Exploring the Proportion of Prime Numbers in Quadratic Extensions of the Integers

Jamie Nelson

Follow this and additional works at: https://vc.bridgew.edu/honors_proj

Recommended Citation
Copyright © 2018 Jamie Nelson
Exploring the Proportion of Prime Numbers in Quadratic Extensions of the Integers

Jamie Nelson

Submitted in Partial Completion of the Requirements for Commonwealth Honors in Mathematics

Bridgewater State University

May 8, 2018

Dr. Jacqueline Anderson, Thesis Director

Dr. Laura Gross, Committee Member

Dr. Ward Heilman, Committee Member
Exploring the Proportion of Prime Numbers in Quadratic Extensions of the Integers

By

Jamie Nelson

Bridgewater State University
Bridgewater, MA

May 8, 2018
Acknowledgments

This paper would not have been possible without the generous contributions and support of three people.

First, I would like to thank my thesis director, Dr. Anderson, who is by far the most patient person I have ever met. Thank you for explaining everything in a way that I was able to understand and often reexplaining concepts until it finally stuck in my mind. I appreciate the way you took each question with serious consideration, even the ones that were not always completely thought through on my part. It helped me truly grasp the material and without your guidance I would have been lost. Thank you for being an amazing professor and mentor.

Second, I would like to thank Dr. Heilman, who met me at a time when I was stuck in a rut. Thank you for being the helping hand I needed to keep going. Your excitement for mathematics is contagious and reminds me why I love mathematics as well. Your wisdom is beyond measure and I appreciate all the advice you pass along. Your words of wisdom will always be in my mind to help me persevere through any challenges I face in the future.

Finally, I would like to thank Dr. Gross, who I met when I was just starting my collegiate career. Thank you for shaping me into the person I am today. I started college undecided with no direction of what I wanted to do in life. You guided me to my passion for mathematics and nudged me down the path of completing honors and conducting research. I want to spend the rest of my life doing research and I would not have known that without your guidance. Thank you for making me the person I am proud to be today.

Each of you provided a pivotal part in this paper, my undergraduate career, and my future plans. So thank you very much for being the caring, supportive mentors you are.
Exploring the Proportion of Prime Numbers in Quadratic Extensions of the Integers

Jamie Nelson

Bridgewater State University
Bridgewater, MA
May 8, 2018

ABSTRACT

A number $p$ is considered prime given it satisfies a specific property. If a prime number $p$ divides the product $\alpha \beta$, then either $p$ divides $\alpha$ or $p$ divides $\beta$ for any integers $\alpha$ and $\beta$. Prime numbers have been studied for centuries. However, it was not until 1896 that mathematicians Hadamard and de la Vallee Poussin proved the Prime Number Theorem. The Prime Number Theorem gives an approximation for the amount of prime numbers in the set of integers less than an upper limit $x$. The theorem states that $\lim_{x \to \infty} \frac{\pi(x)}{\frac{x}{\ln(x)}} = 1$, where $\pi(x)$ is the quantity of prime numbers less than or equal to $x$. For this thesis, we explored the quantity of primes numbers in quadratic extensions of the integers which is the set $\mathbb{Z}[\sqrt{r}]$ for different square-free integer values of $r$. We began with the Gaussian Integers. The Gaussian Integers are numbers in the form $a + bi$, where $i$ is defined as the square root of $-1$ and $a, b$ are elements of the integers. We determine which numbers are prime using the Gaussian Prime Theorem. Next, using the Python programming language, we counted the prime numbers and graphed the proportion. We also examined the sets $\mathbb{Z}[\sqrt{11}], \mathbb{Z}[\sqrt{15}], \mathbb{Z}[\sqrt{34}], \mathbb{Z}[\sqrt{35}]$, and $\mathbb{Z}[\sqrt{79}]$. Using a variation of the sieve algorithm, we wrote programs that count the prime numbers and graphs the results. The proportion of prime numbers out of the total amount of numbers is graphed to compare all the sets.
# Table of Contents

1 Introduction .................................................. 1

2 Background Information .................................... 2
   2.1 Quadratic Extensions of the Integers .................. 2
   2.2 Units ..................................................... 3
   2.3 Unique Factorization .................................... 4
   2.4 Irreducible Numbers ...................................... 4
   2.5 The Norm ................................................ 4
   2.6 Prime Numbers .......................................... 5

3 The Gaussian Integers ....................................... 7

4 Algorithm to Obtain Prime Numbers ...................... 11

5 Verification of Implementation ............................ 13

6 Another Example with Unique Factorization .............. 17

7 Examples without Unique Factorization .................. 19

8 Example with Different Class Number .................... 25

9 Conclusions .................................................. 28

10 Future Work ................................................ 30

References ...................................................... 31
# Programs

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1 Code for The Gaussian Integers</td>
<td>32</td>
</tr>
<tr>
<td>A.2 Code for $\mathbb{Z}[\sqrt{11}]$</td>
<td>50</td>
</tr>
<tr>
<td>A.3 Code for $\mathbb{Z}[\sqrt{15}]$</td>
<td>63</td>
</tr>
<tr>
<td>A.4 Code for $\mathbb{Z}[\sqrt{34}]$</td>
<td>76</td>
</tr>
<tr>
<td>A.5 Code for $\mathbb{Z}[\sqrt{35}]$</td>
<td>89</td>
</tr>
<tr>
<td>A.6 Code for $\mathbb{Z}[\sqrt{79}]$</td>
<td>102</td>
</tr>
<tr>
<td>A.7 Code for Proportion Graph</td>
<td>115</td>
</tr>
</tbody>
</table>
List of Figures

3.1 The graph of the amount of prime numbers in the Gaussian integers . . . 8
3.2 The graph of the amount of prime numbers in the Gaussian integers with
an approximate function . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
3.3 The graph of the proportion of prime numbers in the Gaussian integers . 9
3.4 The graph of the proportion of prime numbers in the Gaussian integers
compared to the integers . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
3.5 A close up of the proportion of prime numbers in the Gaussian integers
compared to the integers on a smaller set . . . . . . . . . . . . . . . . . . 10

5.1 On the left is the graph of prime numbers in the Gaussian Integers gen-
erated by our code and on the right is the picture provided by Dekker,
5.2 On the left is the graph of prime numbers in \(\mathbb{Z}[\sqrt{11}]\) generated by our
code and on the right is the picture provided by Dekker, T. J. (2010).
Primes in quadratic fields. Ithaca, NY. . . . . . . . . . . . . . . . . . . . 14
5.3 On the left is the graph of prime numbers in \(\mathbb{Z}[\sqrt{15}]\) generated by our
code and on the right is the picture provided by Dekker, T. J. (2010).
Primes in quadratic fields. Ithaca, NY. . . . . . . . . . . . . . . . . . . . 14
5.4 On the left is the graph of prime numbers in \(\mathbb{Z}[\sqrt{34}]\) generated by our
code and on the right is the picture provided by Dekker, T. J. (2010).
Primes in quadratic fields. Ithaca, NY. . . . . . . . . . . . . . . . . . . . 14
5.5 On the left is the graph of prime numbers in \(\mathbb{Z}[\sqrt{35}]\) generated by our
code and on the right is the picture provided Dekker, T. J. (2010). Primes in
quadratic fields. Ithaca, NY. . . . . . . . . . . . . . . . . . . . . . . . . . 15
5.6 On the left is the graph of prime numbers in \(\mathbb{Z}[\sqrt{79}]\) generated by our
code and on the right is the picture provided Dekker, T. J. (2010). Primes in
quadratic fields. Ithaca, NY. . . . . . . . . . . . . . . . . . . . . . . . . . 15
5.7 The count of prime numbers generated by the Gaussian Prime Theorem
and Dekker’s algorithm (Dekker, 2010) . . . . . . . . . . . . . . . . . . . 16

6.1 The graph of the amount of prime numbers in \(\mathbb{Z}[\sqrt{11}]\) . . . . . . . . . 17
6.2 The graph of the amount of prime numbers in $\mathbb{Z}[\sqrt{11}]$ with an approximate function .................................................. 18
6.3 The graph of the proportion of prime numbers in $\mathbb{Z}[\sqrt{11}]$ ................. 18
7.1 The graph of the amount of prime numbers in $\mathbb{Z}[\sqrt{15}]$ ....................... 19
7.2 The graph of the amount of prime numbers in $\mathbb{Z}[\sqrt{15}]$ with an approximate function .................................................. 20
7.3 The graph of the proportion of prime numbers in $\mathbb{Z}[\sqrt{15}]$ ................. 20
7.4 The graph of the amount of prime numbers in $\mathbb{Z}[\sqrt{34}]$ ....................... 21
7.5 The graph of the amount of prime numbers in $\mathbb{Z}[\sqrt{34}]$ with an approximate function .................................................. 21
7.6 The graph of the proportion of prime numbers in $\mathbb{Z}[\sqrt{34}]$ ................. 22
7.7 The graph of the amount of prime numbers in $\mathbb{Z}[\sqrt{35}]$ ....................... 22
7.8 The graph of the amount of prime numbers in $\mathbb{Z}[\sqrt{35}]$ with an approximate function .................................................. 23
7.9 The graph of the proportion of prime numbers in $\mathbb{Z}[\sqrt{35}]$ ................. 23
8.1 The graph of the amount of prime numbers in $\mathbb{Z}[\sqrt{79}]$ ....................... 25
8.2 The graph of the amount of prime numbers in $\mathbb{Z}[\sqrt{79}]$ with an approximate function .................................................. 26
8.3 The graph of the proportion of prime numbers in $\mathbb{Z}[\sqrt{79}]$ ................. 26
9.1 Graph of the proportion of prime numbers .................................................. 28
9.2 Graph of the first five hundred values of the proportion of prime numbers 29
Chapter 1

Introduction

Prime numbers have been studied for centuries. However, it was not until 1896 that mathematicians Hadamard and de la Vallee Poussin proved the Prime Number Theorem. The Prime Number Theorem states that when \( x \) is large, the number of primes less than or equal to \( x \) is approximately equal to \( x/\ln(x) \) (Silverman, 1997/2013, p. 91). Specifically,

\[
\lim_{x \to \infty} \frac{\pi(x)}{x/\ln(x)} = 1
\]

where \( \pi(x) \) is the number of primes less than or equal to \( x \in \mathbb{N} \). This theorem is used to measure the amount of integers that are prime. For this thesis, we propose to explore the proportion of prime numbers in the set \( \mathbb{Z}[\sqrt{r}] \) for different square-free integer values of \( r \).

In this thesis, we consider six different quadratic extensions of the integers, \( \mathbb{Z}[\sqrt{r}] \). We begin with some background information in Chapter 2. The prime numbers in the Gaussian Integers, when \( r = -1 \), are plotted using the Gaussian Prime Theorem in Chapter 3. We consider an algorithm to find the prime numbers for other sets \( \mathbb{Z}[\sqrt{r}] \) in Chapter 4. In Chapter 5 we verify that the programs we wrote using the algorithm accurately produce the prime numbers in these sets. We graph the set \( \mathbb{Z}[\sqrt{11}] \) in Chapter 6. Chapter 7 presents the graphs of \( \mathbb{Z}[\sqrt{15}] \), \( \mathbb{Z}[\sqrt{34}] \), and \( \mathbb{Z}[\sqrt{35}] \) which all do not have unique factorization. We consider a set with a different class number in Chapter 8. Chapter 9 presents some conclusions and in Chapter 10 we propose an extension of this work.
CHAPTER 2

Background Information

In order to fully understand this research, we need to understand some core definitions and principles.

2.1 Quadratic Extensions of the Integers

The quadratic extensions of the integers are numbers in the form \( a + b\sqrt{r} \) where \( a, b, \) and \( r \) are integers and \( r \) is square-free. In the sets we explored, we look at fixed values of \( r \). These are examples of algebraic numbers, or numbers which are roots of nonzero polynomials with rational coefficients.

The integers with adjoined multiples of a square root, \( a + b\sqrt{r} \), were discovered as solutions to quadratic equations \( X^2 - AX + B \). Suppose we have \( \alpha = a + b\sqrt{r} \) then the conjugate is defined as \( \bar{\alpha} = a - b\sqrt{r} \). We have that \( (X - \alpha)(X - \bar{\alpha}) \) equals \( X^2 - (\alpha + \bar{\alpha})X + (\alpha\bar{\alpha}) \). We know that \( \alpha + \bar{\alpha} \) is \( (a + b\sqrt{r}) + (a - b\sqrt{r}) \) which equals \( 2a \). Therefore, \( \alpha + \bar{\alpha} \) is a rational number. We have that \( \alpha\bar{\alpha} \) is \( (a + b\sqrt{r})(a - b\sqrt{r}) \) which equals \( a^2 - rb^2 \). Therefore, \( \alpha\bar{\alpha} \) is also a rational number. So, \( X^2 - (\alpha + \bar{\alpha})X + (\alpha\bar{\alpha}) \) is a polynomial with rational coefficients and roots \( \alpha \) and \( \bar{\alpha} \).

In this paper, we are only considering the ring of integers. Algebraic numbers have rational coefficients while the ring of integers consists of numbers which are roots of monic polynomials with integer coefficients, where monic means the leading coefficient is 1. If \( r \equiv 1 \pmod{4} \), then the ring of integers is the set of numbers of the form
\( a + b \left( \frac{1 + \sqrt{r}}{2} \right) \) and if \( r \equiv 2 \) or \( 3 \pmod{4} \), then the ring of integers is the set of numbers of the form \( a + b\sqrt{r} \), where \( a, b \in \mathbb{Z} \) (Conrad, 2017).

We limited our search to \( r \) values congruent to \( 2 \pmod{4} \) and \( 3 \pmod{4} \) for simplicity. Our exploration includes the sets \( a + b\sqrt{-1} \), \( a + b\sqrt{11} \), \( a + b\sqrt{15} \), \( a + b\sqrt{34} \), \( a + b\sqrt{35} \), and \( a + b\sqrt{79} \).

2.2 Units

A unit is an element \( u \) of a ring that satisfies \( uv = vu = 1 \). In other words, units are elements of the ring that have multiplicative inverses. In the set of integers, the only multiplicative inverses are 1 and \(-1\). However, in the Gaussian integers, the special set of numbers where \( r = -1 \), the only units are \( 1, -1, i \) and \(-i \). In the other sets, the units satisfy the equation \((a + b\sqrt{r})(c + d\sqrt{r}) = 1\). For example, if \( r = 2 \), \( 1 + \sqrt{2} \) is a unit because \((1 + \sqrt{2})(-1 + \sqrt{2}) = 1\). We also know that the norm of a unit is equal to 1 which means \(|(a + b\sqrt{r})(a - b\sqrt{r})| = 1\). Simplifying this expression, we find that \(|a^2 - rb^2| = 1\).

