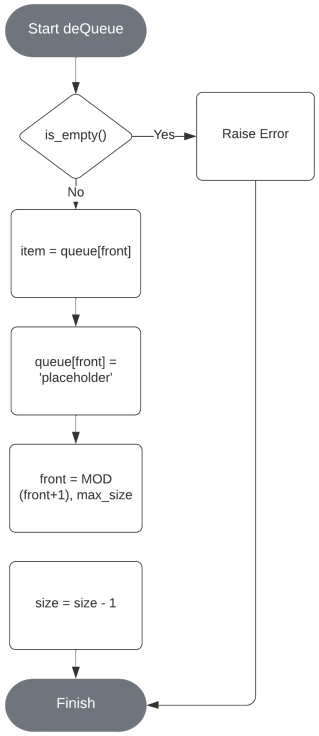
As well as this is the enQueue method, which will take a parameter of an item. In this procedure, the item is appended to the rear of the circular queue. This procedure can only be run if the queue is not already full, if it is full, then an error will be thrown. If the queue is not full, then the element in the queue that is at the position of the rear pointer will be set equal to the item that is being enqueued. The rear pointer will then need to be incremented. The rear pointer is increased by 1, but if this value goes above the max_size of the queue, then it will need to be set to 0. For this the mod function can be used. The variable for the size of the list will then need to be incremented by one.

Similar to this is deQueue, where an item is removed from the front of the queue and it's value is returned. This function can only be run if the queue is not already empty, otherwise an error will be thrown. The variable item will be used to hold the value of the element in the queue where it's index is equal to the front pointer. The element at this position can then be set to some blank / placeholder value. This is not necessary for computation but helps with readability when printing the queue. The front pointer will then need to be incremented by one, but be modded in a similar way to the rear queue during an enQueue. The size variable will need to be decremented by one and the item variable will be returned.

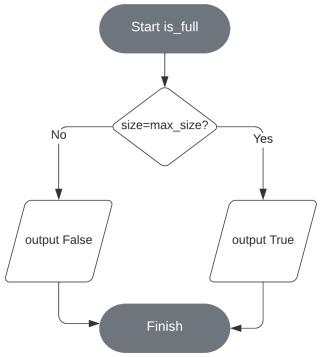
The flowcharts for each of these subroutines looks as follows:



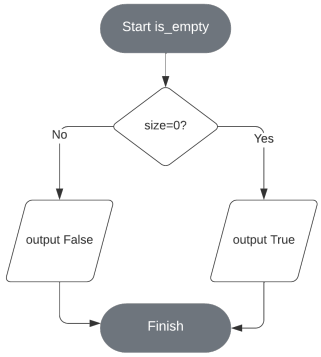
(a) enQueue method flowchart



(b) deQueue method flowchart



(c) is_full method flowchart



(d) is_empty method flowchart

Figure 22: Flowcharts for Circular Queue Methods

Implementing the circular queue in pseudocode looks as follows:

Algorithm 6 Circular Queue Pseudocode

```
1:  class CIRCULAR QUEUE(input_queue, max_size)
2:      public function __INIT__(self, input_queue, max_size)
3:          size ← length input_queue
4:          front ← 0
5:          rear ← size
6:          if size > max_size then
7:              return Error
8:          else
9:              blanks ← list()
10:             no_of_blanks ← max_size − size
11:             for i ← 0 to no_of_blanks do
12:                 blanks = blanks +′ placeholder′
13:             end for
14:             queue ← input_queue + blanks
15:         end if
16:     end function
17:
18:     public prodecure ENQUEUE(item)
19:         if IS_FULL() = True then
20:             return Error
21:         else
22:             queue[rear] ← item
23:             rear ← mod rear + 1, max_size
24:             size ← size + 1
25:         end if
26:     end procedure
```

Algorithm 7 Circular Queue Pseudocode Continued

```
27:     public function DEQUEUE(item)
28:         if IS_EMPTY() = True then
29:             return Error
30:         else
31:             item ← queue[front]
32:             queue[front] ←' placeholder'
33:             front ←  mod front + 1, max_size
34:             size ← size − 1
35:             return item
36:         end if
37:     end function
38:
39:     public function IS_FULL()
40:         if size = max_size then
41:             return True
42:         else
43:             return False
44:         end if
45:     end function
46:
47:     public function IS_EMPTY()
48:         if size = 0 then
49:             return True
50:         else
51:             return False
52:         end if
53:     end function
54: end class
```

And then converting this pseudocode to python it will look as follows:

```
class Queue:

    """
    Implementation of a circular queue

    Contains the subroutines:
        - enQueue
        - deQueue
        - is_full
        - is_empty

    """
```

```
def __init__(self, input_queue, **kwargs):
    self.input_queue = input_queue
    self.size = len(self.input_queue)
    self.front = 0
    self.rear = len(self.input_queue)
    if 'max_size' in kwargs.keys():
        self.max_size = kwargs['max_size']
    else:
        self.max_size = len(self.input_queue)
    if self.size > self.max_size:
        raise IndexError("max_size must be greater than or equal to
            the size of the input queue")
    else:
        self.blanks = [False for i in range(self.max_size -
            self.size)]
        self.queue = self.input_queue + self.blanks

def enQueue(self, item):
    """Appends an item to the rear of the circular queue"""
    if self.is_full():
        raise IndexError("Tried to enqueue to a full queue")
    else:
        self.queue[self.rear] = item
        self.rear = (self.rear+1) % self.max_size
        self.size += 1

def deQueue(self):
    """Remove and return the value at the front of the circular
        queue"""
    if self.is_empty():
        raise IndexError("Tried to dequeue from an empty queue")
    else:
        item = self.queue[self.front]
        self.queue[self.front] = False
        self.front = (self.front+1) % self.max_size
        self.size -= 1
        return item

def is_full(self):
    """Check if the circular queue is full"""
    return self.size == self.max_size

def is_empty(self):
    """Check if the circular queue is empty"""
    return self.size == 0
```

2.4.4 Binary Insertion Sort

Throughout the program I will be required to sort sets of data. To sort the data, I will be using a binary insertion sort algorithm, due to it's low time and space complexity. Although not too efficient on large data sets, I should only be handling relatively small data sets during this program so the binary insertion sort should suffice for all sorting in the program.

A normal insertion sort works by having a queue which will originally contain each element in the original array. The item at the front of the queue will be removed from the queue and be placed in an array containing the elements in sorted order. When an element is taken from the queue and put in the sorted array, it is put in a position such that the elements either side of it are lower and higher than it. Once evert element has been moved from the queue to it's correct position in the sorted list, the sorted list will contain every element that was in the original array but in sorted order.

A binary insertion sort is a variation of the normal insertion sort, but instead it uses a binary search, rather than a linear search to find the position that the element will go in the sorted list. This improves the time complexity of the program from O(n) to O(log n).