The equation \( a^2 - rb^2 = 1 \) is known as Pell’s equation. From Pell’s Equation Theorem, we know that if \( r \) is positive and square-free then \( a^2 - rb^2 = 1 \) always has a solution \((a_1, b_1)\) and all other solutions are formed by taking powers, \((a_1 + b_1\sqrt{r})^k \) where \( k = 1, 2, 3, \ldots \). For example, in the set \( a + b\sqrt{34} \) the smallest solution is \( a = 35 \) and \( b = 6 \) and all other units are \((35 + 6\sqrt{34})^k \) for \( k = 1, 2, 3, \ldots \) (Silverman, 2013).
2.3 Unique Factorization

The fundamental theorem of algebra states that every integer is formed from a product of prime numbers and that product is unique. The general case of this property is called unique factorization (Silverman, 2013). Some of the sets we look at are unique factorization domains such as the Gaussian integers and the set \( \mathbb{Z}[\sqrt{11}] \). The sets \( \mathbb{Z}[\sqrt{15}] \), \( \mathbb{Z}[\sqrt{34}] \), \( \mathbb{Z}[\sqrt{35}] \), and \( \mathbb{Z}[\sqrt{79}] \) do not have unique factorization (Dekker, 2010).

2.4 Irreducible Numbers

The definition of an irreducible number is often confused with the definition of primality. However, it is important to distinguish the two. An irreducible number is a number \( \alpha \) that is only divisible by itself, a unit \( u \) and a unit times the number \( u\alpha \) (Dekker, 2010). An example of an irreducible number int the integers is 5. We know that 5 is only divisible by itself, units in the integers which are 1 and \(-1\), and a unit times \( \alpha \) which are 5 and \(-5\).

2.5 The Norm

In each set of numbers, we can use the norm \(|a^2 - r(b^2)|\) to measure the magnitude of each element. The most important aspect of the norm is the multiplicative property, where \( N(\alpha\beta) = N(\alpha)N(\beta) \). If we find that the norm of some number \( \alpha \) equals \( n \), then we know that the possible factors of \( \alpha \) have to produce a norm that is a factor of \( n \). In other words if the norm of a number \( \alpha \) is prime in the integers then \( \alpha \) is also prime in that set.

For example, we have that \( \alpha = 4 + 5i \) in the Gaussian integers. The norm for the
Gaussian integers is $a^2 + b^2$. The norm of this number $\alpha$ is $16 + 25$ which equals 41. We know that 41 is prime so we can not split up 41 into two norm values and $4 + 5i$ does not factor. We can assume $4 + 5i$ is a Gaussian prime. Alternatively, if we have $\alpha = 5 + 3\sqrt{11}$, then the norm is $|25 - (11)(9)|$ which equals 74. We know that 74 has the factors 1, 2, 37, and 74. So the only possible numbers that could factor $5 + 3\sqrt{11}$ are numbers with the norm of 2 or 37. We would have to find $a$ and $b$ values which satisfy the equations $|a^2 - 11(b^2)| = 2$ and $|a^2 - 11(b^2)| = 37$.

### 2.6 Prime Numbers

A prime number is defined as satisfying a specific property. If we have a prime number $p$ and $p|\alpha \beta$ then either $p|\alpha$ or $p|\beta$. We know that in unique factorization domains an irreducible number is a prime number and vice versa. However, in domains where we do not have unique factorization, a prime number is an irreducible number but not every irreducible number is prime.

We can gain a deeper understanding by exploring a few examples. Again, we know 5 satisfies the definition for irreducibility in the integers, $5 = 5 \cdot 1$. We also know the number 5 is prime in the integers. However, 5 is not prime in the set of Gaussian integers. If we choose $\alpha$ to be $2 + i$ and $\beta$ as $2 - i$, we have that $\alpha \beta = 5$. We know that 5|5 but 5 does not divide $\alpha$ or $\beta$. Therefore, 5 is not prime in the Gaussian integers. In fact, 5 is not irreducible in the Gaussian integers. We note that the Gaussian integers are a unique factorization domain so every irreducible number is a prime number.

We can consider a number that is irreducible but not prime by looking at a non-unique factorization domain. If we have the number $\alpha = 1 + \sqrt{-3}$ in the set $\mathbb{Z}[\sqrt{-3}]$, we know that the norm of $\alpha$ is $1^2 + 3(1)^2$ which equals 4. If $\alpha$ is reducible in $\mathbb{Z}[\sqrt{-3}]$, that is there exists a factorization $\alpha = \beta \gamma$ where $\beta$ and $\gamma$ are not units, then $\beta$ and $\gamma$ would have a
norm of 2 \( N(\alpha) = N(\beta \gamma) = N(\beta)N(\gamma) = 4 = 2 \times 2 \). We need to find an \( a \) and \( b \) value to satisfy \( a^2 + 3b^2 = 2 \). There are no integer solutions to this equation. Therefore, there are no numbers with the norm \( 2 \in \mathbb{Z}[\sqrt{-3}] \) and \( \alpha \) is only divisible by itself and units. So, \( \alpha \) is irreducible. We also have that \((1 + \sqrt{-3})(1 - \sqrt{-3}) = 4 \) so \( \alpha \) divides the product of two numbers \( 2 \times 2 \) but \( \alpha \nmid 2 \). Therefore, \( \alpha \) is not prime (Gallian, 2017).

This is what makes our research interesting. Will the amount of prime numbers be the same in each set despite different elements being prime? Will all the sets be have a different amount of prime numbers? Will some sets have distinct properties like unique factorization that produce similar results? Is there any correlation between the sets that we can make by observing the proportion of prime numbers? We will try to explore these ideas further in the following chapters.
CHAPTER 3

The Gaussian Integers

We begin our exploration with the Gaussian integers ($\mathbb{Z}[i]$), the set of numbers in the form $a + bi$ where $i$ is defined as $\sqrt{-1}$ and $a, b \in \mathbb{Z}$. The Gaussian Prime Theorem states that all the Gaussian primes fit into one of three categories. First, we know $1 + i$ is a Gaussian prime. Second, if a prime $p$ in the integers is equal to $3 \pmod{4}$ then $p$ is also a prime in the Gaussian integers. Lastly, if a prime $p$ in the integers is equal to $1 \pmod{4}$ and $u^2 + v^2 = p$ then $u + vi$ is a Gaussian prime (Silverman, 2013). This encompasses all the Gaussian primes. We used the Gaussian Prime Theorem to generate the prime numbers and more importantly to count them. We created a program to determine the amount of prime numbers in $\mathbb{Z}[i]$ less than a given amount of numbers, shown in Figure 3.1.

Figure 3.1 shows that the graph has a similar shape to the graph of $y = \frac{x}{\ln(x)}$, shown as the red dashed line. This red line is the function that approximates the count of prime numbers in the integers as stated in the Prime Number Theorem. Through trial and error we obtain an approximation of the prime numbers in the Gaussian integers with a similar function. The function $y = \frac{x}{5\ln(x)}$ gives a close count to the actual amount for the first 5000 values, shown in Figure 3.2.
Fig. 3.1: The graph of the amount of prime numbers in the Gaussian integers

Fig. 3.2: The graph of the amount of prime numbers in the Gaussian integers with an approximate function
Next, we considered the percentage of prime numbers out of the total amount of numbers. This is shown in Figure 3.3, with the amount of numbers on the x-axis and the proportion on the y-axis. The Gaussian integers have many similar properties to the integers so it is no surprise that the graphs look very similar. We also see this comparison in Figure 3.4. The graph starts with a spike up to about half the values then quickly decreases and levels off at about 0.2. It is important to note that the amount of prime numbers in the Gaussian integers is consistently greater than the amount of primes in the integers.

Fig. 3.3: The graph of the proportion of prime numbers in the Gaussian integers
Fig. 3.4: The graph of the proportion of prime numbers in the Gaussian integers compared to the integers

Fig. 3.5: A close up of the proportion of prime numbers in the Gaussian integers compared to the integers on a smaller set

Next, we looked at another set with some similarities, $\mathbb{Z}[\sqrt{11}]$. However, first we explored an algorithm to find prime numbers in quadratic extension rings, such as $\mathbb{Z}[\sqrt{7}]$. 
CHAPTER 4
Algorithm to Obtain Prime Numbers

In order to produce the prime numbers, we need a way to determine which numbers are prime and which are not for the general case \( \mathbb{Z}[\sqrt{r}] \). Dr. Thedoreus Dekker presents such an algorithm in his paper *Primes in quadratic fields* (Dekker, 2010). He uses a variation of the sieve algorithm starting with a set of all possible norm values then looping through the set to remove norms of non-prime numbers.

First, we need to understand the parts of the algorithm. We define the discriminant \( d \) of the set as \( d = r \) if \( r \equiv 1 \pmod{4} \) and \( d = 4r \) if \( r \equiv 2 \) or \( 3 \pmod{4} \). For example, the discriminant of \( \mathbb{Z}[\sqrt{15}] \) is 60 because \( 15 \equiv 3 \pmod{4} \).

A rational integer \( q \) is a quadratic residue modulo \( n \) if there exists some number \( x \) such that \( x^2 \equiv q \pmod{n} \). We use Jacobi symbols to determine if a number is a quadratic residue or not. The value 1 is given to a number \( q \) that is a quadratic residue and \(-1\) otherwise. For example, \( 7^2 \equiv 9 \pmod{10} \) so 9 is a quadratic residue modulo 10 and the Jacobi symbol of \( \left( \frac{9}{10} \right) \) equals 1.

In the algorithm, the quadratic character is defined as the Jacobi symbol of the discriminant \( d \) over \( n \) for a given value \( n \). The Python library SymPy provides the calculation for the Jacobi symbol in our code. We begin with a set \( S \) containing all the prime divisors of \( d \) and elements \( s \) which are greater than 2 and less than the given maximum norm value and the quadratic character of \( s \) is 1. We ignore any even values.
For example, suppose we were considering the set \( \mathbb{Z}[\sqrt{11}] \) and the maximum norm value of 50. Then the discriminant is 44 and set \( S \) will contain the prime divisors of 44 specifically 2 and 11. The set \( S \) will also contain any odd number \( s \) between 2 and 50 where the Jacobi symbol \( \left( \frac{44}{s} \right) \) equals 1. Thus, in this case set \( S \) will contain 2, 5, 7, 9, 11, 19, 25, 35, 37, 39, 43, 45, and 49.

Next, we make a copy of set \( S \) called set \( T \). We start with the smallest value \( t \) in set \( T \) such that the quadratic character equals 1. Then we check if the product of \( t \) and any element \( s_i \) in set \( S \) is in set \( T \). If the product \( ts_i \) is in set \( T \) then we remove it. Once we have checked all the products \( t \) and \( s_i \), we select the next smallest \( t \) value in set \( T \) with quadratic character equal to 1. This continues until we have reached the end of set \( T \) or the next element \( t \) is greater than the square root of the maximum norm value we are exploring.

We will continue the example \( \mathbb{Z}[\sqrt{11}] \) with set \( S \) as 2, 5, 7, 9, 11, 19, 25, 35, 37, 39, 43, 45, and 49. Therefore, set \( T \) begins as the same values. We pick the smallest number in \( T \) with the quadratic character of 1, which is 5. Next, we multiply 5 by every element in set \( S \) and check if that product is in set \( T \). We observe that 5 * 5 or 25 is in set \( T \) so we remove that value. We also remove 5 * 7 and 5 * 9 from set \( T \). Once we have checked every product, we choose the next smallest \( t \) value with quadratic character 1, which is 7. Again, we check every product of 7 * \( s_i \). We find 49 or 7 * 7 in set \( T \) so we remove it. This process continues until we have set \( T \) as 2, 5, 7, 9, 11.

Now, set \( T \) contains all the norm values of prime numbers. We finish the algorithm by looping through all possible \( a \) and \( b \) values for a number \( a + b\sqrt{r} \) in our set \( \mathbb{Z}[\sqrt{r}] \) with norm less than the maximum norm. We check if any of these numbers have a norm in our set \( T \). If it does then that value is a prime number in \( \mathbb{Z}[\sqrt{r}] \) (Dekker, 2010). For example, 7 - 2\( \sqrt{11} \) has a norm of 5 which is in set \( T \) so 7 - 2\( \sqrt{11} \) is prime in \( \mathbb{Z}[\sqrt{11}] \).
We did not use the code provided by Dekker; we wrote our own variation (provided in the appendix). We had to verify that our code was correctly identifying prime numbers. Dekker’s paper includes graphs of the prime numbers using the $a$ value on the x-axis and the $b$ value on the y-axis for a number $a + b\sqrt{r}$. We were able to reconstruct this using his description of the algorithm and plotting at $(a, b)$, $(a, -b)$, $(-a, b)$, and $(-a, -b)$ for any prime number $p = a + b\sqrt{r}$ we found. We can see the comparison of those graphs to Dekker’s graphs in the Figures 5.1, 5.2, 5.3, 5.4, 5.5, and 5.6.

Fig. 5.1: On the left is the graph of prime numbers in the Gaussian Integers generated by our code and on the right is the picture provided by Dekker, T. J. (2010). *Primes in quadratic fields*. Ithaca, NY.
Fig. 5.2: On the left is the graph of prime numbers in $\mathbb{Z}[\sqrt{11}]$ generated by our code and on the right is the picture provided by Dekker, T. J. (2010). *Primes in quadratic fields.* Ithaca, NY.

Fig. 5.3: On the left is the graph of prime numbers in $\mathbb{Z}[\sqrt{15}]$ generated by our code and on the right is the picture provided by Dekker, T. J. (2010). *Primes in quadratic fields.* Ithaca, NY.