In pseudocode, the algorithm for a binary insertion sort looks as follow:

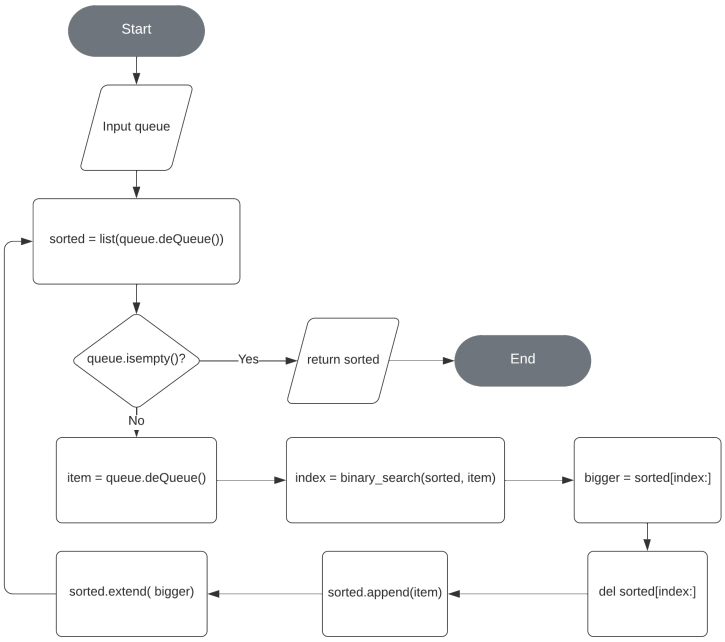


Figure 23: Binary Insertion Sort Flowchart

This algorithm written in pseudocode looks as follows:

Algorithm 8 Binary Insertion Sort Pseudocode

```
1:  function BINARY_INSERTION_SORT(data)
2:      queue = Queue(data)
3:      sorted ← list()
4:      sorted ← sorted + queue.DEQUEUE()
5:      while not queue.IS_EMPTY() do
6:          index ← binary_search(sorted, item)
7:          bigger ← sorted[index :]
8:          delete sorted[index :]
9:      end while
10:     return sorted
11: end function
```

And then writing the algorithm in python:

```
def binary_insertion_sort(data, descending=False):

    """
    Sorts a list of data into ascending order using a binary insertion
        sort
    """


    try:
        array = list(map(float, data ))
    except ValueError:
        array = data
    queue = Queue(array)
    sorted = [queue.deQueue()]
    while not queue.is_empty():
        item = queue.deQueue()
        index = binary_search(sorted, item)
        bigger = sorted[index:]
        del sorted[index:]
        sorted.append(item)
        sorted.extend(bigger)
    if not descending:
        return sorted
    else:
        return sorted[::-1]
```

This python variation of the binary insertion sort algorithm also includes an option to return the sorted list in descending order, as well as the default ascending order.

2.5 File Structure

In this section, I will showcase the full planned File Structure for the program.

- **program**
  - **user_interface**
    - **introduction_ui**
      - __init__.py
      - Various python files for the introduction UI
    - **ivnestigation_ui**
      - __init__.py
      - Various python files for the investigation UI
    - **login_ui**
      - __init__.py
      - Various python files for the login UI
    - **notes_ui**
      - __init__.py
      - Various python files for the notes UI
    - **summary_ui**
      - __init__.py
      - Various python files for the summary UI
    - **tutorial_ui**
      - __init__.py
      - Various python files for the tutorial UI
    - __init__.py
    - main_menu.py
    - mat_menu.py
    - progress.py
  - **utils**
    - __init__.py
    - computational_functions.py
    - cryptography_functions.py

        * database_functions.py
        * email_functions.py
        * file_handling.py
        * mathematical_functions.py
        * number_systems.py
        * screen_design.py
        * user.py
    - __init__.py
    - introduction_section.py
    - investigation_section.py
    - login_section.py
    - main_section.py
    - notes.py
    - summary_section.py
    - tutorial_section.py
- main.py

As you can see from this file structure list, there will be many python files involved in my project. I have sufficiently modularised each file, and each directory to make imports of functions across files as streamlined as possible. main.py will be the main entry point in the program, and will run the code located in the **program** directory. The **user_interface** directory will contain a large amount of files, however these files are somewhat unimportant when it comes to the complexity and meeting my objectives of the project. Inside the **user_interface** directory is a directory for the UI's for each of the main sections in the program. Inside each of these directories is the code required to run the user interface for the page.

Each of these different UI sections will then it's own python file in the **program** directory, which is used to display the UI at the correct point, and allow buttons, tabs, and other interactive features to work.

The more interesting and complex code can be found in the **utils** directory. Inside this directory are key algorithms, functions, procedures, and classes that are used throughout the program.

2.6 HCI and Screen Designs

Throughout this section, I will be creating initial designs for my screen pages used throughout the program.

Each page will have a consistent structure and layout, such that it will not be confusing for a user to be able to use and navigate page.

I will not be showing the design of every page used - because they are largely similar - I will instead just be showcasing and explaining some of the designs for my most important or unique parts of the project.

2.6.1 Generic Template

The consistent design for my user interface will follow the following template design.

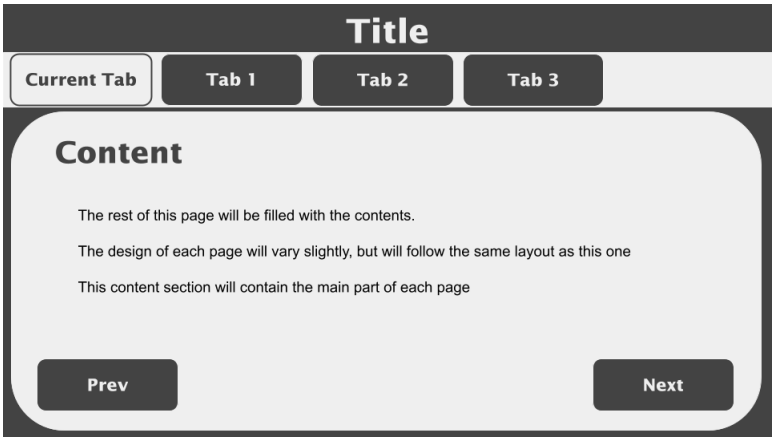


Figure 24: Generic Screen Design

At the top of the page is the title. This is there to show the user which page they are on.

Below this, is the tab line. Each tab on the bar will be a clickable button, that will take the user to a different screen. The user will be able to go through the tabs in whichever order they please, which may not necessarily be in the order they are arranged; although the program will be designed such that the next tab will follow on from the previous one.

The section with the title 'Content' will show the main part of each page. For example, if the page was showing a graph plot, it will be displayed in this section.

At the bottom of the page, are the previous and next buttons, this will display the previous and next tabs respectively. These button will allow for the user to navigate between each of the pages more easily.

2.6.2 Main Menu Screen Design

The main menu is the main entry point for the user into the program. This will be what the user first sees when they launch the program, and will display all of the possible menu options for them.

It is important that this page looks visually appealing, but is also very functional and allows for efficient traversal between screen pages.

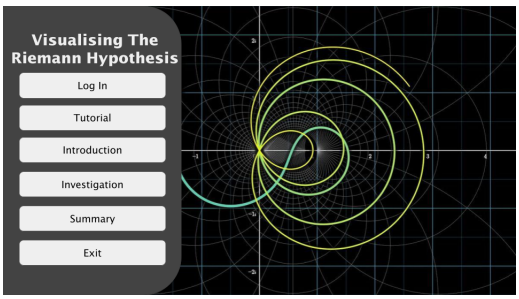


Figure 25: Main Menu Screen Design

On the left hand side, is displayed a menu of all of the sections of the program that the user is able to navigate to. It is recommended that the user goes through them in the order shows, however, they will be able to navigate the sections however they please.

Selecting each menu option will show a new page for that respective section, and close the main menu page. When the user is finished with that section, they will then be sent back to the main menu screen.

If the user chooses to log in, the their username will shown on the main menu screen, as demonstrated in the figure below.

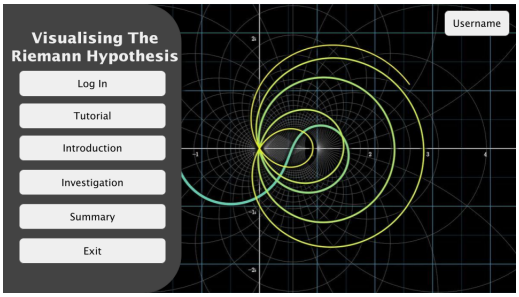


Figure 26: Logged In Main Menu Screen Design

2.6.3 Login Screen Design

Figure 27: Login Screen Design

From the figure above, we can see a prototype design for the Login Page. On the tab bar are the different options for the user. The login screen starts on the page that will be most-used, that is to be able to sign in to the program. However, the tab bar means that if a user wants to create an account, or reset their password, then they have the option there.

The user's password will not be visible when they type it in for security reasons, but in case they want to see what the have typed, then there is a show button for them.

If the user incorrectly types their username/password, then an error message will be displayed just above the submit button.

The submit button will take the user's input for their username and password, and check it against the stored values in the database. If they are incorrect, the error will be displayed, if the credentials are correct, then the user will be taken back to the main menu, and it will display their username, showing that they are logged in.

The user will then be able to go to any part of the program they wish, and have their data saved for the next time they want to use the program.

2.6.4 Graph Plots Screen Design

As part of the investigation section of the project, the user will have the opportunity to interact with various graphs. An example of how this would look is shown in the figure below.

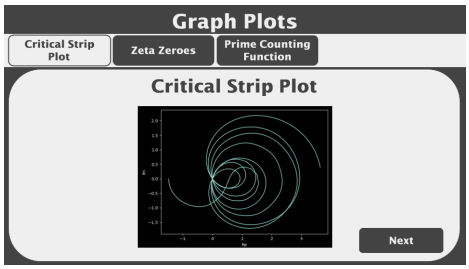


Figure 28: Graph Plot Screen Design

The graph will be animated, such that it updates in real time as the user is looking at it. The user will also be able to change input values into the graph plots.

2.6.5 Calculator Screen Design

Another part of the investigation section will be where the user can calculate values of the zeta function. The basic version will be where the user can enter a single input value, and an output value is calculated.

A more advanced version is where the user defines a range of value for the input, by given a start value, a stop value and a step value for the difference between the values.

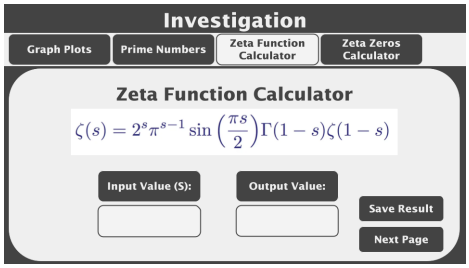


Figure 29: Zeta Function Calculator Screen Design

The user will then be able to store the output values from this calculator in a database. The database will contain values of zeta that every user who uses the program calculates, and each input output pair calculated will be stored with the user who found out the value, then there will be a leaderboard of who has calculated the most values for the zeta function.

3 Technical Solution

3.1 Program Re-design

While creating my solution to the problem, I realised that my initial design was lacking in some areas, and a lot if it needs to be updated.

3.1.1 Database Redesign

The first area that needed to be redesigned was the database. I realised that parts of my old design were almost obsolete and unnecessary, while my database was also not in complete third normal form. I redesigned that database, such that it now contains the following tables:

Users Table

The user's table is vastly unchanged from my original design. The main change is that I removed the User_ID Field, as it was obsolete and used the Username as the primary key as it will be unique for each user. This table will store the credentials for each user who has registered an account

Table: Users				
Field	**Key**	**Data Type**	**Validation**	**Notes**
Username	Primary	TEXT	\w{1,20}	Unique Username for each user
Email		TEXT	.+@.+\..+	A unique email address for each user
Password		TEXT		A hashed version of the user's password

Table 9: Data table for the Users Table

User's Answers Table

The User's Answers table will be used to store the answers that each user has input to the various questions that are asked throughout the program.

Table: User's Answers				
Field	**Key**	**Data Type**	**Validation**	**Notes**
Question_No	Primary	INTEGER	\d+	The number of the question that is being answered
Username	Primary	TEXT	\w{1,20}..+	The username of the user that is answering the question
UserAnswer		TEXT		The user's answer to the question

Table 10: Data table for the User's Answers Table

Notes Table

The Notes table will be used to store the notes that each user makes when they use the program, so that these notes can be loaded next time they log in

Table: Notes				
Field	**Key**	**Data Type**	**Validation**	**Notes**
Username	Primary	TEXT	\w{1,20}	The username of the user who is making the note
Section	Primary	TEXT	[a-zA-Z]	The section that the note is being made for
Text		TEXT		The user's note

Table 11: Data table for the Notes Table

Zeroes Table

The zeroes table will store the input for every zeta zero calculated and saved by users

Table: Zeroes				
Field	**Key**	**Data Type**	**Validation**	**Notes**
Zero_ID	Primary	INTEGER	\d+	A unique ID for this zeta zero
Zero_Real_ Input		REAL	\d(\. \d+)?	The real part of the input for this zeta zero
Zero_Imag_ Input		REAL	\d(\. \d+)?	The imaginary part of the input for this zeta zero

Table 12: Data table for the Zeroes Table

User Zeroes Table

The user zeroes will relate every zeta zero calculated back to the user who calculated it.

Table:User Zeroes				
Field	**Key**	**Data Type**	**Validation**	**Notes**
Zero_ID	Primary	INTEGER	\d+	A unique ID for this zeta zero
Username	Primary	TEXT	\w{1,20}	The user that computed this zeta zero

Table 13: Data table for the User Zeroes Table

Zeta Table

The zeta table will contain all of the stored inputs and outputs of the zeta function.

Table: Zeta				
Field	**Key**	**Data Type**	**Validation**	**Notes**
Zeta_ID	Primary	INTEGER	\d+	A unique ID for each input-output pair
Input_Real		REAL	\d(\.\d+)?	Real part of the input for this value
Input_Imag		REAL	\d(\.\d+)?	Imaginary part of the input for this value
Output_Real		REAL	\d(\.\d+)?	Real part of the output for this value
Output_Imag		REAL	\d(\.\d+)?	Imaginary part of the output for this value

Table 14: Data table for the Zeta Table

User Zeta Table

The user zeroes table will relate every zeta value calculated to the user who computed it.

Table: User Zeta				
Field	**Key**	**Data Type**	**Validation**	**Notes**
Zeta_ID	Primary	INTEGER	\d+	A unique ID for this zeta value
Username	Primary	TEXT	\w1,20	The user that computed this value

Table 15: Data table for the User Zeta Table

Correct Answers Table

The correct answers table lists all of the acceptable answers for the questions asked in the program.

Table: Correct Answers				
Field	**Key**	**Data Type**	**Validation**	**Notes**
Questions_No	Primary	INTEGER	\d+	The question that this answer is correct for
Correct Answer	Primary	TEXT	\w+	A valid answer for that question

Table 16: Data table for the Correct Answers Table

Questions Table

The questions table lists all of the questions that are asked in the program.

Table: Questions				
Field	**Key**	**Data Type**	**Validation**	**Notes**
Question_No	Primary	INTEGER	\d+	
Question	Primary	TEXT	\w+	The questions that is being asked

Table 17: Data table for the Questions Table

Entity Relationship Diagram

Now that the database has been properly designed, here is the entity relationship diagram for the database, showing all of the connections between tables