Fig. 5.4: On the left is the graph of prime numbers in $\mathbb{Z}[\sqrt{34}]$ generated by our code and on the right is the picture provided by Dekker, T. J. (2010). *Primes in quadratic fields.* Ithaca, NY.
From Figures 5.1, 5.2, 5.3, 5.4, 5.5, and 5.6, it appears that we have properly identified the prime numbers. We decided to check one more verification by comparing Dekker’s algorithm for the Gaussian integers with the Gaussian Prime Theorem in the graph of the count of prime numbers Figure 5.7.
Fig. 5.7: The count of prime numbers generated by the Gaussian Prime Theorem and Dekker’s algorithm (Dekker, 2010)

We observe in Figure 5.7 that these graphs are identical. This means our algorithm will give us the right prime numbers. Our next step is to use this verified code to construct the count of prime numbers in each set and graph the proportions.
Chapter 6

Another Example with Unique Factorization

So far we have seen two sets of numbers with unique factorization, the integers and the Gaussian integers. We provide one more example of a set with unique factorization, precisely \( \mathbb{Z}[\sqrt{11}] \). In Figure 5.2, we can see the graph of the count of prime numbers in the set \( \mathbb{Z}[\sqrt{11}] \). Again, the shape of the graph is similar to \( \frac{x}{\ln(x)} \) with a different magnitude. If we graph \( \frac{6x}{\ln(x)} \), we see that the function gives us an approximate amount of the count of prime numbers for the first 5,000 numbers, see Figure 6.2. Next, we graphed the proportion of prime numbers in Figure 6.3.

![Graph of Count of Prime Numbers in \( \mathbb{Z}[\sqrt{11}] \)](image)

Fig. 6.1: The graph of the amount of prime numbers in \( \mathbb{Z}[\sqrt{11}] \)
Fig. 6.2: The graph of the amount of prime numbers in \( \mathbb{Z} \sqrt{11} \) with an approximate function

Fig. 6.3: The graph of the proportion of prime numbers in \( \mathbb{Z} \sqrt{11} \)

We see an identical shape to the graphs of the Gaussian integers and the integers (Figure 3.4). The curve increases to around 0.6 then quickly decreases to about 0.2. Since all these sets have similar properties, it will be interesting to look at sets without unique factorization next.
CHAPTER 7

Examples without Unique Factorization

All of the sets we have explored so far displayed a specific property, unique factorization. The sets of numbers without unique factorization provide interesting context. When the set does not have unique factorization, we have numbers that are irreducible but not prime. We would like to observe if that makes a difference in the graphs. We began with the set $\mathbb{Z}[\sqrt{15}]$, see Figure 7.1.

Fig. 7.1: The graph of the amount of prime numbers in $\mathbb{Z}[\sqrt{15}]$
Fig. 7.2: The graph of the amount of prime numbers in $\mathbb{Z}\lfloor\sqrt{15}\rfloor$ with an approximate function

First, we observe the graph has the same shape as we have seen in Figure 3.1 and 6.1. Figure 7.2 shows an approximation for the first 5,000 values given by the function $y = \frac{97x}{100\ln(x)}$. We graph the proportion in Figure 7.3.

Fig. 7.3: The graph of the proportion of prime numbers in $\mathbb{Z}\lfloor\sqrt{15}\rfloor$

This is a much more interesting graph. The proportion does not spike in the beginning
as in the graphs of the unique factorization domains (Figure 3.3 and 6.3). The highest
the graph reaches is 0.2 then it levels off around the lower number, 0.125. Next in Figure
7.4, we look at $\mathbb{Z}[\sqrt{34}]$.

![Graph of prime numbers in $\mathbb{Z}[\sqrt{34}]$](image1)

**Fig. 7.4:** The graph of the amount of prime numbers in $\mathbb{Z}[\sqrt{34}]$

![Graph of prime numbers in $\mathbb{Z}[\sqrt{34}]$ with an approximate function](image2)

**Fig. 7.5:** The graph of the amount of prime numbers in $\mathbb{Z}[\sqrt{34}]$ with an approximate function

Again, we see little change in the graph of the count of prime numbers, except for the
magnitude. We graph the proportion of prime numbers in $\mathbb{Z}[\sqrt{34}]$ in Figure 7.6.

![Proportion of Prime Numbers in $\mathbb{Z}[\sqrt{34}]$](image1.png)

Fig. 7.6: The graph of the proportion of prime numbers in $\mathbb{Z}[\sqrt{34}]$

We observe a similar graph to $\mathbb{Z}[\sqrt{15}]$ (Figure 7.3). The graph does not increase to 0.5 in the beginning. It only reaches about 0.175 then it levels off to the lowest value of all the proportion graphs, 0.1. We checked if this pattern continued with the set $\mathbb{Z}[\sqrt{35}]$.

![Count of Prime Numbers in $\mathbb{Z}[\sqrt{35}]$](image2.png)

Fig. 7.7: The graph of the amount of prime numbers in $\mathbb{Z}[\sqrt{35}]$
Fig. 7.8: The graph of the amount of prime numbers in $\mathbb{Z}[\sqrt{35}]$ with an approximate function

In Figure 7.7, we see the same count graph for $\mathbb{Z}[\sqrt{35}]$. We graphed the proportion in Figure 7.9 and found a surprising result.

![Graph of Prime Numbers](image1.png)

The graph does not look like the other two graphs of non unique factorization domains (Figure 7.3 and 7.6). In fact, the proportion graph looks similar to the graphs of
unique factorization domains (Figure 3.4 and 6.3). It increases to about 0.4 then quickly decreases to level off at about 0.2. It is astonishing that this graph is different. So, we looked at one more property of these sets, the class number.
Chapter 8

Example with Different Class Number

The class number is used to describe the extent to which unique factorization fails in different integer rings. The non unique factorization domains we have looked at, \( \mathbb{Z}[\sqrt{15}] \), \( \mathbb{Z}[\sqrt{34}] \), and \( \mathbb{Z}[\sqrt{35}] \), all have a class number of 2. We added one more extension ring, specifically \( \mathbb{Z}[\sqrt{79}] \), which has a class number of 3. In Figure 8.1, we graphed the count of prime numbers in \( \mathbb{Z}[\sqrt{79}] \).

![Graph of prime numbers in \( \mathbb{Z}[\sqrt{79}] \)](image)

Fig. 8.1: The graph of the amount of prime numbers in \( \mathbb{Z}[\sqrt{79}] \)
Fig. 8.2: The graph of the amount of prime numbers in $\mathbb{Z}[\sqrt{79}]$ with an approximate function

Again, we see the same shape of $\frac{x}{\ln(x)}$ so we also graphed the proportion of prime numbers.

Fig. 8.3: The graph of the proportion of prime numbers in $\mathbb{Z}[\sqrt{79}]$
Figure 8.3, shows the proportion of prime number in $\mathbb{Z}[\sqrt{79}]$. We see that it matches the graphs of non unique factorization domains. However, the proportion values are smaller with a low value around 0.08. We conclude our findings with a comparison of all the proportion graphs generated.
Chapter 9
Conclusions

Below is the graph of the proportion of prime numbers for each of the sets explored in this paper. In Figure 9.1 and 9.2, we observe the rational integers in gray, the Gaussian integers in blue, \( r = 11 \) in orange, \( r = 15 \) in green, \( r = 34 \) in red, \( r = 35 \) in purple, and \( r = 79 \) in pink.

![Graph of the proportion of prime numbers](image)

Fig. 9.1: Graph of the proportion of prime numbers

We observe that the proportion of prime numbers is not always the same. In some cases, such as the Gaussian integers, \( \mathbb{Z}[\sqrt{11}] \), and \( \mathbb{Z}[\sqrt{35}] \) as well as \( \mathbb{Z}[\sqrt{34}] \) and \( \mathbb{Z}[\sqrt{79}] \),
Fig. 9.2: Graph of the first five hundred values of the proportion of prime numbers

the graphs converge to a similar value. Other graphs, such as $\mathbb{Z}[\sqrt{15}]$ and $\mathbb{Z}[\sqrt{-1}]$, have much different percentages of prime numbers. The proportion values for $\mathbb{Z}[\sqrt{15}]$, $\mathbb{Z}[\sqrt{34}]$ and $\mathbb{Z}[\sqrt{79}]$ have much smaller values than the other sets. This could be because each of those sets do not have unique factorization. However, that conclusion would ignore $\mathbb{Z}[\sqrt{35}]$ which also does not have unique factorization.

We could also hypothesize that all the graphs converge after a certain value. We can make a strong assumption that as the graphs grow closer to infinity the proportion converges to zero. However, looking at the graphs at what point do they intersect with one another and is there a distinction in how fast they collapse into one line if they do at all. There is much more room for exploration. We have concluded that each set does not behave the same, which itself is interesting.
This work can be extended in many different ways. We could apply the programs we used to other values of $r$. We could then add the proportion of prime numbers to the graph with the combined proportions of all the sets. This could help us find patterns and determine if any graphs are identical. We could use that and the properties of the sets to make conjectures. One hypothesis is that if the set does not have unique factorization then the proportion levels off to a smaller percentage in the first few values. If we had more sets to examine, we could test our hypothesis.

Also, if we continued to graph our results for larger amounts we could see if and when the proportion graphs converge. We hypothesize that the proportion is converging to 0 similar to the proportion of prime numbers in the integers. We could also try to formulate a set of rules for when a number is prime and when it is not for specific sets, similar to the Gaussian Prime Theorem. There are many exciting extensions for this work that can be easily explored given the code and knowledge from this paper.
References


Below we list the programs for each of the above graphs. The programs are written in the Python 3.6.3 programming language and include libraries sympy, pyplot, numpy, csv, and math.

A.1 Code for The Gaussian Integers

Listing A.1: Program that counts the prime numbers using the Gaussian Prime Theorem

```python
#Program to count the prime numbers in the set of Gaussian Integers
#using the Gaussian Prime Theorem
#Author: Jamie Nelson
#Mentor: Jackie Anderson

donumpy from sympy import *
import matplotlib.pyplot as plt
import numpy
import math

def main():

    #array to store the count and norm pairs
    countData = list()
    numberData = list()

    #array to store the x/ln(x) function
    xvalues = list()
    yvalues = list()
```

32
normMax = 0
numberCount = 0

#loop through max norm values while the total amount of numbers is less than 5000
while numberCount <= 5184:

    norm = 0
    maxA = int(sqrt(normMax))

    #start the total count of prime numbers at 0
    primeCount = 0
    numberCount = 0

    #for each "a" value from 1 to max A value
    #loop through all possible "b" values up to given max
    for a in range(1, (maxA + 1)):
        b = 0
        norm = (a * a) + (b * b)
        aMod = a % 4
        if isprime(a) and aMod == 3:
            #add one to count for this prime
            primeCount += 1
        while norm <= normMax:
            numberCount += 1
            if isprime(norm):
                #add one to count for this prime
                primeCount += 1
            b += 1
            norm = (a * a) + (b * b)

    #add this number count and the prime count value to an array
    #so we can graph the output after
    numberData.append(numberCount)
    countData.append(primeCount)
    normMax += 1

    #compute the values for x/ln(x)
    for x in range(2, 5184):
        xvalues.append(x)
        y = x / (math.log(x))
yvalues.append(y)

# plot the values for the amount of numbers on the x axis
# and the prime number count on the y axis
# also plot x/ln(x) in red dashed line to compare
plt.plot(numberData, countData, 'b-', label='Count of Gaussian Prime Numbers')
plt.plot(xvalues, yvalues, 'r--', label=r'$y = \frac{x}{\ln(x)}$')
plt.legend(loc='lower right')
plt.xlabel('Total Amount of Numbers')
plt.ylabel('Amount of Prime Numbers')
plt.title('Count of Prime Numbers in the Gaussian Integers')
plt.show()

main()
Listing A.2: Program that graphs an asymptotic function of the count of prime numbers

#Program to count the prime numbers in the set of Gaussian Integers
#using the Gaussian Prime Theorem and graphs an asymptotic function
#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math

def main():
    #array to store the count and norm pairs
    countData = list()
    numberData = list()
    #array to store the 7x/5ln(x) function
    xvalues = list()
    yvalues = list()
    normMax = 0
    numberCount = 0

    #loop through max norm values while the total amount of numbers is less than 5000
    while numberCount <= 5184:
        norm = 0
        maxA = int(sqrt((normMax)))
        #start the total count of prime numbers at 0
        primeCount = 0
        numberCount = 0

        #for each "a" value from 1 to max A value
        #loop through all possible "b" values up to given max
        for a in range(1, (maxA + 1)):
            b = 0
            norm = (a * a) + (b * b)
            aMod = a % 4
            #...
if isprime(a) and aMod == 3:
    #add one to count for this prime
    primeCount += 1
while norm <= normMax:
    numberCount += 1
    if isprime(norm):
        #add one to count for this prime
        primeCount += 1
    b += 1
    norm = (a * a) + (b * b)

#add this number count and the prime count value to an array
#so we can graph the output after
numberData.append(numberCount)
countData.append(primeCount)
normMax += 1

#compute the values for 7x/5\ln(x)
for x in range(2, 5184):
    xvalues.append(x)
    y = (7 * x) / (5 * math.log(x))
yvalues.append(y)

#plot the values for the amount of numbers on the x axis
#and the prime number count on the y axis
#also plot 7x/5\ln(x) in red dashed lines to compare
plt.plot(numberData, countData, 'b-', label='Count of Gaussian Prime Numbers')
plt.plot(xvalues, yvalues, 'r--', label=r'$y = \frac{7x}{5\ln(x)}$')
plt.legend(loc='lower right')
plt.xlabel('Total Amount of Numbers')
plt.ylabel('Amount of Prime Numbers')
plt.title('Count of Prime Numbers in the Gaussian Integers')
plt.show()
Listing A.3: Program that calculates the proportion of prime numbers

```python
#Program to calculate the proportion of prime numbers
#out of the total amount of numbers in the set of Gaussian Integers
#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math

def main():
    #array to store the percentage of prime numbers
    #and the amount of total numbers
    proportionData = list()
    numberData = list()

    numberCount = 0
    normMax = 1

    #loop through max norm values while the total amount of numbers is less than 5184
    while numberCount <= 5184:
        norm = 0
        maxA = int(sqrt((normMax)))

        #start the total count of prime numbers at 0
        primeCount = 0

        #start the total count of numbers at 0
        numberCount = 0
        proportion = 0

        #for each "a" value from 1 to max A value
        #loop through all possible "b" values up to given max
        for a in range(1, (maxA + 1)):
            b = 0
            norm = (a * a) + (b * b)
            aMod = a % 4
```
if isprime(a) and aMod == 3:
    #add one to count for this prime
    primeCount += 1
while norm <= normMax:
    numberCount += 1
    if isprime(norm):
        #add one to count for this prime
        primeCount = primeCount + 1
        b += 1
        norm = (a * a) + (b * b)

#find the proportion
proportion = primeCount / numberCount

#add this amount of numbers and the proportion value to an array
#so we can graph the output after
numberData.append(numberCount)
proportionData.append(proportion)
normMax += 1

#plot the values for the amount of numbers on the x axis
#and proportion on the y axis
plt.plot(numberData, proportionData)
plt.xlabel('Amount of Numbers')
plt.ylabel('Proportion of Prime Numbers out of the Total Amount of Numbers')
plt.title('Proportion of Prime Numbers in the Gaussian Integers')
plt.show()
Listing A.4: Program that calculates the proportion of prime numbers with the integers

```python
#Program to calculate the proportion of prime numbers
#out of the total amount of numbers in the set of Gaussian Integers
#and compare that to the integers

#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math

def main():
    #array to store the percentage of prime numbers
    #and the amount of total numbers
    proportionData = list()
    numberData = list()

    numberCount = 0
    normMax = 1

    #loop through max norm values while the total amount of numbers is less than 5184
    while numberCount <= 5184:
        norm = 0
        maxA = int(sqrt((normMax)))

        #start the total count of prime numbers at 0
        primeCount = 0

        #start the total count of numbers at 0
        numberCount = 0
        proportion = 0

        #for each "a" value from 1 to max A value
        #loop through all possible "b" values up to given max
        for a in range(1, (maxA + 1)):
            b = 0
            norm = (a * a) + (b * b)
```

39
aMod = a % 4
if isprime(a) and aMod == 3:
    #add one to count for this prime
    primeCount += 1
while norm <= normMax:
    numberCount += 1
    if isprime(norm):
        #add one to count for this prime
        primeCount = primeCount + 1
    b += 1
    norm = (a * a) + (b * b)

#find the proportion
proportion = primeCount / numberCount

#add this amount of numbers and the proportion value to an array
#so we can graph the output after
numberData.append(numberCount)
proportionData.append(proportion)
normMax += 1

#compute the proportion for integers
proportionDataIntegers = list()
numberDataIntegers = list()

#loop through all integers
for integer in range(1, 5184):
    #start prime count and proportion at 0
    primeCount = 0
    proportion = 0
    #start number at 0
    num = 0
    #loop through all numbers less than the current integer
    #and count the amount of prime numbers
    while num < integer:
        num += 1
        if isprime(num):


40
primeCount += 1

# find the proportion
proportion = primeCount / integer

# add this number amount and the proportion value to an array
# so we can graph the output after
numberDataIntegers.append(integer)
proportionDataIntegers.append(proportion)

# plot the values for the amount of numbers on the x axis
# and proportion on the y axis
# also plot the integers in red dashed line to compare
plt.plot(numberData, proportionData, 'b-', label='Proportion of Gaussian Prime Numbers')
plt.plot(numberDataIntegers, proportionDataIntegers, 'r--', label='Proportion of Prime Numbers in the Integers')
plt.legend(loc='upper right')
plt.xlabel('Amount of Numbers')
plt.ylabel('Proportion of Prime Numbers out of the Total Amount of Numbers')
plt.title('Proportion of Prime Numbers in the Gaussian Integers')
plt.show()
Listing A.5: Program that graphs the prime numbers using Dr. Dekker’s algorithm

```python
# Program that graphs the prime numbers in the Gaussian Integers
# using algorithm by Theodorus J. Dekker
# Author: Jamie Nelson
# Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy

def quadchar(d, n):
    # this function takes a number n
    # and returns \((d / n)\)
    # the quadratic residue

    # skip even numbers
    if (n % 2) == 0:
        return -1

    quadchar = jacobi_symbol(d, n)
    return quadchar

def main():
    # the max norm value is 72 x 72 (5184)
    # to try and match the graphs by Theodorus J. Dekker
    normMax = 5184

    avalues = list()
    bvalues = list()

    # in this case the radicand is \(-1\)
    r = -1

    # the discriminant is equal to \(4r\) for \(r = 3 \pmod{4}\)
    d = 4*r

    # Start with a set \(S\) of the natural numbers \(2 \leq n \leq \text{normMax}\)
    # where \(n\) is a (prime) divisor of \(d\)
```

42
```python
# or quadchar(n) = 1

setS = list()
for n in range(2, (normMax + 1)):
    if (d % n) == 0 and isprime(n):
        setS.append(n)
    elif quadchar(d, n) == 1:
        setS.append(n)

# make a copy of S called T
setT = list()
for s in setS:
    setT.append(s)

# for every element in T
# take the smallest number, t
# such that quadchar(t) = 1
# remove elements in T which are products of t and some element in S
# complete when next element is larger than squareroot of normMax

i = 0
while t <= sqrt(normMax) and i < len(setT):
    t = setT[i]
    if quadchar(d, t) == 1:
        for s in setS:
            product = t * s
            for value in setT:
                if value == product:
                    setT.remove(value)
            i = i + 1

# setT contains all the norms of prime numbers

# loop through all the numbers and if the norm is in setT then
# that number is prime

norm = 0
maxA = int(sqrt((normMax)))

# for each "a" value from 1 to max A value
```
#loop through all possible "b" values up to given max norm
for a in range(1, (maxA + 1)):
    b = 0
    norm = abs((a * a) - (r * (b * b)))
    while norm <= normMax:
        for normsInT in setT:
            if norm == normsInT:
                #if the number is a norm in setT
                #that number is prime
                #add the value to graph
                avalues.append(a)
                bvalues.append(b)

                avalues.append(-a)
                bvalues.append(b)

                avalues.append(-a)
                bvalues.append(-b)

                avalues.append(a)
                bvalues.append(-b)

        b += 1
        norm = abs((a * a) - (r * (b * b)))

#graph all the prime numbers
plt.scatter(avalues, bvalues, s=9)
plt.show()

main()
Listing A.6: Program that compares the count of prime numbers using two different algorithms

```python
#Program to compare the count of prime numbers in the set of Gaussian Integers
#using the Gaussian Prime Theorem
#and using the algorithm by Dr. Theodorus J. Dekker
#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy

def quadchar(d, n):
    #this function takes a number n
    #and returns (d / n)
    #the quadratic residue
    #skip even numbers
    if (n % 2) == 0:
        return -1
    quadchar = jacobi_symbol(d, n)
    return quadchar

def main():
    #array to store the count and amount of numbers pairs in Dekker’s algorithm
    countDataDA = list()
    numberDataDA = list()

    #array to store the count and amount of numbers pairs in Gaussian Prime Theorem
    countDataGPT = list()
    numberDataGPT = list()

    #in this case our radicand is -1
    r = -1

    #the discriminant is equal to 4r for r = 3 (mod 4)
```
d = 4*r

setS = list()
setT = list()

numberCount = 0
normMax = 1

#loop through max norm values using Dekker's algorithm
#while the total amount of numbers is less than 5184
while numberCount <= 5184:

    #Start with a set S of the natural numbers 2 <= n <= normMax
    #where n is a (prime) divisor of d
    #or quadchar(n) = 1

    setS.clear()
    setT.clear()

    for n in range(2, (normMax + 1)):
        if (d % n) == 0 and isprime(n):
            setS.append(n)
        elif quadchar(d, n) == 1:
            setS.append(n)

    #make a copy of S called T
    for s in setS:
        setT.append(s)

    #for every element in T
    #take the smallest number, t
    #such that quadchar(t) = 1
    #remove elements in T which are products of t and some element in S
    #complete when next element is larger than squareroot of normMax
    t = 0
    i = 0
    while (t <= sqrt(normMax)) and (i < len(setT)):
        t = setT[i]
        if quadchar(d, t) == 1:
            for s in setS:
product = t * s
for value in setT:
    if value == product:
        setT.remove(value)
    i = i + 1

#setT contains all the norms of prime numbers

#loop through all the numbers and if the norm is in setT then
#that number is prime
norm = 0
maxA = int(sqrt((normMax)))

#start the total count of prime numbers at 0
primeCount = 0
numberCount = 0

#for each "a" value from 1 to max A value
#loop through all possible "b" values up to given max
for a in range(1, (maxA + 1)):
    b = 0
    norm = abs((a * a) - (r * (b * b)))
    while norm <= normMax:
        numberCount += 1
        for normsInT in setT:
            if norm == normsInT:
                primeCount = primeCount + 1
        b += 1
        norm = abs((a * a) - (r * (b * b)))

numberDataDA.append(numberCount)
countDataDA.append(primeCount)
normMax += 1

numberCount = 0
normMax = 1

#loop through max norm values using Gaussian Prime Theorem
#while the total amount of number is less than 5184
while numberCount <= 5184:

    norm = 0
    maxA = int(sqrt((normMax)))

    #start the total count of prime numbers at 0
    primeCount = 0
    numberCount = 0

    #for each "a" value from 1 to max A value
    #loop through all possible "b" values up to given max
    for a in range(1, (maxA + 1)):
        b = 0
        norm = (a * a) + (b * b)
        aMod = a % 4
        if isprime(a) and aMod == 3:
            #add one to count for this prime
            primeCount += 1
        while norm <= normMax:
            numberCount += 1
            if isprime(norm):
                #add one to count for this prime
                primeCount += 1
            b += 1
            norm = (a * a) + (b * b)

    #add this number amount and the prime count value to an array
    #so we can graph the output after
    numberDataGPT.append(numberCount)
    countDataGPT.append(primeCount)
    normMax += 1

    #plot the values for the amount of numbers on the x axis
    #and the prime number count on the y axis
    #Gaussian Prime Theorem in red dashed line to compare
    plt.plot(numberDataDA, countDataDA, 'b-',
             label='Count of Prime Numbers using the Dr. Dekker\'s Algorithm')
    plt.plot(numberDataGPT, countDataGPT, 'r--',
             label='Count of Prime Numbers using the Gaussian Prime Theorem')
    plt.legend(loc='lower right')

48
plt.xlabel('Total Amount of Numbers')
plt.ylabel('Amount of Prime Numbers')
plt.title('Comparison of the Count of Prime Numbers in the Gaussian Integers using Different Algorithms')
plt.show()
A.2 Code for $\mathbb{Z}[\sqrt{11}]$

Listing A.7: Program that graphs the prime numbers using Dr. Dekker’s algorithm

```python
#Program that graphs the prime numbers in the set $\mathbb{Z}[\sqrt{11}]$
#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy

def quadchar(n):
    #this function takes a number n
    #and returns ($44 / n$)
    #the quadratic residue
    #skip even numbers
    if (n % 2) == 0:
        return -1
    quadchar = jacobi_symbol(44, n)
    return quadchar

def main():
    #the max norm value is 72 x 72 (5184)
    #to try and match the graphs by Theodorus J. Dekker
    normMax = 5184
    avalues = list()
    bvalues = list()
    #in this case the radicand is 11
    r = 11
    #the discriminant is equal to $4r$ for $r = 3 \mod 4$
    d = 4*r
```
# Start with a set S of the natural numbers 2 \leq n \leq \text{normMax} \\
# where n is a (prime) divisor of d \\
# or \quad \text{quadchar}(n) = 1 \\

```python
setS = list() 
for n in range(2, (normMax + 1)):
    if (d % n) == 0 and isprime(n):
        setS.append(n)
    elif quadchar(n) == 1:
        setS.append(n)
```

# make a copy of S called T 
setT = list() 
for s in setS:
    setT.append(s)

# for every element in T 
# take the smallest number, t 
# such that \text{quadchar}(t) = 1 
# remove elements in T which are products of t and some element in S 
# complete when next element is larger than \sqrt{\text{normMax}}

t = 0
i = 0
while t \leq \sqrt{\text{normMax}} and i < len(setT):
    t = setT[i]
    if quadchar(t) == 1:
        for s in setS:
            product = t * s
            for value in setT:
                if value == product:
                    setT.remove(value)
    i = i + 1

# setT contains all the norms of prime numbers

# loop through all the numbers and if the norm is in setT then 
# that number is prime

norm = 0
maxA = int(\sqrt{(\text{normMax})})
#for each "a" value from 1 to max A value
#loop through all possible "b" values up to given max norm
for a in range(1, (maxA + 1)):
    b = 0
    norm = abs((a * a) - (11 * (b * b)))
    while norm <= normMax:
        for normsInT in setT:
            if norm == normsInT:
                #if the number is a norm in setT
                #that number is prime
                #add the value to graph
                avalues.append(a)
                bvalues.append(b)

                avalues.append(-a)
                bvalues.append(b)

                avalues.append(-a)
                bvalues.append(-b)

                avalues.append(a)
                bvalues.append(-b)

        b += 1
        norm = abs((a * a) - (11 * (b * b)))

    #graph all the prime numbers
    plt.scatter(avalues, bvalues, s=16)
plt.show()
main()
#Program to count the prime numbers in the set $a + b\sqrt{11}$

#using algorithm by Dr. Theodorus J. Dekker

#Author: Jamie Nelson

#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

def quadchar(d, n):
    #this function takes a number n
    #and returns $(44 / n)$
    #the quadratic residue

    #skip even numbers
    if (n % 2) == 0:
        return -1

    quadchar = jacobi_symbol(d, n)
    return quadchar

def main():
    download_dir = "elevenData.csv"

    myFile = open(download_dir, "a")
    #"a" indicates that you’re appending strings to the file
    writer = csv.writer(myFile)

    #in this case our radicand is 11
    r = 11

    #the discriminant is equal to $4r$ for $r \equiv 3 \pmod{4}$
\(d = 4r\)

```python
def main():
    #the norm that gives us the first 5000 values
    normMax = 14885

    #Start with a set S of the natural numbers 2 <= n <= normMax
    #where n is a (prime) divisor of d
    #or quadchar(n) = 1
    for n in range(2, (normMax + 1)):
        if (d \% n) == 0 and isprime(n):
            S.append(n)
        elif quadchar(d, n) == 1:
            S.append(n)

    #make a copy of S called T
    for s in S:
        T.append(s)

    #for every element in T
    #take the smallest number, t
    #such that quadchar(t) = 1
    #remove elements in T which are products of t and some element in S
    #complete when next element is larger than squareroot of normMax
    t = 0
    i = 0
    while t <= \sqrt{\text{normMax}} and i < len(T):
        if quadchar(d, t) == 1:
            for s in S:
                product = t * s
                for value in T:
                    if value == product:
                        T.remove(value)

        i = i + 1

    #setT contains all the norms of prime numbers less then the norm of 14885
```

54
# loop through all the numbers and if the norm is in setT then
# that number is prime

for normMax in range(1, 14885):
    norm = 0
    maxA = int(sqrt((normMax)))

    # start the total count of prime numbers at 0
    primeCount = 0
    numberCount = 0

    # for each "a" value from 1 to max A value
    # loop through all possible "b" values up to given max
    for a in range(1, (maxA + 1)):
        b = 0
        norm = abs((a * a) - (r * (b * b)))
        while norm <= normMax:
            numberCount += 1
            for normsInT in setT:
                if norm == normsInT:
                    # add one to count for this prime
                    primeCount = primeCount + 1
                    b += 1
                    norm = abs((a * a) - (r * (b * b)))