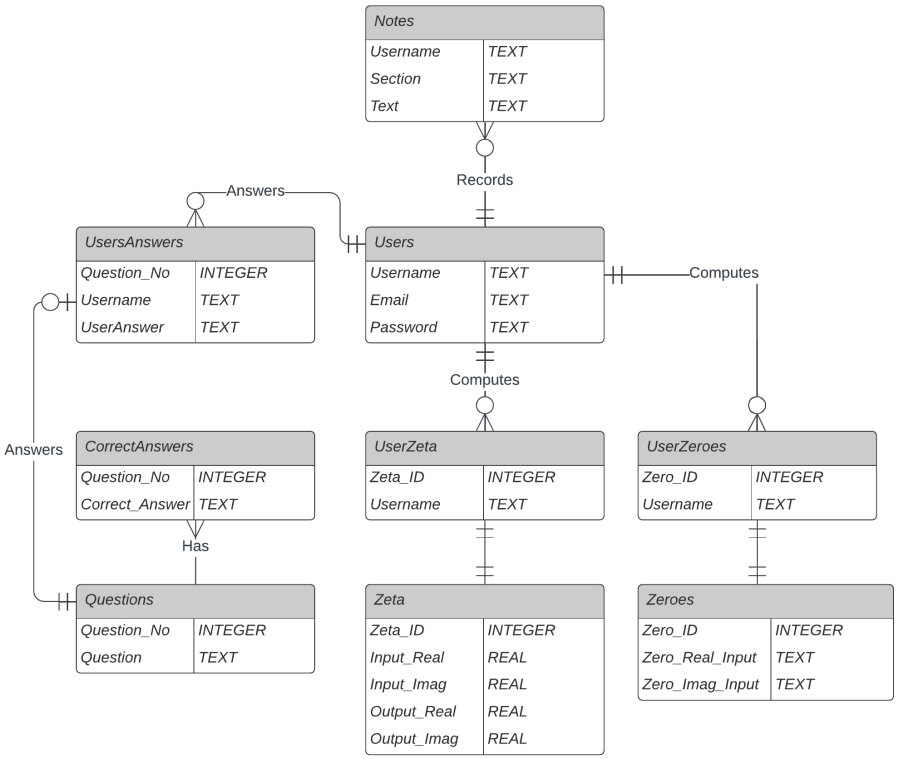


Figure 30: Database Entity Relationship Diagram

Third Normal Form

Unlike the previous database that I had designed, this database is now in third normal form. To be in third normal form, a database must meet the following criteria:

First Normal Form:

- All rows must be unique
- Each cell must contain a single value
- Data must be atomic

Second Normal Form:

- Must be of first normal form
- Contains no partial dependencies

Third Normal Form:

- Must be of second normal form
- Contains no non-key dependencies

Since redesigning the database, there are now no non-key dependencies, no partial dependencies, all data is atomic, each cell only contains one value and all of the rows are unique, meaning that this database is now in third normal form. This helps throughout the creation of my technical solution, as data will be easier to change and maintain, there will be no duplication of data, meaning that data integrity is maintained, and smaller tables with fewer fields allows for faster searches.

3.1.2 User Interface Redesign

Although my screen designs will look the same in the technical solution as they did in my design, I have made vast changes to how these designs will be implemented and displayed throughout the program.

Each page in the GUI will have it's own file in the **user_interface** directory. Each of these files will contain a class that is used to configure the correct GUI for that page. Then, in the **program** directory, there will be a python file for each of the different sections of the program. In each of these python files there will be a class for each page of the GUI. This class creates an instance of the class used to configure the GUI and actually displays the page. Each of these classes then inherits methods and attributes from a class for that overall section, which sets up key buttons and tabs to function properly and contains some methods that are run frequently in the different classes for each page.

These section classes then inherit methods and attributes from a main Screen class, which has some commonly run functions and procedures in it.

There are also separate files and classes that are used to display graphs, as these are displayed in the program as their own screens. There is one class for a graph that is static and one for dynamic graphs.

Below is a class diagram for the tutorial section, showing how all of these classes relate to each other.

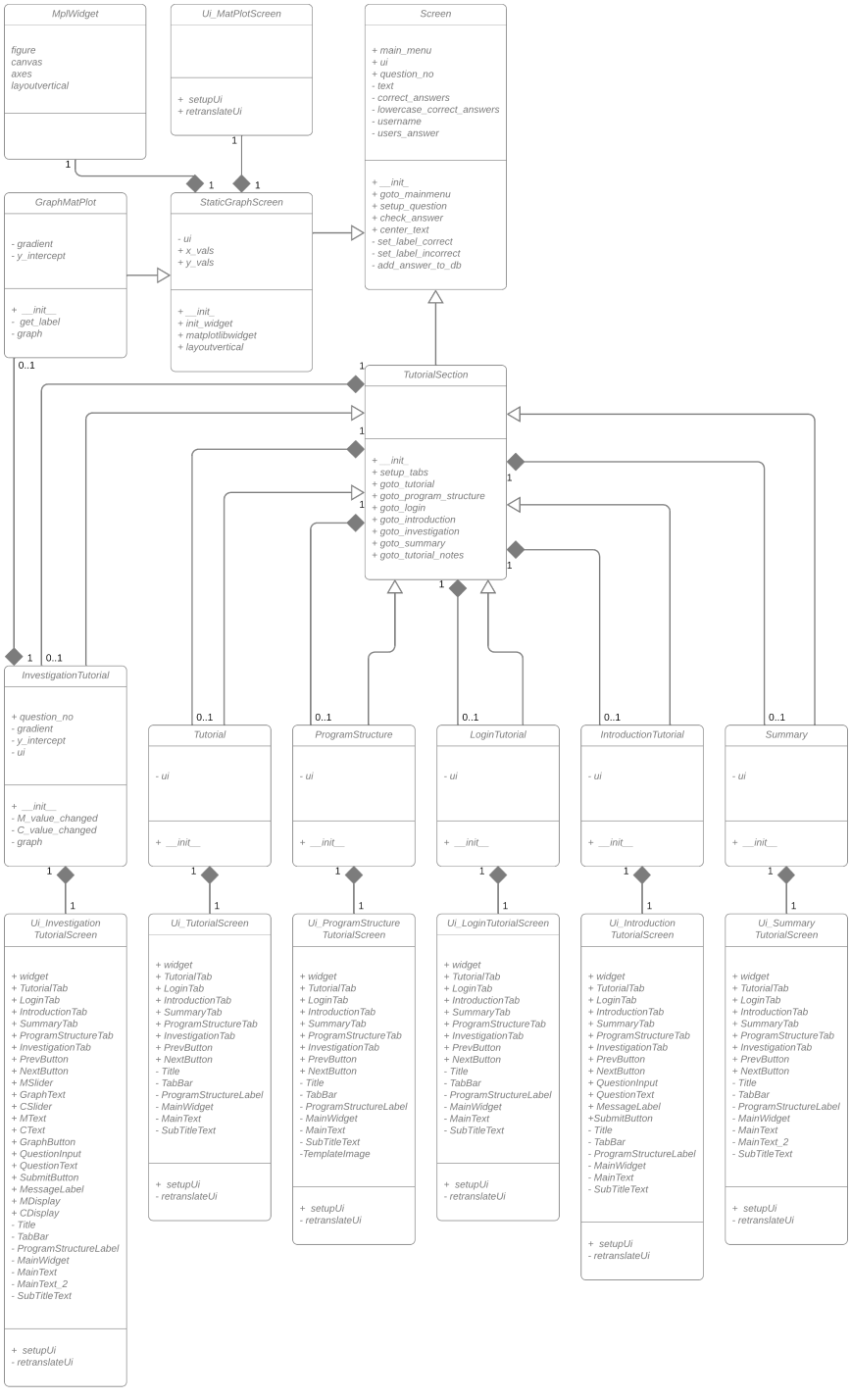


Figure 31: Tutorial Class Diagram

If we look at the investigation screen in this tutorial section, we can see that we initially have the Ui_InvestigationTutorialScreen class; which is used to configure the GUI for this page. This class is then in instance of the InvestigationTutorial class, which displays the Investigation Tutorial Screen and contains methods used for displaying the graph and updating the display when the sliders change.

This InvestigationTutorial class then inherits and is an instance of the TutorialSection class, which contains methods that are used throughout the different classes in the tutorial section.

The TutorialSection class then inherits methods and attributes from the Screen class, which contains methods and functions that are used in screen classes throughout the program.

However, the InvestigationTutorial class also has an instance of the GraphMatPlot class, which is used to set up the graph screen for that page. The GraphMatPlot screen then inherits methods and attributes from the StaticGraphScreen class.

The StaticGraphScreen class has creates instances of the UI_MatPlotScreen class and the MplWidget class, which together allow a new page to be created with a graph on it. The StaticGraphScreen class also inherits attributes from the Screen class.

This setup may seem overly complex for simply showing a GUI, however, it has been designed such that there is no repeated code, and each class is used for just one purpose.

3.2 Full Technical Solution

See `https://github.com/jackm245/Riemann-Hypothesis/` for the full write-up, or appendix A at the bottom of this document

3.3 Code Contents Page

Code Contents Page - 1		
Code	**Description**	**Pages**
main function	The main entry point to the program	
MainMenu class	Displays the main menu	
Progress class	Displays the user's progress through the program	
LoginSection class	The login system, inherited by all other login classes	
ResetPassword2 class	Allows the user to permanently change their passwords	
ResetPassword class	Screen that requires user to login before password change	
ForgottenPassword2 class	User enters verification code to prove identity	
ForgottenPassword class	Verification code sent to user's email	
SignUp class	Allows for the creation of a new user	
Login class	Allows a user to sign in to an account	
TutorialSection class	Main class for the tutorial, inherited by all other tutorial classes	
Tutorial class	Entry point to the tutorial section	
ProgramStructure class	Displays the Program Structure Tutorial Screen	
LoginTutorial class	Displays the Login Tutorial Screen	
IntroductionTutorial class	Displays the Introduction Tutorial Screen	
InvestigationTutorial class	Displays the Investigation Tutorial Screen	
GraphMatPlot class	Displays graph in the Investigation Tutorial Screen	
SummaryTutorial class	Displays the Summary Tutorial Screen	
IntroductionSection class	Main class for the introduction, inherited by all other introduction classes	
Introduction class	Entry point to the Introduction section	
HistoricalBackground class	Displays the Historical Background Screen	

Table 18: Code Contents Page - 1

Code Contents Page - 2		
Code	**Description**	**Pages**
WhatIsTheRiemann Hypothesis class	Displays the What is the Riemann Hypothesis Screen	
PracticalApplications class	Displays the Practical Applications Screen	
InvestigationSection class	Main class for the investigation, inherited by all other investigation classes	
CalculateZeroes2 class	Displays the zeta zeroes that the user has calculated	
CalculateZeroes class	Asks the user how many zeta zeroes they want to calculate	
Zeroes class	Displays the zeroes screen in the investigation	
CalculatorLeaderboard class	Displays the zeroes screen in the investigation	
TableCalculator2 class	Displays the zeta values that the user calculated	
TableCalculator class	Allows the user to input values for the table calculator	
SingleCalculator class	Allows the user to calculate values of the zeta function	
Calculator class	Displays the calculator screen in the investigation section	
PrimeNumbers class	Displays the prime numbers screen in the investigation section	
ZetaApproximation MatPlot class	Displays a graph showing the convergent nature of the riemann zeta function	
ZetaApproximation class	Allows the user to input a value to graph	
PrimeCountingFunction MatPlot class	Displays a graph of the prime counting function and other related functions	
PrimeCountingFunction MatPlot class	Displays a graph of the prime counting function and other related functions	
PrimeCountingFunction class	Displays the prime counting function screen	
ZetaZeroesMatPlot class	Displays a graph of the zeta zeroes	
ZetaZeroesPlot class	Displays the zeta zeroes mat plot screen	
PolarGraphMatPlot class	Displays a polar graph of the zeta function	
PolarGraph class	Asks the user for an input for the polar graph	
GraphPlot class	Entry point to the investigation section	

Table 19: Code Contents Page - 2

Code Contents Page - 3		
Code	**Description**	**Pages**
SummarySection class	Class inherited by all other classes in the summary section	
Summary class	Main entry point to the summary section	
TheoryRecap class	Displays the theory recap screen	
InvestigationResults class	Displays the investigation results screen	
Conclusion class	Displays the conclusion and & evaluation screen	
Imapct class	Displays the impact screen	
Notes class	A class inherited by all other classes in the notes section	
TutorialNotes class	Allows the user to make notes on the tutorial	
IntroductionNotes class	Allows the user to make notes on the introduction	
InvestigationNotes class	Allows the user to make notes on the investigation	
SummaryNotes class	Allows the user to make notes on the summary	
utils directory	Contains key functions, procedures and algorithms that are used throughout the program	
binary_search function	Searches for a target in a set of data with time complexity O(log n)	
binary_insertion_sort function	Sorts a set of data efficiently using a queue	
Queue class	An abstract data structure of a FIFO Queue	
Stack class	An abstract data structure of a LIFO Stack	
save_zeta_to_file function	Uses the binary insertion sort and dictionaries to save data to a csv file	
change_datatype function	Uses structural pattern matching to change the datatype of a variable, when the datatype is given as a string	
cryptography_functions file	Contains functions used for hashing passwords	
database_query function	A function used to easily be able to query the database	
create_database function	Creates the database and all of its tables, populating some of the tables	
get_id function	Uses recursion to auto increment the ID for a table in the database	
email_functions file	Sends the verification code to the user's email if they have forgotten their password	
touch function	Uses exception handling to create a file	
remove function	Uses exception handling to remove a file	

Table 20: Code Contents Page - 3

Code Contents Page - 4		
Code	**Description**	**Pages**
ncr function	Uses iterables to compute factorials and thus the binomial coefficient	
zeta function	Uses generators to compute the riemann zeta function for complex numbers	
sieve_of_eratosthenes function	Uses numpy arrays to find all prime numbers up to a given limit	
integration function	Integrate a given mathematical function between two limit	
exponential_integral function	Calculates the exponential integral for a given input	
prime_power_function function	Calculates the prime power function	
Number class	Abstract data type for a given number	
Complex class	Abstract data type for complex numbers	
Screen class	Class inherited by all other GUI classes	
MplWidget class	A GUI widget that allows graphs to be displayed in the GUI	
StaticGraphScreen class	Default class for displaying static graphs. Creates an instance of MplWidget and inherits Screen class	
DynamicGraphScreen class	Default class for displaying dynamic graphs. Inherits StaticGraphScreen class	
ProgramUser class	An instance of this class is used to store the user's credentials during the runtime of the program	
user_interface directory	An extensive directory, where each python file is the configuration for the GUI for that page	

Table 21: Code Contents Page - 4

Although this is not a list of every single class and function in the program, I have included a list of all of the important and complex features. For example, in the utils directory I describe almost every class/functions due to their high significance, whereas with the user_interface directory, there are 6 folders, 43 files and 40 classes, however all they do is configure the UI which is very repetitive and doesn't require a lot of technical programming skill, thus they have not been mentioned in this contents page.

3.4 Completeness of Solution

Completeness of Solution - 1		
Objective	**Achieved**	**Evidence**
1. The program will be interactive and engaging for the user	Yes	
1.(a) There will be various questions throughout the program that the user can answer	Yes	
1.(b) The user will be able to make notes on any of the content in the program	Yes	
1. (c) The user will be able to choose what they do and where they go in the program, not forced along a single route	Yes	
1. (d) The user will be able to input their own values into various functions and graphs	Yes	
2. The user will be able to save their progress through the program via a login system	Yes	
2. (a) The login system should allow the user to:	Yes	
2. (a) i. Sign in to their account	Yes	
2. (a) ii. Create a new account	Yes	
2. (a) iii. Reset their password if they want to change it	Yes	
2. (a) iv. Reset their password if they have forgotten it	Yes	
2. (b) Multiple users will be able to make separate accounts on the system	Yes	
2. (c) The user's data such as passwords must be saved securely	Yes	
2. (d) The user must be able to see how they have progressed throughout the program	Yes	