    # add this amount of numbers
    # and the prime number count to a csv file
    # so we can graph the output after
    row = [numberCount] + [primeCount]
    writer.writerow(row)

myFile.close()

main()
# Program to graph the prime numbers in the set \( a + b \sqrt{11} \)
using algorithm by Dr. Theodorus J. Dekker
from the values stored in a csv file

Author: Jamie Nelson
Mentor: Jackie Anderson

---

```python
from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

numberData = list()
countData = list()

def download_file():
    download_dir = "elevenData.csv"
    myFile = open(download_dir, "r")
    reader = csv.reader(myFile)
    for row in reader:
        numberData.append(int(row[0]))
        countData.append(int(row[1]))
    myFile.close()

plt.plot(numberData, countData, 'b-')
plt.xlabel('Total Amount of Numbers')
plt.ylabel('Amount of Prime Numbers')
plt.title('Count of Prime Numbers in the set \( a + b \sqrt{11} \)')
plt.show()
```

---

56
Listing A.10: Program that graphs an asymptotic function of the count of prime numbers

```python
from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

def _list():
    numberData = list()
    countData = list()

    #csv file with values
    download_dir = "elevenData.csv"

    myFile = open(download_dir, "r")
    "r" indicates that you’re reading strings from the file

    reader = csv.reader(myFile)
    for row in reader:
        numberData.append(int(row[0]))
        countData.append(int(row[1]))

    myFile.close()

#array to store the 6x/5ln(x) function
xvalues = list()
yvalues = list()

#compute the values for 6x/5ln(x)
for x in range(2, 5184):
    xvalues.append(x)
    y = (6*x) / (5 * math.log(x))
yvalues.append(y)
```

57
# plot the values for the amount of numbers on the x axis
# and the prime number count on the y axis
# also plot 6x/5ln(x) in red dashed lines to compare
plt.plot(numberData, countData, 'b-', label='Count of Prime Numbers')
plt.plot(xvalues, yvalues, 'r--', label=r'$y = \frac{6x}{5\ln(x)}$')
plt.legend(loc='lower right')
plt.xlabel('Total Amount of Numbers')
plt.ylabel('Amount of Prime Numbers')
plt.title('Count of Prime Numbers in the set $a + b + 11$')
plt.show()}
Listing A.11: Program that calculates the proportion of prime numbers

```python
#Program to calculate the proportion of prime numbers
#out of the total amount of numbers in the set a + b\sqrt{11}
#and stores the values in a csv file
#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

#skip even numbers
if (n % 2) == 0:
    return -1

quadchar = jacobi_symbol(d, n)
return quadchar

def main():
    #store values in csv file for future use
    download_dir = "elevenProportionData.csv"

    myFile = open(download_dir, "a")
    #"a" indicates that you're appending strings to the file
    writer = csv.writer(myFile)

    #in this case our radicand is 11
    r = 11

    #the discriminant is equal to 4r for r = 3 (mod 4)
```
\[ d = 4r \]

```python
setS = list()
setT = list()

#the norm that gives us the first 5000 values
normMax = 14885

#Start with a set S of the natural numbers 2 <= n <= normMax
#where n is a (prime) divisor of d
#or quadchar(n) = 1
for n in range(2, (normMax + 1)):
    if (d % n) == 0 and isprime(n):
        setS.append(n)
    elif quadchar(d, n) == 1:
        setS.append(n)

#make a copy of S called T
for s in setS:
    setT.append(s)

#for every element in T
#take the smallest number, t
#such that quadchar(t) = 1
#remove elements in T which are products of t and some element in S
#complete when next element is larger than squareroot of normMax
t = 0
i = 0
while t <= sqrt(normMax) and i < len(setT):
    t = setT[i]
    if quadchar(d, t) == 1:
        for s in setS:
            product = t * s
            for value in setT:
                if value == product:
                    setT.remove(value)
        i = i + 1

#setT contains all the norms of prime numbers less then the norm of 14885
```
#loop through all the numbers and if the norm is in setT then
#that number is prime
for normMax in range(1, 14885):
    norm = 0
    maxA = int(sqrt((normMax)))

    #start the total count of prime numbers at 0
    primeCount = 0

    #start the total count of numbers at 0
    numberCount = 0
    proportion = 0

    #for each "a" value from 1 to max A value
    #loop through all possible "b" values up to given max
    for a in range(1, (maxA + 1)):
        b = 0
        norm = abs((a*a) - (r*(b*b)))
        while norm <= normMax:
            numberCount += 1
            for normsInT in setT:
                if norm == normsInT:
                    #add one to count for this prime
                    primeCount = primeCount + 1
                    b += 1
                    norm = abs((a*a) - (r*(b*b)))

    #calculate the proportion
    proportion = primeCount / numberCount

    #add this amount of numbers and the proportion value to a csv file
    #so we can graph the output after
    row = [numberCount] + [proportion]
    writer.writerow(row)

    myFile.close()
Listing A.12: Program that graphs the proportion of prime numbers

```python
# Program to graph the proportion of prime numbers in the set a + b[sqrt(11)]
# using algorithm by Dr. Theodorus J. Dekker
# from the values stored in a csv file
# Author: Jamie Nelson
# Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

# array to store the percentage of prime numbers and the amount of numbers pairs
proportionData = list()
numberData = list()

# csv file with values
download_dir = "elevenProportionData.csv"

myFile = open(download_dir, "r")
# "r" indicates that you’re reading strings from the file
reader = csv.reader(myFile)
for row in reader:
    numberData.append(int(row[0]))
    proportionData.append(float(row[1]))

myFile.close()

# plot the values for the amount of numbers on the x axis
# and the proportion on the y axis
plt.plot(numberData, proportionData, 'b-')
plt.xlabel('Total Amount of Numbers')
plt.ylabel('Proportion of Prime Numbers out of the Total Amount of Numbers')
plt.title('Proportion of Prime Numbers in the set a + b\u221a11')
plt.show()
```
A.3 Code for $\mathbb{Z}[\sqrt{15}]$

Listing A.13: Program that graphs the prime numbers using Dr. Dekker’s algorithm

```python
from sympy import *
import matplotlib.pyplot as plt
import numpy

def quadchar(n):
    # this function takes a number n
    # and returns (d / n) or (60 / n)
    # the quadratic residue

    # skip even numbers
    if (n % 2) == 0:
        return -1

    quadchar = jacobi_symbol(60, n)
    return quadchar

def main():
    # the max norm value is 72 x 72 (5184)
    # to try and match the graphs by Theodorus J. Dekker
    normMax = 5184

    avalues = list()
    bvalues = list()

    # in this case the radicand is 15
    r = 15

    # the discriminant is equal to 4r for r = 3 (mod 4)
    d = 4*r
```

63
# Start with a set $S$ of the natural numbers $2 \leq n \leq \text{normMax}$
# where $n$ is a (prime) divisor of $d$
# or $\text{quadchar}(n) = 1$

```python
setS = list()
for n in range(2, (normMax + 1)):
    if (d % n) == 0 and isprime(n):
        setS.append(n)
    elif quadchar(n) == 1:
        setS.append(n)

# make a copy of $S$ called $T$
setT = list()
for s in setS:
    setT.append(s)

# for every element in $T$
# take the smallest number, $t$
# such that $\text{quadchar}(t) = 1$
# remove elements in $T$ which are products of $t$ and some element in $S$
# complete when next element is larger than square root of normMax

# set $T$ contains all the norms of prime numbers
```

```python
norm = 0
maxA = int(sqrt((normMax)))
```
for each "a" value from 1 to max A value
    #loop through all possible "b" values up to given max norm
    for a in range(1, (maxA + 1)):
        b = 0
        norm = abs((a * a) - (r * (b * b)))
        while norm <= normMax:
            for normsInT in setT:
                if norm == normsInT:
                    #if the number is a norm in setT
                    #that number is prime
                    #add the value to graph
                    avalues.append(a)
                    bvalues.append(b)
                    avalues.append(-a)
                    bvalues.append(b)
                    avalues.append(-a)
                    bvalues.append(-b)
                    avalues.append(a)
                    bvalues.append(-b)
                    b += 1
                    norm = abs((a * a) - (r * (b * b)))
        #graph all the prime numbers
        plt.scatter(avalues, bvalues, s=16)
        plt.show()
main()
Listing A.14: Program that counts the prime numbers

```python
# Program to count the prime numbers in the set a + b[\sqrt{15}] using algorithm by Theodorus J. Dekker
# and stores the values in a csv file

# Author: Jamie Nelson
# Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

def quadchar(d, n):
    # this function takes a number n and returns (60/n)
    # the quadratic residue

    # skip even numbers
    if (n % 2) == 0:
        return -1
    quadchar = jacobi_symbol(d, n)
    return quadchar

def main():
    # store values in csv file for future use
    download_dir = "fifteenData.csv"

    myFile = open(download_dir, "a")
    # "a" indicates that you’re appending strings to the file
    writer = csv.writer(myFile)

    # in this case our radicand is 15
    r = 15

    # the discriminant is equal to 4r for r = 3 (mod 4)
```

66
```python
d = 4*r

setS = list()
setT = list()

#the norm that gives us the first 5000 values
normMax = 17340

#Start with a set S of the natural numbers 2 <= n <= normMax
#where n is a (prime) divisor of d
#or quadchar(n) = 1

for n in range(2, (normMax + 1)):
    if (d % n) == 0 and isprime(n):
        setS.append(n)
    elif quadchar(d, n) == 1:
        setS.append(n)

#make a copy of S called T
for s in setS:
    setT.append(s)

#for every element in T
#take the smallest number, t
#such that quadchar(t) = 1
#remove elements in T which are products of t and some element in S
#complete when next element is larger than squareroot of normMax

for i in range(2, int(sqrt(normMax))):
    for s in setS:
        if quadchar(t, s) == 1:
            for value in setT:
                if value == product:
                    setT.remove(value)

#setT contains all the norms of prime numbers less than the norm of 17340
```

67
# loop through all the numbers and if the norm is in setT then that number is prime
for normMax in range(1, 17340):
    norm = 0
    maxA = int(sqrt((normMax)))

    # start the total count of primes at 0
    primeCount = 0
    numberCount = 0

    # for each "a" value from 1 to max A value
    # loop through all possible "b" values up to given max
    for a in range(1, (maxA + 1)):
        b = 0
        norm = abs((a * a) - (r * (b * b)))
        while norm <= normMax:
            numberCount += 1
            for normsInT in setT:
                if norm == normsInT:
                    # add one to count for this prime
                    primeCount = primeCount + 1
                    b += 1
                    norm = abs((a * a) - (r * (b * b)))

    # add this amount of numbers
    # and the prime number count to a csv file
    # so we can graph the output after
    row = [numberCount] + [primeCount]
    writer.writerow(row)

myFile.close()
Listing A.15: Program that graphs the count of the prime numbers

```python
#Program to graph the prime numbers in the set a + b[\sqrt{15}] 
#using algorithm by Dr. Theodorus J. Dekker
#from the values stored in a csv file
#Author: Jamie Nelson
#Mentor: Jackie Anderson

def count_prime_numbers(a, b):
    # Count the prime numbers in the set a + b[\sqrt{15}]
    # and store them in list countData
    # import the necessary libraries
    from sympy import *
    import matplotlib.pyplot as plt
    import numpy
    import math
    import csv

    numberData = list()
    countData = list()

    # csv file with values
    download_dir = "fifteenData.csv"
    myFile = open(download_dir, "r")
    # "r" indicates that you're reading strings from the file
    reader = csv.reader(myFile)
    for row in reader:
        numberData.append(int(row[0]))
        countData.append(int(row[1]))
    myFile.close()

    # plot the values for the amount of numbers on the x axis
    # and the prime number count on the y axis
    plt.plot(numberData, countData, 'b-')
    plt.xlabel('Total Amount of Numbers')
    plt.ylabel('Amount of Prime Numbers')
    plt.title('Count of Prime Numbers in the set a + b[\sqrt{15}]')
    plt.show()
```
#Program to graph the prime numbers in the set a + b[\sqrt(15)]
#using algorithm by Dr. Theodorus J. Dekker
#from the values stored in a csv file
#and graphs an asymptotic function
#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

numberData = list()
countData = list()

def read_csv_file(download_dir):
    myFile = open(download_dir, "r")
    reader = csv.reader(myFile)
    for row in reader:
        numberData.append(int(row[0]))
        countData.append(int(row[1]))
    myFile.close()

def array_to_store_97x/100ln(x) function:
xvalues = list()
yvalues = list()

for x in range(2, 5184):
xvalues.append(x)
y = (97 * x) / (100 * math.log(x))
yvalues.append(y)
#plot the values for the amount of numbers on the x axis
#and the prime number count on the y axis
#also plot $97x/100\ln(x)$ in red dashed lines to compare
plt.plot(numberData, countData, 'b-', label='Count of Prime Numbers')
plt.plot(xvalues, yvalues, 'r--', label='\( y = \frac{97x}{100\ln(x)} \)')
plt.legend(loc='lower right')
plt.xlabel('Total Amount of Numbers')
plt.ylabel('Amount of Prime Numbers')
plt.title('Count of Prime Numbers in the set a + b + u^{221}\) + 15")
plt.show()}
Listing A.17: Program that calculates the proportion of prime numbers

```python
#Program to calculate the proportion of prime numbers
#out of the total amount of numbers in the set a + b\sqrt{15}
#and stores the values in a csv file
#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

def quadchar(d, n):
    # this function takes a number n
    # and returns (d / n)
    # the quadratic residue

    # skip even numbers
    if (n % 2) == 0:
        return -1

    quadchar = jacobi_symbol(d, n)
    return quadchar

def main():
    # store values in csv file for future use
    download_dir = "fifteenProportionData.csv"

    myFile = open(download_dir, "a")
    # "a" indicates that you're appending strings to the file
    writer = csv.writer(myFile)

    # in this case our radicand is 15
    r = 15

    # the discriminant is equal to 4r for r = 3 (mod 4)
```
\[ d = 4r \]
\[ \text{setS} = \text{list()} \]
\[ \text{setT} = \text{list()} \]

#the norm that gives us the first 5000 values

\[ \text{normMax} = 17340 \]

#Start with a set S of the natural numbers 2 \(\leq n \leq \text{normMax} \)
#where n is a (prime) divisor of d
#or quadchar(n) = 1

\[
\text{for n in range}(2, (\text{normMax} + 1)):\]
\[ \text{if } (d \% n) == 0 \text{ and isprime(n)}:\]
\[ \text{setS}.\text{append}(n) \]
\[ \text{elif quadchar}(d, n) == 1: \]
\[ \text{setS}.\text{append}(n) \]