Table 22: Completeness of Solution Table 1

Completeness of Solution - 2		
Objective	**Achieved**	**Evidence**
3. The user will be given some background information about the Riemann Hypothesis such that almost anyone could feel comfortable using the program	Yes	
3. (a) The user should be able to learn the historical background of the Riemann Hypothesis	Yes	
3. (b) The user should be able to learn what imaginary and complex numbers are	Yes	
3. (c) The user should be able to learn what the Riemann Hypothesis states	Yes	
3. (d) The user should be able to learn about the practical applications of the Riemann Hypothesis	Yes	
4. The program will allow the user to plot various graphs in order to develop their understanding of the Riemann Zeta function and allow them to learn about it	Yes	
4. (a) A polar graph of the Riemann Zeta Function	Yes	
4. (b) A graph of the prime counting function, and other related functions used to estimate it	Yes	
4. (c) A graph of the zeroes of the Riemann Zeta Function	Yes	
4. (d) A visualisation of the convergence of the infinite series in the Riemann Zeta Function	Yes	
5. The user will be able to calculate specific values of the Riemann Zeta Function	Yes	
5. (a) A single calculator, where the user inputs a complex input and receives an output	Yes	

Table 23: Completeness of Solution Table 2

Completeness of Solution - 3		
Objective	**Achieved**	**Evidence**
5. (b) A table calculator, where the user can calculate the zeta function for a range of input values	Yes	
5. (c) A leaderboard showing how many values of the zeta function, each user has computed	Yes	
5. (d) The program will allow the user to store these value(s) to a database and to a csv file	Yes	
6. The user will be able to calculate the zeroes of the Riemann Zeta Function	Yes	
6. (a) The user will input how many zeroes they want to calculate	Yes	
6. (b) The program will calculate these zeroes	Yes	
6. (c) The zeroes will be displayed to the user in a table	Yes	
7. The user will be able to store the data that they have collected	Yes	
7. (a) The program will include a database of multiple values and zeroes of the zeta function	Yes	
7. (a) i. The user will be able to store values of the zeta function	Yes	
7. (a) ii. The user will be able to store the non-trivial zeta zeros	Yes	
7. (a) iii. The user program will retrieve data from the database and display its contents suitably to the user	Yes	
8. The Program will have a graphical user interface	Yes	

Table 24: Completeness of Solution Table 3

4 Testing

4.1 Iterative Testing

Imports

With many different python files with many different functions and classes, imports are imperative to make sure that every function/class can get accessed from where it needs to. Especially with many directories and subdirectories in the project - the file structure does not make imports easy.

Initially, when writing investigation_section.py, I needed to import the Complex class from utils/number_systems.py. So I tried too:

```
from utils/number_systems import Complex
```

But this returned an error. So I tried many variations of this such that

```
from utils.number_systems import Complex
```

And

```
from utils import number_systems.Complex
```

```
from number_systems import Complex
```

Until I found out the I need to create a __init__.py file in the utils/ directory. __init__ files are used by python to manage import and create modules from subdirectories. So in utils/__init__.py I Wrote:

```
from .number_systems import Complex
```

This makes utils its own module that can be imported from elsewhere. So now in investigation_section.py I Wrote:

```
from .utils import Complex
```

This then allows me to import the Complex class, although it's inside a different file which is inside a different directory.

But this wasn't the end of it. When now trying to create a button that goes from the investigation_section to the main_menu, I tried to write

```
from .main_section import main_menu
```

However, as I was also trying to import investigation_section from main_section, this gave me a circular import error. So instead of importing the main_menu when at the top of the file, instead, it has to be imported from within the function it is run.

```
def goto_mainmenu(self):
    from ..main_section import MainMenu
    self.main_menu = MainMenu()
    self.hide()
```

Which seems slightly unpythonic, but in practice it is the only way to get around this circular import error, unless I totally redesign the file structure of my code.

Zeta Zeroes

I encountered various different errors when creating the algorithm to find the non-trivial zeta zeroes. The only way I am able to compute the zeta zeroes is through brute force - trying every single possibility before I get them. If this weren't the case, the Riemann Hypothesis would already be solved. When brute forcing to try and get the zeta zeroes, I let the real part of the input to the zeta function be $\frac{1}{2}$, and let the imaginary part of the input be a function of time. Such that as the time spent trying to compute the zeroes increases, the imaginary part of the input increases. The first issue was to do with the accuracy of the input values and output. When changing the imaginary part of the input, if I incremented this value too much, then I could potentially skip over possible zeroes. For example, the first zeta zero is at 14.1, but if I increase the imaginary input by 0.2 every time the graph updates, then I would calculate $\zeta(14.0)$ and $\zeta(14.2)$, but completely skip 14.1, meaning that I would have missed a zero. However, if I increment the imaginary input by too small of a number, then the program will take longer to find each zero, and it could accidentally double count some zeroes. If the imaginary input incremented by 0.001 each time, then it might mistake 14.099, 14.100, and 14.101 as three zeroes, even though there was only one there.

On order to solve this problem, I used the method of trial and improvement until I got an algorithm that was fast, and didn't overcount or undercount.Eventually, I came up with the algorithm that

```
def is_zeta_zero(real, imag):

    """
    Given a complex number, the function checks to see if this number is
    approximately a root (zero) of the Riemann Zeta Function
    """

    zeta_value = zeta(real, imag)
    return abs(zeta_value) < 10e-3
```

```
def find_zeta_zeroes(no_of_zeroes):
    zeroes = Queue()
    count = 0
    while zeroes.get_size() < no_of_zeroes:
        accuracy = count // 500 + 100
        real = 1/2
        imag = count / accuracy
        if is_zeta_zero(real, imag) and [real, round(imag, 1)] not in
            zeroes.get_queue():
           zeroes.enQueue([real, round(imag, 1)])
        count += 1
    return zeroes
```

Where count is a function of time and updates at regular intervals, such that changing the imag input by $\frac{count}{accuracy}$, when accuracy $= \frac{count}{500} + 100$, allows zeta zeroes to be calculated by a high precision while also not trying so many values that the algorithm takes ages. Furthermore, having the range of values from -0.001 to 0.001 for the zero to be in further ensures that no zeroes get skipped. However, the drawback of this is that it does take a long time, and I am also only able to measure the zeroes to 1 decimal place, or I get too many repeats.

4.2 Post-Development Testing

Post-Development Test Table - 2						
Objective	**Test #**	**Input**	**Expected Output**	**Actual Output**	**Outcome**	**Comments**
1.						
1.(a)						
1.(b)						
1.(c)						
1.(d)						
2.						
2.(a)						
2.(a)i.						
2.(a)ii.						
2.(a)iii.						
2.(a)iv.						
2.(b)						
2.(c)						
2.(d)						
3.(a)						
3.(b)						
3.(c)						
3.(d)						
4.						
4.(a)						
4.(b)						
4.(c)						
4.(d)						
5.						
5.(a)						
5.(b)						
5.(c)						
5.(d)						

Table 25: Post Development Test Table - 1

Post-Development Test Table - 2						
Objective	**Test #**	**Input**	**Expected Output**	**Actual Output**	**Outcome**	**Comments**
6.						
6.(a)						
6.(b)						
6.(c)						
7.						
7.(a)						
7.(a)i.						
7.(a)ii.						
7.(a)iii.						
8.						

Table 26: Post Development Test Table - 2

One error which I discovered during my testing was to do with saving values of the zeta function to the csv file. After going back and looking at the problem, I realised that I had a logic error where the csv file was unable to have two records where the InputReal record was the same. This is due to how I chose to sort of the data. The code at the time look as such:

```
def save_zeta_to_file(csv_values, filepath, regex, index, fieldnames):

    """
    Given a list of complex numbers, combine these with the contents of
        the
    file that they are going to be saved to, sort these values using the
        first
    real number, and save them back into the csv file
    """

    if not os.path.isfile(filepath):
        os.mknod(filepath)
    with open(filepath, 'r') as csv_file:
        csv_reader = csv.reader(csv_file)
        for row in csv_reader:
            if row != fieldnames:
                csv_values.append(list(map(str, row)))
    sorting_dict = {list(map(float,
        re.findall(regex, ','.join(row))))[index] : row for row in
            csv_values}
    sorted_keys = binary_insertion_sort(list(set(sorting_dict.keys())))
    sorted_values = [sorting_dict[key] for key in sorted_keys]
    with open(filepath, 'w') as csv_file:
        csv_writer = csv.writer(csv_file)
        csv_writer.writerow(fieldnames)
```

```
        for row in sorted_values:
            csv_writer.writerow(row)
```

The problems came with lines 16 and 17, which say that

```
    sorting_dict = {list(map(float,
        re.findall(regex, ','.join(row))))[index] : row for row in
            csv_values}
```

This creates a dictionary with the InputReal of each record as the key, and each record as the value, for all of the records in the file. However, if two records had the same InputReal, then they would create the same key. So then when I call

```
    sorted_keys = binary_insertion_sort(list(set(sorting_dict.keys())))
```

I am sorting the set of all of the keys. This removes duplicates. However, I have to sort the data as a set, otherwise the binary_insertion_sort will not work. So I need to change the data in order to eliminate any two records having the same first value. To do this, I added the number of the record onto the end of each key. This allows each record to still be sorted, and also make them unique.

With these changes, the working function now looks as follows:

```
def save_zeta_to_file(csv_values, filepath, regex, index, fieldnames):

    """
    Given a list of complex numbers, combine these with the contents of
        the
    file that they are going to be saved to, sort these values using the
        first
    real number, and save them back into the csv file
    """

    if not os.path.isfile(filepath):
        os.mknod(filepath)
    with open(filepath, 'r') as csv_file:
        csv_reader = csv.reader(csv_file)
        for row in csv_reader:
            if row != fieldnames:
                csv_values.append(list(map(str, row)))
    sorting_dict = {float(str(list(map(float,
        re.findall(regex, ','.join(row))))[index]) + str(row_no)) : row
            for row_no, row in enumerate(csv_values)}
    sorted_keys = binary_insertion_sort(list(set(sorting_dict.keys())))
    sorted_values = [sorting_dict[key] for key in sorted_keys]
    with open(filepath, 'w') as csv_file:
        csv_writer = csv.writer(csv_file)
        csv_writer.writerow(fieldnames)
```

```
        for row in sorted_values:
            csv_writer.writerow(row)
```

5 Evaluation

5.1 Objective Completion

In this section, I will list many of the key objectives of my program, and evaluate how well I have achieved them.

1. The program will be interactive and engaging for the user

User input in the forms of questions, notes and inputs to functions are abundant throughout the program. I met this first objective well by adding features that allow the user to actually control what happens in a program, instead of just getting a user to click a 'next' button over and over. Steps have been taken to make this program interactive and I believe that these features make the program both interactive and engaging. This objective has been completed well.

1. (a) There will be various questions throughout the program that the user can answer

In total, there are 10 questions asked to the user. The user's answer is then marked correct or incorrect. And if the user is signed in, their progress is saved, such that their answer is saved and they can see which questions they have answered correctly or incorrectly. I have met this objective.

1. (b) The user will be able to make notes on any of the content in the program

Throughout the program, there are many buttons that the user can click on to access the notes section, so it has been made very accessible. The notes have been split into a separate page for each section, and the user can make their notes as long as they like. This objective has been completed sufficiently.

1. (c) The user will be able to choose what they do and where they go in the program, not forced along a single route

Although there is a sort of default linear way to go through the program using the prev and next buttons, the tab bar on each page allows the user to jump between sections and pages as they wish. Although the actual layout and organisation of the pages could have been made more straightforwards, the user is still able to access all parts of the program at any point, so I have achieved this objective.

2. (a) The login system should allow the user to i. Sign in to their account ii. Create a new account iii. Reset their password if they want to change it iv. Reset their password if they have forgotten it

I am very happy with how they login system turned out in my project. It is very straightforward for the user to login, create an account, or change their password. Two key elements of this that stand out to me are the input validation for the usernames, emails, and passwords - making sure that they are always

of the correct type and structure; and also with the emailing of the user when they have forgotten their password - this is very neat. An improvement I could make to the login system would be to add a page or button to log out of an account. This would not be hard to implement at all (just requires updating the User instance of the ProgramUser class), as this would stop the user having to close and reopen the program every time they want to log out of their account. Overall however, the login system does meet all of the objectives and does so well.

2. (d) The user must be able to see how they have progressed throughout the program

The progress section in the program sufficiently completes this objective. It displays to the user which of the questions they have answered, and if they have done so successfully. This allows the user to see what they have gotten right or wrong, and also how much they actually know about the Riemann Hypothesis. If a user starts using this program knowing nothing about the Riemann Hypothesis, then reads through the introduction and uses the investigation, and is then able to answer some, if not all, of the questions, then they have made significant progress, not just through the program but of their understanding of the Riemann Hypothesis, and this is shown on the progress page. Therefore, I have achieved this objective.

3. The user will be given some background information about the Riemann Hypothesis such that almost anyone could feel comfortable using the program

Through the tutorial and the introduction section, the user is given sufficient explanations regarding to the historical background of the Riemann Hypothesis, what complex numbers are, what the Riemann Hypothesis is, and what the practical applications of the hypothesis are. This information is then solidified as knowledge through the questions asked in the program, showing that the user has understood what they've read and learnt. Through these pages in the introduction section, I have reached objectives 3. (a), 3. (b), 3. (c) and 3, (d), thus meeting the overall objective 3 sufficiently. However, it would help the user understand a lot more about the program if they had a more in-depth knowledge than the information that is said during the introduction. The problem is that this project is not solely about teaching people what the hypothesis is, and reading pages and pages of information about the hypothesis would more than likely not be interesting to the user - they could just go to the wikipedia page for that. Overall, I think this program has a good balance of introductory knowledge teaching while also allowing the user to actually investigate the Hypothesis.

4. (a) A polar graph of the Riemann Hypothesis

Overall, most of the graphs I created in the project were clear in what they showed, and allowed the user to understand further what the Riemann Hypothesis is about. However, I am especially pleased with the turn out of this graph.

Going from iteration to iteration of the zeta function, eventually finding a way of computing it such that the graph appears to calculate values of the zeta function took a lot of research but definitely was worth the while. An improvement that could be made to not just this graph but also other ones is to make them more interactive for the user. This would entail, for example, the user being able to hover with the mouse over different parts of the graph, and be shown a pop-up of the input and output of that point, or having the graph change colour slightly as the imaginary input increases with time. It's small additions like this that would make the program more appealing to use and provide just that much more useful information.

4. (d) A graph of the zeroes of the Riemann Zeta Function

The zeroes graph was one of the trickiest parts of this project, due to the constraints when it comes to how the values are actually calculated. There is no real formula for calculating the zeroes of the Riemann Zeta Function, so the only way to find them is to try every value as an input, and find which ones give an output of zero, essentially using brute force to try and find the zeta zeroes. It's not exactly an elegant design but is unfortunately the only way of doing so. This brute force method means that the zeroes take a long time to be calculated, with only a few points plotted on the graph after 20 or so minutes of waiting. One way of speeding this up would to just get a better computer. With a more advanced CPU with more cores, larger cache storage, and a faster clock speed; I would be able to calculate many more zeta zeroes in a given period of time. It would require a supercomputer to calculate enough zeta zeroes, to a good enough accuracy to actually get any meaningful data out of it. However, with the resources that I had available I'm pleased with what I managed to come up with and did complete this objective.

5. (b) A table calculator where the user can calculate the zeta function for a range of input values

The table calculator I have created in my program is very efficient. Given a range of input values, it will calculate the outputs from the zeta function for the inputs, and then display then in the table. This meets the objective. However, I don't think it's as developed as it could be. Having a table of a list of inputs and outputs isn't very useful or meaningful to the user. If instead, or as well, this data was represented using a graph - possibly even one involving domain colouring - or some other way to visualise the data, it would be a lot more useful to the user than just a table of numbers. I am happy with the table calculator feature as it meets the objective well, but I believe it could be more developed.