#make a copy of S called T

\[
\text{for s in setS}: \\
\text{setT}.\text{append}(s) \\
\]

#for every element in T
#take the smallest number, t
#such that quadchar(t) = 1
#remove elements in T which are products of t and some element in S
#complete when next element is larger than squareroot of normMax

\[ t = 0 \]
\[ i = 0 \]

\[
\text{while } t \leq \text{sqrt(normMax)} \text{ and } i < \text{len(setT)}: \\
\text{t = setT}[i] \\
\text{if quadchar}(d, t) == 1: \\
\text{for s in setS}: \\
\text{product} = t \ast s \\
\text{for value in setT}: \\
\text{if value == product}: \\
\text{setT}.\text{remove}(value) \\
\text{i} = i + 1 \\
\]

#setT contains all the norms of prime numbers less then the norm of 17340

73
#loop through all the numbers and if the norm is in setT then
#that number is prime
for normMax in range(1, 17340):
    norm = 0
    maxA = int(sqrt((normMax)))
    #start the total count of prime numbers at 0
    primeCount = 0
    #start the total count of numbers at 0
    numberCount = 0
    proportion = 0
    #for each "a" value from 1 to max A value
    #loop through all possible "b" values up to given max
    for a in range(1, (maxA + 1)):
        b = 0
        norm = abs((a * a) - (r * (b * b)))
        while norm <= normMax:
            numberCount += 1
            for normsInT in setT:
                if norm == normsInT:
                    #add one to count for this prime
                    primeCount = primeCount + 1
                    b += 1
                    norm = abs((a * a) - (r * (b * b)))
                break
            #calculate the proportion
            proportion = primeCount / numberCount
        #add this amount of numbers and the proportion value to a csv file
        #so we can graph the output after
        row = [numberCount] + [proportion]
        writer.writerow(row)
myFile.close()
# Program to graph the proportion of prime numbers in the set \( a + b \sqrt{15} \)

using algorithm by Dr. Theodorus J. Dekker

from the values stored in a csv file

Author: Jamie Nelson

Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

# array to store the percentage of prime numbers and the amount of numbers pairs
proportionData = list()
numberData = list()

csv file with values
download_dir = "fifteenProportionData.csv"

myFile = open(download_dir, "r")
"r" indicates that you're reading strings from the file

reader = csv.reader(myFile)
for row in reader:
    numberData.append(int(row[0]))
    proportionData.append(float(row[1]))

myFile.close()

# plot the values for the amount of numbers on the x axis
# and the proportion on the y axis
plt.plot(numberData, proportionData, 'b-')
plt.xlabel('Total Amount of Numbers')
plt.ylabel('Proportion of Prime Numbers out of the Total Amount of Numbers')
plt.title('Proportion of Prime Numbers in the set \( a + b \sqrt{15} \)')
plt.show()
A.4 Code for $\mathbb{Z}[\sqrt{34}]$

Listing A.19: Program that graphs the prime numbers using Dr. Dekker’s algorithm

```python
#Program that graphs the prime numbers in the set $\mathbb{Z}[\sqrt{34}]$
#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy

def quadchar(d, n):
    # this function takes a number n
    # and returns (d / n)
    # the quadratic residue

    # skip even numbers
    if (n % 2) == 0:
        return -1

    quadchar = jacobi_symbol(d, n)
    return quadchar

def main():
    # the max norm value is 72 x 72 (5184)
    # to try and match the graphs by Theodorus J. Dekker
    normMax = 5184

    avalues = list()
bvalues = list()

    # in this case our radicand is 34
    r = 34

    # the discriminant is equal to 4r for r = 2 (mod 4)
    d = 4*r
```

76
#Start with a set S of the natural numbers 2 ≤ n ≤ normMax
#where n is a (prime) divisor of d
#or quadchar(n) = 1

setS = list()
for n in range(2, (normMax + 1)):
    if (d % n) == 0 and isprime(n):
        setS.append(n)
    elif quadchar(d, n) == 1:
        setS.append(n)

#make a copy of S called T
setT = list()
for s in setS:
    setT.append(s)

#for every element in T
#take the smallest number, t
#such that quadchar(t) = 1
#remove elements in T which are products of t and some element in S
#complete when next element is larger than squareroot of normMax

maxA = int(sqrt((normMax)))

norm = 0
while t <= sqrt(normMax) and i < len(setT):
    t = setT[i]
    if quadchar(d, t) == 1:
        for s in setS:
            product = t * s
            for value in setT:
                if value == product:
                    setT.remove(value)
                i = i + 1

#setT contains all the norms of prime numbers

#loop through all the numbers and if the norm is in setT then
#that number is prime

norm = 0
maxA = int(sqrt((normMax)))
# for each "a" value from 1 to max A value
# loop through all possible "b" values up to given max
for a in range(1, (maxA + 1)):
    b = 0
    norm = abs((a * a) - (r * (b * b)))
    while norm <= normMax:
        for normsInT in setT:
            if norm == normsInT:
                # if the number is a norm in setT
                # that number is prime
                # add the value to graph
                avalues.append(a)
bvalues.append(b)

                avalues.append(-a)
bvalues.append(b)

                avalues.append(-a)
bvalues.append(-b)

                avalues.append(a)
bvalues.append(-b)

        b += 1
        norm = abs((a * a) - (r * (b * b)))

    # graph all the prime numbers
plt.scatter(avalues, bvalues, s=16)
plt.show()

main()
Listing A.20: Program that counts the prime numbers

1 #Program to count the prime numbers in the set \( a + b \sqrt{34} \)
2 #using algorithm by Theodorus J. Dekker
3 #and stores the values in a csv file
4 #Author: Jamie Nelson
5 #Mentor: Jackie Anderson

6 from sympy import *
7 import matplotlib.pyplot as plt
8 import numpy
9 import math
10 import csv

13 def quadchar(d, n):
14    #this function takes a number n
15    #and returns \( \left( \frac{d}{n} \right) \)
16    #the quadratic residue
17
18    #skip even numbers
19    if (n % 2) == 0:
20        return -1
21    quadchar = jacobi_symbol(d, n)
22    return quadchar

26 def main():
27    #store values in csv file for future use
28    download_dir = "thirtyfourData.csv"
29
30    myFile = open(download_dir, "a")
31    #"a" indicates that you're appending strings to the file
32    writer = csv.writer(myFile)
33
34    #in this case our radicand is 34
35    r = 34
36
37    #the discriminant is equal to 4r for \( r \equiv 2 \pmod{4} \)
40 \quad d = 4r

41 \quad \text{setS} = \text{list}()
42 \quad \text{setT} = \text{list}()

43 \quad \#\text{the norm that gives us the first 5000 values}
44 \quad \text{normMax} = 25922

45 \quad \#\text{Start with a set S of the natural numbers } 2 \leq n \leq \text{normMax}
46 \quad \#\text{where } n \text{ is a (prime) divisor of } d
47 \quad \#\text{or quadchar}(n) = 1

48 \quad \text{for } n \text{ in range}(2, (\text{normMax} + 1)):
49 \quad \quad \text{if } (d \% n) == 0 \text{ and isprime}(n):
50 \quad \quad \quad \text{setS}.append(n)
51 \quad \quad \text{elif quadchar}(d, n) == 1:
52 \quad \quad \quad \text{setS}.append(n)

53 \quad \text{#make a copy of S called T}
54 \quad \text{for } s \text{ in setS}:
55 \quad \quad \text{setT}.append(s)

56 \quad \text{#for every element in T}
57 \quad \text{#take the smallest number, } t
58 \quad \text{#such that quadchar}(t) = 1
59 \quad \text{#remove elements in T which are products of } t \text{ and some element in } S
60 \quad \text{#complete when next element is larger than square root of normMax}
61 \quad t = 0
62 \quad i = 0
63 \quad \text{while } t \leq \text{sqrt}(\text{normMax}) \text{ and } i < \text{len}(\text{setT}):
64 \quad \quad t = \text{setT}[i]
65 \quad \quad \text{if quadchar}(d, t) == 1:
66 \quad \quad \quad \text{for } s \text{ in setS}:
67 \quad \quad \quad \quad \text{product} = t * s
68 \quad \quad \quad \quad \text{for } value \text{ in setT}:
69 \quad \quad \quad \quad \quad \text{if } value == \text{product}:
70 \quad \quad \quad \quad \quad \quad \text{setT}.remove(value)
71 \quad \quad \quad i = i + 1

72 \quad \text{#setT contains all the norms of prime numbers less than the norm of 25922}
#loop through all the numbers and if the norm is in setT then
#that number is prime
for normMax in range(1, 25922):
    norm = 0
    maxA = int(sqrt((normMax)))

    #start the total count of primes at 0
    primeCount = 0
    numberCount = 0

    #for each "a" value from 1 to max A value
    #loop through all possible "b" values up to given max
    for a in range(1, (maxA + 1)):
        b = 0
        norm = abs((a * a) - (r * (b * b)))
        while norm <= normMax:
            numberCount += 1
            for normsInT in setT:
                #add one to count for this prime
                primeCount = primeCount + 1
            b += 1
            norm = abs((a * a) - (r * (b * b)))

    #add this amount of numbers
    #and the prime number count to a csv file
    #so we can graph the output after
    row = [numberCount] + [primeCount]
    writer.writerow(row)

myFile.close()
Listing A.21: Program that graphs the count of the prime numbers

```python
#Program to graph the prime numbers in the set a + b[sqrt(34)]
#using algorithm by Dr. Theodorus J. Dekker
#from the values stored in a csv file
#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

numberData = list()
countData = list()

#csv file with values
download_dir = "thirtyfourData.csv"

myFile = open(download_dir, "r")
#"r" indicates that you're reading strings from the file

reader = csv.reader(myFile)
for row in reader:
    numberData.append(int(row[0]))
    countData.append(int(row[1]))

myFile.close()

#plot the values for the amount of numbers on the x axis
#and the prime number count on the y axis
plt.plot(numberData, countData, 'b−')
plt.xlabel("Total Amount of Numbers")
plt.ylabel("Amount of Prime Numbers")
plt.title("Count of Prime Numbers in the set a + b + \sqrt{34}\)"
plt.show()
```
Listing A.22: Program that graphs an asymptotic function of the count of prime numbers

```python
from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

numberData = list()
countData = list()

download_dir = "thirtyfourData.csv"

myFile = open(download_dir, "r")
"r" indicates that you're reading strings from the file

reader = csv.reader(myFile)
for row in reader:
    numberData.append(int(row[0]))
    countData.append(int(row[1]))

myFile.close()

#array to store the 69x/100ln(x) function
xvalues = list()
yvalues = list()

#compute the values for 69x/100ln(x)
for x in range(2,5184):
    xvalues.append(x)
    y = (69*x) / (100 * math.log(x))
    yvalues.append(y)
```

83
#plot the values for the amount of numbers on the x axis
#and the prime number count on the y axis
#also plot 69x/100ln(x) in red dashed lines to compare
plt.plot(numberData, countData, 'b-', label='Count of Prime Numbers')
plt.plot(xvalues, yvalues, 'r--', label=r'$y = \frac{69x}{100\ln(x)}$')
plt.legend(loc='lower right')
plt.xlabel('Total Amount of Numbers')
plt.ylabel('Amount of Prime Numbers')
plt.title('Count of Prime Numbers in the set $a + b + 2^a + 17$')
plt.show()
Listing A.23: Program that calculates the proportion of prime numbers

```python
from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

#Program to calculate the proportion of prime numbers
#out of the total amount of numbers in the set a + b\sqrt(34)
#and stores the values in a csv file

#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

def quadchar(d, n):
    #this function takes a number n
    #and returns (d / n)
    #the quadratic residue

    #skip even numbers
    if (n % 2) == 0:
        return -1

    quadchar = jacobi_symbol(d, n)
    return quadchar

def main():
    #store values in csv file for future use
    download_dir = "thirtyfourProportionData.csv"

    myFile = open(download_dir, "a")
    #"a" indicates that you're appending strings to the file
    writer = csv.writer(myFile)

    #in this case our radicand is 34
    r = 34

    #the discriminant is equal to 4r for r = 2 (mod 4)
```
\[ d = 4r \]

```python
setS = list()
sdT = list()

# the norm that gives us the first 5000 values
normMax = 25922

# Start with a set \( S \) of the natural numbers \( 2 \leq n \leq \text{normMax} \)
# where \( n \) is a \( (\text{prime}) \) divisor of \( d \)
# or \( \text{quadchar}(n) = 1 \)
for n in range(2, (normMax + 1)):
    if (d % n) == 0 and isprime(n):
        setS.append(n)
    elif quadchar(d, n) == 1:
        setS.append(n)

# make a copy of \( S \) called \( T \)
for s in setS:
    setT.append(s)

# for every element in \( T \)
# take the smallest number, \( t \)
# such that \( \text{quadchar}(t) = 1 \)
# remove elements in \( T \) which are products of \( t \) and some element in \( S \)
# complete when next element is larger than \( \sqrt{\text{normMax}} \)
t = 0
i = 0
while t <= sqrt(normMax) and i < len(setT):
    t = setT[i]
    if quadchar(d, t) == 1:
        for s in setS:
            product = t * s
        for value in setT:
            if value == product:
                setT.remove(value)
    i = i + 1

# setT contains all the norms of prime numbers less than the norm of 25922
```
#loop through all the numbers and if the norm is in setT then that number is prime

```python
for normMax in range(1, 25922):
    norm = 0
    maxA = int(sqrt(normMax))

    #start the total count of prime numbers at 0
    primeCount = 0

    #start the total count of numbers at 0
    numberCount = 0
    proportion = 0

    #for each "a" value from 1 to max A value
    #loop through all possible "b" values up to given max
    for a in range(1, (maxA + 1)):
        b = 0
        norm = abs((a * a) - (r * (b * b)))
        while norm <= normMax:
            numberCount += 1
            for normsInT in setT:
                if norm == normsInT:
                    #add one to count for this prime
                    primeCount = primeCount + 1
            b += 1
            norm = abs((a * a) - (r * (b * b)))

    #calculate the proportion
    proportion = primeCount / numberCount

    #add this amount of numbers and the proportion value to a csv file
    #so we can graph the output after
    row = [numberCount] + [proportion]
    writer.writerow(row)
```

myFile.close()

main()
Listing A.24: Program that graphs the proportion of prime numbers

```python
#Program to graph the proportion of prime numbers in the set a + b[sqrt(34)]
#using algorithm by Dr. Theodorus J. Dekker
#from the values stored in a csv file
#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

#array to store the percentage of prime numbers and the amount of numbers pairs
proportionData = list()
numberData = list()

#csv file with values
download_dir = "thirtyfourProportionData.csv"

myFile = open(download_dir, "r")
#"r" indicates that you're reading strings from the file
reader = csv.reader(myFile)
for row in reader:
    numberData.append(int(row[0]))
    proportionData.append(float(row[1]))
myFile.close()

#plot the values for the amount of numbers on the x axis
#and the proportion on the y axis
plt.plot(numberData, proportionData, 'b-')
plt.xlabel('Total Amount of Numbers')
plt.ylabel('Proportion of Prime Numbers out of the Total Amount of Numbers')
plt.title('Proportion of Prime Numbers in the set a + b\u221a34')
plt.show()
```
A.5  Code for $\mathbb{Z}[\sqrt{35}]$

Listing A.25: Program that graphs the prime numbers using Dr. Dekker’s algorithm

```python
#Program that graphs the prime numbers in the set $\mathbb{Z}[\sqrt{35}]$
#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy

def quadchar(d, n):
    #this function takes a number n
    #and returns (d / n)
    #the quadratic residue

    #skip even numbers
    if (n % 2) == 0:
        return -1

    quadchar = jacobi_symbol(d, n)
    return quadchar

def main():
    #the max norm value is 72 x 72 (5184)
    #to try and match the graphs by Theodorus J. Dekker
    normMax = 5184

    avalues = list()
    bvalues = list()

    #in this case our radicand is 35
    r = 35

    #the discriminant is equal to 4r for r = 3 (mod 4)
    d = 4*r
```

89
# Start with a set S of the natural numbers 2 \leq n \leq \text{normMax}
# where n is a (prime) divisor of d
# or quadchar(n) = 1

setS = list()
for n in range(2, (normMax + 1)):
    if (d \% n) == 0 and isprime(n):
        setS.append(n)
    elif quadchar(d, n) == 1:
        setS.append(n)

# make a copy of S called T
setT = list()
for s in setS:
    setT.append(s)

# for every element in T
# take the smallest number, t
# such that quadchar(t) = 1
# remove elements in T which are products of t and some element in S
# complete when next element is larger than squareroot of normMax

i = 0
while t \leq \sqrt{\text{normMax}} and i < len(setT):
    t = setT[i]
    if quadchar(d, t) == 1:
        for s in setS:
            product = t * s
            for value in setT:
                if value == product:
                    setT.remove(value)
        i = i + 1

# setT contains all the norms of prime numbers

# loop through all the numbers and if the norm is in setT then
# that number is prime

norm = 0
maxA = int(\sqrt{\text{normMax}})
# for each "a" value from 1 to max A value
# loop through all possible "b" values up to given max
for a in range(1, (maxA + 1)):
    b = 0
    norm = abs((a * a) - (r * (b * b)))
    while norm <= normMax:
        for normsInT in setT:
            if norm == normsInT:
                # if the number is a norm in setT
                # that number is prime
                # add the value to graph
                avalues.append(a)
                bvalues.append(b)
                avalues.append(-a)
                bvalues.append(b)
                avalues.append(-a)
                bvalues.append(-b)
                avalues.append(a)
                bvalues.append(-b)
        b += 1
        norm = abs((a * a) - (r * (b * b)))

# graph all the prime numbers
plt.scatter(avalues, bvalues)
plt.show()
main()
Listing A.26: Program that counts the prime numbers

```python
# Program to count the prime numbers in the set a + b[sqrt(35)]
# using the algorithm by Theodorus J. Dekker
# and stores the values in a csv file
# Author: Jamie Nelson
# Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

def quadchar(d, n):
    # this function takes a number n
    # and returns (d / n)
    # the quadratic residue

    # skip even numbers
    if (n % 2) == 0:
        return -1

    quadchar = jacob_symbol(d, n)
    return quadchar

def main():
    # store values in csv file for future use
    download_dir = "thirtyfiveData.csv"

    myFile = open(download_dir, "a")
    # "a" indicates that you're appending strings to the file
    writer = csv.writer(myFile)

    # in this case our radicand is 35
    r = 35

    # the discriminant is equal to 4r for r = 3 (mod 4)
```

92
\begin{verbatim}
40 d = 4*r
41
42 setS = list()
setT = list()

#the norm that gives us the first 5000 values
normMax = 26317

#Start with a set S of the natural numbers 2 <= n <= normMax
#where n is a (prime) divisor of d
#or quadchar(n) = 1

for n in range(2, (normMax + 1)):
    if (d % n) == 0 and isprime(n):
        setS.append(n)
    elif quadchar(d, n) == 1:
        setS.append(n)

#make a copy of S called T
for s in setS:
    setT.append(s)

#for every element in T
#take the smallest number, t
#such that quadchar(t) = 1
#remove elements in T which are products of t and some element in S
#complete when next element is larger than squareroot of normMax

t = 0
i = 0

while t <= sqrt(normMax) and i < len(setT):
    t = setT[i]
    if quadchar(d, t) == 1:
        for s in setS:
            product = t * s
        for value in setT:
            if value == product:
                setT.remove(value)
    i = i + 1

#setT contains all the norms of prime numbers less than the norm of 26317
\end{verbatim}
# loop through all the numbers and if the norm is in setT then
# that number is prime
for normMax in range(1, 26317):
    norm = 0
    maxA = int(sqrt(normMax))

    # start the total count of primes at 0
    primeCount = 0
    numberCount = 0

    # for each "a" value from 1 to max A value
    # loop through all possible "b" values up to given max
    for a in range(1, (maxA + 1)):
        b = 0
        norm = abs((a * a) - (r * (b * b)))
        while norm < normMax:
            numberCount += 1
            for normsInT in setT:
                # add one to count for this prime
                if norm == normsInT:
                    # add this amount of numbers
                    # and the prime number count to a csv file
                    # so we can graph the output after
                    row = [numberCount] + [primeCount]
                    writer.writerow(row)
                    primeCount = primeCount + 1
                b += 1
                norm = abs((a * a) - (r * (b * b)))

    # add this amount of numbers
    # and the prime number count to a csv file
    # so we can graph the output after
    row = [numberCount] + [primeCount]
    writer.writerow(row)

    myFile.close()
Listing A.27: Program that graphs the count of the prime numbers

```python
#Program to graph the prime numbers in the set a + b[\sqrt{35}] 
#using algorithm by Dr. Theodorus J. Dekker
#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

numberData = list()
countData = list()

csv file with values
download_dir = "thirtyfiveData.csv"

myFile = open(download_dir, "r")
#"r" indicates that you're reading strings from the file

reader = csv.reader(myFile)
for row in reader:
    numberData.append(int(row[0]))
    countData.append(int(row[1]))

myFile.close()

#plot the values for the amount of numbers on the x axis
#and the prime number count on the y axis
plt.plot(numberData, countData, 'b−')
plt.xlabel('Total Amount of Numbers')
plt.ylabel('Amount of Prime Numbers')
plt.title('Count of Prime Numbers in the set a + b' + u'\"\u221a' + '35')
plt.show()
```
Listing A.28: Program that graphs an asymptotic function of the count of prime numbers

```python
#Program to graph the prime numbers in the set a + b[sqrt(35)]
#using algorithm by Dr. Theodorus J. Dekker
#and graphs an asymptotic function
#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

numberData = list()
countData = list()

#csv file with values
download_dir = "thirtyfiveData.csv"

myFile = open(download_dir, "r")
#"r" indicates that you’re reading strings from the file
reader = csv.reader(myFile)
for row in reader:
    numberData.append(int(row[0]))
    countData.append(int(row[1]))
myFile.close()

#array to store the 29x/25ln(x) function
xvalues = list()
yvalues = list()

#compute the values for 29x/25ln(x)
for x in range(2, 5184):
    xvalues.append(x)
    y = (29*x) / (25 * math.log(x))
yvalues.append(y)
```

96
# plot the values for the amount of numbers on the x axis
# and the prime number count on the y axis
# also plot 29x/25ln(x) in red dashed lines to compare
plt.plot(numberData, countData, 'b-', label='Count of Prime Numbers')
plt.plot(xvalues, yvalues, 'r--', label=r'$y = \frac{29x}{25\ln(x)}$')
plt.legend(loc='lower right')
plt.xlabel('Total Amount of Numbers')
plt.ylabel('Amount of Prime Numbers')
plt.title('Count of Prime Numbers in the set $a + b + u^{221a} + 35$')
plt.show()
# Program to calculate the proportion of prime numbers

# out of the total amount of numbers in the set \( a + b\sqrt{35} \)
# and stores the values in a csv file

# Author: Jamie Nelson
# Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

def quadchar(d, n):
    # this function takes a number \( n \)
    # and returns \( \frac{d}{n} \)
    # the quadratic residue

    # skip even numbers
    if (n % 2) == 0:
        return -1

    quadchar = jacobi_symbol(d, n)
    return quadchar

def main():
    # store values in csv file for future use
    download_dir = "thirtyfiveProportionData.csv"

    myFile = open(download_dir, "a")
    # "a" indicates that you're appending strings to the file
    writer = csv.writer(myFile)

    # in this case our radicand is 35
    r = 35

    # the discriminant is equal to 4r for \( r = 3 \pmod{4} \)
\[ d = 4 \star r \]

```python
d = 4 * r

setS = list()
setT = list()

# the norm that gives us the first 5000 values
normMax = 26317

# Start with a set S of the natural numbers 2 <= n <= normMax
# where n is a (prime) divisor of d
# or quadchar(n) = 1
for n in range(2, (normMax + 1)):
    if (d % n) == 0 and isprime(n):
        setS.append(n)
    elif quadchar(d, n) == 1:
        setS.append(n)

# make a copy of S called T
for s in setS:
    setT.append(s)

# for every element in T
# take the smallest number, t
# such that quadchar(t) = 1
# remove elements in T which are products of t and some element in S
# complete when next element is larger than square root of normMax
i = 0
while t <= sqrt(normMax) and i < len(setT):
    t = setT[i]
    if quadchar(d, t) == 1:
        for s in setS:
            product = t * s
            for value in setT:
                if value == product:
                    setT.remove(value)
    i = i + 1

# setT contains all the norms of prime numbers less than the norm of 26317
```
# loop through all the numbers and if the norm is in setT then that number is prime

for normMax in range(1, 26317):

    norm = 0
    maxA = int(sqrt(normMax))

    # start the total count of prime numbers at 0
    primeCount = 0

    # start the total count of numbers at 0
    numberCount = 0
    proportion = 0

    # for each "a" value from 1 to max A value
    # loop through all possible "b" values up to given max
    for a in range(1, (maxA + 1)):

        b = 0
        norm = abs((a * a) - (r * (b * b)))
        while norm <= normMax:

            numberCount += 1
            for normsInT in setT:

                if norm == normsInT:

                    # add one to count for this prime
                    primeCount = primeCount + 1
                    b += 1
                    norm = abs((a * a) - (r * (b * b)))

            # calculate the proportion
            proportion = primeCount / numberCount

    # add this amount of numbers and the proportion value to a csv file
    # so we can graph the output after
    row = [numberCount] + [proportion]
    writer.writerow(row)

myFile.close()

main()
Listing A.30: Program that graphs the proportion of prime numbers

```python
#Program to graph the proportion of prime numbers in the set a + b[\sqrt{35}] using algorithm by Dr. Theodorus J. Dekker
#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

#array to store the percentage of prime numbers and the amount of numbers pairs
proportionData = list()
numberData = list()

#csv file with values
download_dir = "thirtyfiveProportionData.csv"

myFile = open(download_dir, "r") #"r" indicates that you're reading strings from the file
reader = csv.reader(myFile)
for row in reader:
    numberData.append(int(row[0]))
    proportionData.append(float(row[1]))

myFile.close()

#plot the values for the amount of numbers on the x axis
#and the proportion on the y axis
plt.plot(numberData, proportionData, 'b-')
plt.xlabel('Total Amount of Numbers')
plt.ylabel('Proportion of Prime Numbers out of the Total Amount of Numbers')
plt.title('Proportion of Prime Numbers in the set a + b'+ u"²"+u"35")
plt.show()
```
A.6 Code for $\mathbb{Z}[\sqrt{79}]$

Listing A.31: Program that graphs the prime numbers using Dr. Dekker’s algorithm

```python
#Program that graphs the prime numbers in the set $\mathbb{Z}[\sqrt{79}]$
#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy

def quadchar(d, n):
    #this function takes a number n
    #and returns (d / n)
    #the quadratic residue

    #skip even numbers
    if (n % 2) == 0:
        return -1

    quadchar = jacobi_symbol(d, n)
    return quadchar

def main():
    #the max norm value is 72 x 72 (5184)
    #to try and match the graphs by Theodorus J. Dekker
    normMax = 5184

    avalues = list()
    bvalues = list()

    #in this case our radicand is 79
    r = 79

    #the discriminant is equal to 4r for r = 3 (mod 4)
    d = 4*r
```

102
Start with a set $S$ of the natural numbers $2 \leq n \leq \text{normMax}$

where $n$ is a (prime) divisor of $d$

or quadchar$(n) = 1$

```python
setS = list()
for n in range(2, (normMax + 1)):
    if (d % n) == 0 and isprime(n):
        setS.append(n)
    elif quadchar(d, n) == 1:
        setS.append(n)

# make a copy of S called T
setT = list()
for s in setS:
    setT.append(s)

# for every element in T
# take the smallest number, $t$
# such that quadchar$(t) = 1$
# remove elements in T which are products of $t$ and some element in $S$
# complete when next element is larger than squareroot of normMax

t = 0
i = 0
while t <= sqrt(normMax) and i < len(setT):
    t = setT[i]
    if quadchar(d, t) == 1:
        for s in setS:
            product = t * s
            for value in setT:
                if value == product:
                    setT.remove(value)
    i = i + 1

# setT contains all the norms of prime numbers

# loop through all the numbers and if the norm is in setT then
# that number is prime

norm = 0
maxA = int(sqrt((normMax)))
```
# for each "a" value from 1 to max A value
# loop through all possible "b" values up to given max
for a in range(1, (maxA + 1)):
    b = 0
    norm = abs((a * a) - (b * b))
    while norm <= normMax:
        for normsInT in setT:
            if norm == normsInT:
                # if the number is a norm in setT
                # that number is prime
                # add the value to graph
                avalues.append(a)
                bvalues.append(b)

                avalues.append(-a)
                bvalues.append(b)

                avalues.append(-a)
                bvalues.append(-b)

                avalues.append(a)
                bvalues.append(-b)

        b += 1
        norm = abs((a * a) - (b * b))