6. The user will be able to calculate the zeroes of the Riemann Zeta Function

Following on from many of the points I made while evaluating objective 4. (d), the algorithm used to calculate the zeta zeroes takes a long time to run. In the case of the graph, it could be updated as new zeroes are found, but for the

table as part of this objective, every zeta zero has to be calculated before the graph can be shown. This causes a relatively empty page to be displayed for a long time. Furthermore, this page is not updating itself either. This program does not utilise threads, so the program can only do one thing at a time. This means that while the zeroes are being calculated, the old page is not being redrawn, which can lead to some weird user interface bugs. To solve this issue, I could have implemented threads, such that even while the zeta zeroes are being calculated, the previous screen can still be redrawn. Overall, the zeta zeroes calculator does work in order to find the zeroes of the zeta function, but this algorithm could have been implemented better using threads to stop UI bugs.

7. The user will be able to store the data that they have collected

My database is extremely efficient. Having the database in third normal form means that queries to the database are very fast. Data can be stored and retrieved in an instant. Throughout many parts of the program, the user is given the opportunity to save values to the database, through just the click of a button. Whether it's values of the zeta function, or non-trivial zeroes, the user can permanently store data to the database. Using SQLite3 as the connection between the database and my program works very well, and they way I implemented querying the database though the database_insert, database_select, and database_query subroutines was very neat as it didn't require any global variables (which I was contemplating using before).

8. The Program will have a graphical user interface

Overall, I did well making the GUI. The page designs are all kept relatively simple and minimalistic, while still displaying all of the relevant information and allowing the user to access all parts of the program. The consistent design between pages allows for easy navigation. However, the GUI is by no means perfect. One design feature I would have liked to add would be for each page to be adjustable in it's size. Currently, the screen is a fixed size. It would taken a whole deal of time and effort, and having to learn a lot about the PyQt library, but there is a way to do this using grid layouts, and spacers. Unfortunately, I was already a long way into programming my project when I discovered this, and realised that it would have been a lot of effort to recode everything, while having very minimal reward. Another feature I would have liked to have added to my GUI would be animations. Small transitions between pages or buttons showing that they have been clicked are the small details that would make the project seem more professional. Moreover, the actual programming of my GUI was not very efficient. Creating so many classes and files was not just convoluted and confusion, but just plain inefficient. Instead, I could have created some global features that were common to each page e.g, the colours, the title position, the tab bar, the buttons - and had a template class that had these by default. Then each page could inherit this class and also add whichever extra features it needs.

To conclude, I met all of my objectives well, but there were still some areas in which I feel as if I could have improved my program by adding extra features

or re-designing certain parts.

5.2 Independent Feedback

5.3 Evaluation of Independent Feedback

6 Appendix A - Technical Solution Source Code

7 Appendix B - Links to Resources

7.1 Technical Solution Source Code

7.2 Testing Video

https://www.youtube.com/watch?v=LR21yevVk74

References

3Blue1Brown. *Visualizing the Riemann zeta function and analytic continuation*. Dec. 2016. URL: https://www.youtube.com/watch?v=sD0NjbwqlYw&t=756s.

College, Westmont. *Writing a Fraction Class*. 2016. URL: http://djp3.westmont.edu/classes/2016_09_CS010/Lectures/Lecture_27.pdf.

Franklin, James. *Tackling A Level Projects in Computer Science*. 1st ed. PG Online Limited, 2020.

Kuntner, Nikolaj. *Understanding and computing the Riemann zeta function*. June 2020. URL: https://gist.github.com/Nikolaj-K/996dba1ff1045d767b10d4d07b1b032f.

Magazine, Quanta. *The Riemann Hypothesis, Explained*. Jan. 2021. URL: https://www.youtube.com/watch?v=zlm1aajH6gY&t=606s.

Mathworld, Wolfram. Jan. 2022. URL: https://mathworld.wolfram.com/RiemannZetaFunction.html.

Meurer, Aaron. *Verifying the Riemann Hypothesis with SymPy and mpmath*. Mar. 2020. URL: https://www.asmeurer.com/blog/posts/verifying-the-riemann-hypothesis-with-sympy-and-mpmath/.

Numberphile. *Riemann Hypothesis - Numberphile*. Mar. 2014. URL: https://www.youtube.com/watch?v=d6c6uIyieoo.

Riemann, Bernhard and David Wilkins. *On the Number of Prime Numbers less than a Given Quantity. (Ueber die Anzahl der Primzahlen unter einer gegebenen Gr¨osse.)* 1998. URL: https://www.claymath.org/sites/default/files/ezeta.pdf.

Shepherd, Markus. *Visualising the Riemann Hypothesis*. Apr. 2016. URL: https://www.riemannhypothesis.info/2016/04/visualising-the-riemann-hypothesis/.

singingbanana. *The Riemann Hypothesis*. Jan. 2014. URL: https://www.youtube.com/watch?v=rGo2hsoJSbo.

Sparks, Ben et al. *MEI A Level Further Mathematics Core Year 1*. 4th ed. Hodder Education, 2017.

Sparks, Ben et al. *MEI A Level Further Mathematics Core Year 2*. 4th ed. Hodder Education, 2017.

Wikipedia. *Bernhard riemann*. Nov. 2021. URL: https://en.wikipedia.org/wiki/Bernhard_riemann.

– *Binomial coefficient*. Jan. 2022. URL: https://en.wikipedia.org/wiki/Binomial_Coefficient.

– *Complex analysis*. Dec. 2021. URL: https://en.wikipedia.org/wiki/Complex_analysis.

– *Complex number*. Jan. 2022. URL: https://en.wikipedia.org/wiki/Complex_number.

– *Complex number*. Jan. 2022. URL: https://en.wikipedia.org/wiki/Complex_number.

– *Complex plane*. Sept. 2021. URL: https://en.wikipedia.org/wiki/Complex_plane.

– *Dirichlet eta function*. Nov. 2021. URL: https://en.wikipedia.org/wiki/Dirichlet_eta_function.

Wikipedia. *Domain coloring*. Oct. 2021. URL: https://en.wikipedia.org/wiki/Domain_coloring.
– *Euclidean Algorithm*. Dec. 2021. URL: https://en.wikipedia.org/wiki/Euclidean_algorithm.
– *Gamma function*. Jan. 2022. URL: https://en.wikipedia.org/wiki/Gamma_function.
– *Imaginary number*. Jan. 2022. URL: https://en.wikipedia.org/wiki/Imaginary_number.
– *Imaginary unit*. Jan. 2022. URL: https://en.wikipedia.org/wiki/Imaginary_unit.
– *Leonhard euler*. Jan. 2022. URL: https://en.wikipedia.org/wiki/Leonhard_euler.
– *Logarithmic integral function*. Dec. 2021. URL: https://en.wikipedia.org/wiki/Logarithmic_integral_function.
– *Mill's constant*. Jan. 2022. URL: https://en.wikipedia.org/wiki/Mills%27_constant.
– *Proof of the Euler product formula for the Riemann zeta function*. Dec. 2021. URL: https://en.wikipedia.org/wiki/Proof_of_the_Euler_product_formula_for_the_Riemann_zeta_function.
– *Riemann hypothesis*. Jan. 2022. URL: https://en.wikipedia.org/wiki/Riemann_hypothesis.
– *Riemann zeta function*. Jan. 2022. URL: https://en.wikipedia.org/wiki/Riemann_zeta_function.
– *Zero of a function*. Jan. 2022. URL: https://en.wikipedia.org/wiki/Zero_of_a_function.
Wolfram Alpha. 2022. URL: https://www.wolframalpha.com/.