    # graph all the prime numbers
    plt.scatter(avalues, bvalues)
plt.show()
main()
from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

def quadchar(d, n):
    # this function takes a number n
    # and returns (d / n)
    # the quadratic residue
    # skip even numbers
    if (n % 2) == 0:
        return -1

    quadchar = jacobi_symbol(d, n)
    return quadchar

def main():
    # store values in csv file for future use
    download_dir = "seventynineData.csv"

    myFile = open(download_dir, "a")
    # "a" indicates that you’re appending strings to the file
    writer = csv.writer(myFile)

    # in this case our radicand is 79
    r = 79

    # the discriminant is equal to 4r for r = 3 (mod 4)
\[ d = 4r \]

\[
\text{set } S = \text{list}() \\
\text{set } T = \text{list}() \\
\]

# the norm that gives us the first 5000 values
normMax = 39575

# Start with a set \( S \) of the natural numbers 2 \( \leq n \leq \text{normMax} \)
# where \( n \) is a (prime) divisor of \( d \)
# or \( \text{quadchar}(n) = 1 \)

for \( n \) in range(2, (normMax + 1)):
    if \( (d \% n) == 0 \) and \( \text{isprime}(n) \):
        setS.append(n)
    else if \( \text{quadchar}(d, n) == 1 \):
        setS.append(n)

# make a copy of \( S \) called \( T \)
for \( s \) in setS:
    setT.append(s)

# for every element in \( T \)
# take the smallest number, \( t \)
# such that \( \text{quadchar}(t) = 1 \)
# remove elements in \( T \) which are products of \( t \) and some element in \( S \)
# complete when next element is larger than square root of normMax

\[ t = 0 \]
\[ i = 0 \]

while \( t \leq \sqrt{\text{normMax}} \) and \( i < \text{len}(\text{setT}) \):
    \[ t = \text{setT}[i] \]
    if \( \text{quadchar}(d, t) == 1 \):
        for \( s \) in setS:
            product = \( t \times s \)
            for \( value \) in setT:
                if \( value == \) product:
                    setT.remove(value)
        \[ i = i + 1 \]

# set \( T \) contains all the norms of prime numbers less than the norm of 39575
#loop through all the numbers and if the norm is in setT then
#that number is prime
for normMax in range(1, 39575):
    norm = 0
    maxA = int(sqrt((normMax)))

    #start the total count of primes at 0
    primeCount = 0
    numberCount = 0

    #for each "a" value from 1 to max A value
    #loop through all possible "b" values up to given max
    for a in range(1, (maxA + 1)):
        b = 0
        norm = abs((a * a) - (r * (b * b)))
        while norm <= normMax:
            numberCount += 1
            for normsInT in setT:
                #add one to count for this prime
                primeCount = primeCount + 1
                b += 1
                norm = abs((a * a) - (r * (b * b)))

            #add this amount of numbers
            #and the prime number count to a csv file
            #so we can graph the output after
            row = [numberCount] + [primeCount]
            writer.writerow(row)

    myFile.close()
Listing A.33: Program that graphs the count of the prime numbers

```python
# Program to graph the prime numbers in the set \( a + b [\sqrt{79}] \)
# using algorithm by Dr. Theodorus J. Dekker
# from the values stored in a csv file
# Author: Jamie Nelson
# Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

numberData = list()
countData = list()

# csv file with values
download_dir = "seventynineData.csv"

myFile = open(download_dir, "r")
"r" indicates that you're reading strings from the file

reader = csv.reader(myFile)
for row in reader:
    numberData.append(int(row[0]))
    countData.append(int(row[1]))

myFile.close()

# plot the values for the amount of numbers on the x axis
# and the prime number count on the y axis
plt.plot(numberData, countData, 'b−')
plt.xlabel('Total Amount of Numbers')
plt.ylabel('Amount of Prime Numbers')
plt.title('Count of Prime Numbers in the set \( a + b + \sqrt{79} \)')
plt.show()
```
#Program to graph the prime numbers in the set \( a + b \sqrt{79} \)

#using algorithm by Dr. Theodorus J. Dekker

#from the values stored in a csv file

#and graphs an asymptotic function

#Author: Jamie Nelson

#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

numberData = list()
countData = list()

#csv file with values
download_dir = "seventynineData.csv"

myFile = open(download_dir, "r")
#"r" indicates that you're reading strings from the file

reader = csv.reader(myFile)
for row in reader:
    numberData.append(int(row[0]))
countData.append(int(row[1]))

myFile.close()

#array to store the \( 14x/25 \ln(x) \) function
xvalues = list()
yvalues = list()

#compute the values for \( 14x/25 \ln(x) \)
for x in range(2,5184):
xvalues.append(x)
y = (14*x) / (25 * math.log(x))
yvalues.append(y)
40 #plot the values for the amount of numbers on the x axis
41 #and the prime number count on the y axis
42 #also plot 14x/25ln(x) in red dashed lines to compare
43 plt.plot(numberData, countData, 'b-', label='Count of Prime Numbers')
44 plt.plot(xvalues, yvalues, 'r--', label=r'$y = \frac{14x}{25\ln(x)}$')
45 plt.legend(loc='lower right')
46 plt.xlabel('Total Amount of Numbers')
47 plt.ylabel('Amount of Prime Numbers')
48 plt.title('Count of Prime Numbers in the set a + b\u221a + 79')
49 plt.show()
#Program to calculate the proportion of prime numbers
#out of the total amount of numbers in the set \( a + b\sqrt{79} \)
#and stores the values in a csv file

#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv


def quadchar(d, n):
    #this function takes a number \( n \)
    #and returns \( \frac{d}{n} \)
    #the quadratic residue

    #skip even numbers
    if (n % 2) == 0:
        return -1

    quadchar = jacobi_symbol(d, n)
    return quadchar

def main():

    #store values in csv file for future use
    download_dir = "seventynineProportionData.csv"

    myFile = open(download_dir, "a")
    #"a" indicates that you’re appending strings to the file

    writer = csv.writer(myFile)

    #in this case our radicand is 79
    r = 79

    #the discriminant is equal to \( 4r \) for \( r = 3 \pmod{4} \)
\( \text{setS} = \text{list}() \)
\( \text{setT} = \text{list}() \)

#the norm that gives us the first 5000 values

normMax = 39575

#Start with a set S of the natural numbers 2 <= n <= normMax
#where n is a (prime) divisor of d
#or quadchar(n) = 1

for n in range(2, (normMax + 1)):
    if (d % n) == 0 and isprime(n):
        setS.append(n)
    elif quadchar(d, n) == 1:
        setS.append(n)

#make a copy of S called T
for s in setS:
    setT.append(s)

#for every element in T
#take the smallest number, t
#such that quadchar(t) = 1
#remove elements in T which are products of t and some element in S
#complete when next element is larger than squareroot of normMax

i = 0

while t <= sqrt(normMax) and i < len(setT):
    t = setT[i]
    if quadchar(d, t) == 1:
        for s in setS:
            product = t * s
            for value in setT:
                if value == product:
                    setT.remove(value)
        i = i + 1

#setT contains all the norms of prime numbers less than the norm of 39575
#loop through all the numbers and if the norm is in setT then
#that number is prime
for normMax in range(1, 39575):

    norm = 0
    maxA = int(sqrt((normMax)))

    #start the total count of prime numbers at 0
    primeCount = 0

    #start the total count of numbers at 0
    numberCount = 0
    proportion = 0

    #for each "a" value from 1 to max A value
    #loop through all possible "b" values up to given max
    for a in range(1, (maxA + 1)):
        b = 0
        norm = abs((a * a) - (r * (b * b)))
        while norm <= normMax:
            numberCount += 1
            for normsInT in setT:
                if norm == normsInT:
                    #add one to count for this prime
                    primeCount = primeCount + 1
                    b += 1
                    norm = abs((a * a) - (r * (b * b)))

        #calculate the proportion
        proportion = primeCount / numberCount

        #add this amount of numbers and the proportion value to a csv file
        #so we can graph the output after
        row = [numberCount] + [proportion]
        writer.writerow(row)

    myFile.close()
Listing A.36: Program that graphs the proportion of prime numbers

```python
#Program to graph the proportion of prime numbers in the set a + b[\sqrt{79}]
#using algorithm by Dr. Theodorus J. Dekker
#from the values stored in a csv file
#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

#array to store the percentage of prime numbers and the amount of numbers pairs
proportionData = list()
numberData = list()

#csv file with values
download_dir = "seventynineProportionData.csv"

myFile = open(download_dir, "r")  #"r" indicates that you’re reading strings from the file
reader = csv.reader(myFile)
for row in reader:
    numberData.append(int(row[0]))
    proportionData.append(float(row[1]))

myFile.close()

#plot the values for the amount of numbers on the x axis
#and the proportion on the y axis
plt.plot(numberData, proportionData, 'b-')
plt.xlabel('Total Amount of Numbers')
plt.ylabel('Proportion of Prime Numbers out of the Total Amount of Numbers')
plt.title('Proportion of Prime Numbers in the set a + b+' + u'\u221a ' + "79")
plt.show()
```
A.7 Code for Proportion Graph

Listing A.37: Program that graphs the proportion of prime numbers in seven different sets

```python
#Program to graph the proportion of prime numbers in the
#set of rational integers, Z[i], Z[sqrt(11)], Z[sqrt(15)],
#Z[sqrt(34)], Z[sqrt(35)], and Z[sqrt(79)]
#using algorithm by Dr. Theodorus J. Dekker
#from the values stored in a csv file
#Author: Jamie Nelson
#Mentor: Jackie Anderson

from sympy import *
import matplotlib.pyplot as plt
import numpy
import math
import csv

#arrays to store the percentage of prime numbers and the amount of numbers
proportionDataIntegers = list()
proportionDataGI = list()
proportionDataEleven = list()
proportionDataFifteen = list()
proportionDataThirtyfour = list()
proportionDataThirtyfive = list()
proportionDataSeventynine = list()

teachDataIntegers = list()
teachDataGI = list()
teachDataEleven = list()
teachDataFifteen = list()
teachDataThirtyfour = list()
teachDataThirtyfive = list()
teachDataSeventynine = list()

#loop through all rational integers
for integer in range(1, 5184):
    #start prime count and proportion at 0
```

115
primeCount = 0
proportion = 0

# start number at 0
num = 0

# loop through all numbers less than the current integer
# and count the amount of prime numbers
while num < integer:
    num += 1
    if isprime(num):
        primeCount +=1

# find the proportion
proportion = primeCount / integer

# add this number amount and the proportion value to an array
# so we can graph the output after
numberDataIntegers.append(integer)
proportionDataIntegers.append(proportion)

numberCount = 0
normMax = 1

# loop through the first 5184 values for the Gaussian integers
while numberCount <= 5184:
    norm = 0
    maxA = int(sqrt(normMax))

    # start the total count of prime numbers at 0
    primeCount = 0

    # start the total count of numbers at 0
    numberCount = 0
    proportion = 0

    # for each "a" value from 1 to max A value
    # loop through all possible "b" values up to given max
    for a in range(1, (maxA + 1)):
b = 0
norm = (a * a) + (b * b)
aMod = a % 4
if isPrime(a) and aMod == 3:
    #add one to count for this prime
    primeCount += 1
while norm <= normMax:
    numberCount += 1
    if isPrime(norm):
        #add one to count for this prime
        primeCount = primeCount + 1
    b += 1
    norm = (a * a) + (b * b)

    #find the proportion
    proportion = primeCount / numberCount
    #add this number amount and the proportion value to an array
    #so we can graph the output after
    numberDataGI.append(numberCount)
    proportionDataGI.append(proportion)
    normMax += 1

#csv files with values
elevenFile = open("elevenProportionData.csv", "r")
fifteenFile = open("fifteenProportionData.csv", "r")
thirtyfourFile = open("thirtyfourProportionData.csv", "r")
thirtyfiveFile = open("thirtyfiveProportionData.csv", "r")
seventynineFile = open("seventynineProportionData.csv", "r")
elevenReader = csv.reader(elevenFile)
for row in elevenReader:
    numberDataEleven.append(int(row[0]))
    proportionDataEleven.append(float(row[1]))
elevenFile.close()
fifteenReader = csv.reader(fifteenFile)
for row in fifteenReader:
    numberDataFifteen.append(int(row[0]))
proportionDataFifteen.append(float(row[1]))

fifteenFile.close()

doctorRow = csv.reader(thirtyfourFile)

for row in doctorRow:
    numberDataThirtyfour.append(int(row[0]))
    proportionDataThirtyfour.append(float(row[1]))

thirtyfourFile.close()

for row in doctorRow:
    numberDataThirtyfive.append(int(row[0]))
    proportionDataThirtyfive.append(float(row[1]))

thirtyfiveFile.close()

for row in doctorRow:
    numberDataSeventynine.append(int(row[0]))
    proportionDataSeventynine.append(float(row[1]))

seventynineFile.close()

# plot the values for the amount of numbers on the x axis
# and the proportion on the y axis
plt.plot(numberDataIntegers, proportionDataIntegers, 'tab:gray',
         label='Proportion of Prime Numbers in the Integers')
plt.plot(numberDataGI, proportionDataGI, 'tab:blue',
         label='Proportion of Gaussian Prime Numbers')
plt.plot(numberDataEleven, proportionDataEleven, 'tab:orange',
         label='Proportion of Prime Numbers in Z[+ u\u221a\ u221a11 + |]')
plt.plot(numberDataFifteen, proportionDataFifteen, 'tab:green',
         label='Proportion of Prime Numbers in Z[+ u\u221a\ u221a15 + |]')
plt.plot(numberDataThirtyfour, proportionDataThirtyfour, 'tab:red',
         label='Proportion of Prime Numbers in Z[+ u\u221a\ u221a34 + |]')
plt.plot(numberDataThirtyfive, proportionDataThirtyfive, 'tab:purple',
         label='Proportion of Prime Numbers in Z[+ u\u221a\ u221a35 + |]')
plt.plot(numberDataSeventynine, proportionDataSeventynine, 'tab:pink',
label= 'Proportion of Prime Numbers in $\mathbb{Z}[\sqrt{a^2 + b^2}]$'
plt.legend(loc='upper right')
plt.xlabel('Amount of Numbers')
plt.ylabel('Proportion of Prime Numbers out of Total Amount of Numbers')
plt.title('Proportion of Prime Numbers in the set $a + b\sqrt{a^2 + b^2}$')
plt.show()