AMSI VACATION RESEARCH SCHOLARSHIPS 2020–21

Get a Thirst for Research this Summer

Computational Number Theory in Sage

Cameron Shaw-Carmody
Supervised by Prof Florian Breuer
University of Newcastle, Australia

Vacation Research Scholarships are funded jointly by the Department of Education, Skills and Employment and the Australian Mathematical Sciences Institute.

Contents

1 Acknowledgements

I would like to thank Professor Florian Breuer for his mentorship and help throughout this project. He has provided his time and expertise generously and it has enabled me to explore an area of mathematics and coding that I had not experienced previously. I would also like to thank AMSI and The University of Newcastle for the opportunity to undertake this research, and for the generosity of AMSI in affording me a scholarship.

2 Statement of Ownership

I developed the coding in Sage for the project under the supervision and with the input of Professor Florian Breuer. In addition, from the data generated by this code I created the graphs and interpreted them under his supervision.

My written report has been edited and proofread by Prof. Breuer.

3 Abstract

Peter Stevenhagen has conjectured that the density of square-free $d \equiv 5 \pmod 8$ for which the Pellian equation $x^2 - dy^2 = \pm 4$ has no odd solutions is $1/3$. There are many related problems where $d = p$ is prime with no known definite results. The principal aim of this research was to investigate whether a phenomenon similar to Chebyshev's Bias occurs in the case where $d = p \equiv 1 \pmod 4$ and p is a prime, with fluctuations caused by low-lying zeroes of an L-function. This was undertaken by defining a ψ function corresponding to the data set being investigated and using the programming language Sagemath (commonly referred to as Sage) to determine numerically the values of the ψ function for values of $d = p \equiv 1 \pmod 4$ where p is a prime, up to 2×10^8. These results were manipulated and had the Fourier transform applied, revealing that there was no clearly observable associated L-function to describe the occurrences of primes congruent to $1 \pmod 4$ in the data set of numbers considered.

4 Introduction

Peter Stevenhagen has conjectured that the density of square-free $d \equiv 5 \pmod 8$ for which the Pellian equation $x^2 - dy^2 = \pm 4$ has no odd solutions is $1/3$ (Stevenhagen, 1996). Despite having proven some partial results for this conjecture, there are some areas that remain unexplored. Where $d = p$ is prime is one of these contexts, and there are no definite results known. But the question is of relevance to periods of Ducci sequences and multiplicative orders of Gauss periods in finite fields, in addition to other topics (Breuer, 2019).

A principal aim of the research was to investigate a specific case related to Stevenhagen's conjecture, and whether a phenomenon similar to Chebyshev's Bias occurred. The set $d = p \equiv 1 \pmod 4$ where p is a prime was chosen and it was explored whether fluctuations caused by low-lying zeroes of an L-function were present.

Chebyshev's Bias is the phenomenon that most of the time there are more primes of the form $4k+3$ than of the form $4k+1$, where $k \in \mathbb{N} \cup \{0\}$, up to the same limit (Granville & Martin, 2006), and an L-function is a meromorphic function, that is, a single-valued function in all but a discrete subset on the complex plane (Hutama, 2017).

Within the research, it was necessary to consider the Pellian equation $x^2 - dy^2 = \pm 4$ for a large data set of values for d and determine the residue class of the fundamental solution (Jacobson & Williams, 2009). But in this context, even small values of d produce extremely large integer solutions to x and y (Jacobson & Scheidler, 2014). As a result, in order to determine the residue class of the fundamental solution to $x^2 - dy^2 = \pm 4$, where $d = p \equiv 1 \pmod 4$ and p is a prime without computing the solution itself, the project involved developing fast algorithms in Sage.

Sage is a computational mathematics program that is an extension of Python, thus it has been devised specifically for maths purposes. In particular, it is well-suited for the areas of number theory and algebra. Through the use of Sage it was possible to create code that involved modifying the standard method of solving $x^2 - dy^2 = \pm 4$ via the continued fraction expansion of $\sqrt{d}$. This was done by only determining whether the numbers generated in each step of the process were congruent to 0 or 1 modulo 2, thereby greatly reducing the size of the computer memory needed to store the numbers and increasing the speed of calculations. This also applied to the fundamental solutions of the Pellian equation for each value of d.

5 Chebyshev Function

We begin by defining functions that are relevant to the process that is used to determine the results of the research. The prime counting function is defined as the number of prime numbers p less than or equal to a given number x:

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

There is an associated Chebyshev function $\psi(x)$ which has a similar form to the prime counting function, except that the counts of the number of prime powers p^n less than x are weighted by $\ln(p)$:

$$\psi(x) = \sum_{p^n \leq x} \ln(p) \tag{2}$$

There is also an explicit formula for $\psi(x)$, denoted $\psi_0(x)$, that takes the value half way between the previous and the subsequent unique values of $\psi(x)$. It is defined by the expression:

$$\psi_0(x) = x - \sum_{\rho} \frac{x^\rho}{\rho} - \ln(2\pi) - \frac{1}{2}\ln(1 - x^{-2}) \tag{3}$$

where ρ ranges over the critical zeroes of the Riemann zeta function (Hutama, 2017). This explicit formula provides a connection between the Chebyshev function and the zeroes of the Riemann zeta function, allowing a method to be developed to determine the imaginary parts of the zeroes of the Riemann zeta function. It is based on the Chebyshev function as explored in the following sections.

6 The Riemann Zeta Function

The Riemann zeta function is a function of a complex variable s that analytically continues the sum of the Dirichlet series. The formula below defines $\zeta(s)$ for $\Re(s) > 1$, but analytic continuation of the function $\zeta(s)$ can be used to extend its applicability to the whole complex plane, except for a pole at s=1 (Breuer, 2013). It is suited to the problem because the zeroes of the Riemann zeta function are related to the distribution of prime numbers on the number line. It has the form:

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s} = \prod_{p} \frac{1}{1 - p^{-s}} \tag{4}$$

such that in the product, p runs over all primes.

A property of the Riemann zeta function is that it possesses two types of zeroes: trivial zeroes and critical zeroes. Both types of zeroes are shown within Figure 1. Trivial zeroes are those that occur on the negative half of the real axis and are not of interest for this research. However, the critical zeroes are of much greater importance within this project.

The critical zeroes are the zeroes of the Riemann zeta function that are governed by the Riemann Hypothesis that all critical zeroes of $\zeta(s)$ lie on the critical line $\mathbb{R}(s) = \frac{1}{2}$ (Breuer, 2013). Therefore, these critical zeroes ρ are all of the form $\rho = \frac{1}{2} + it$ for various $t \in \mathbb{R}$.

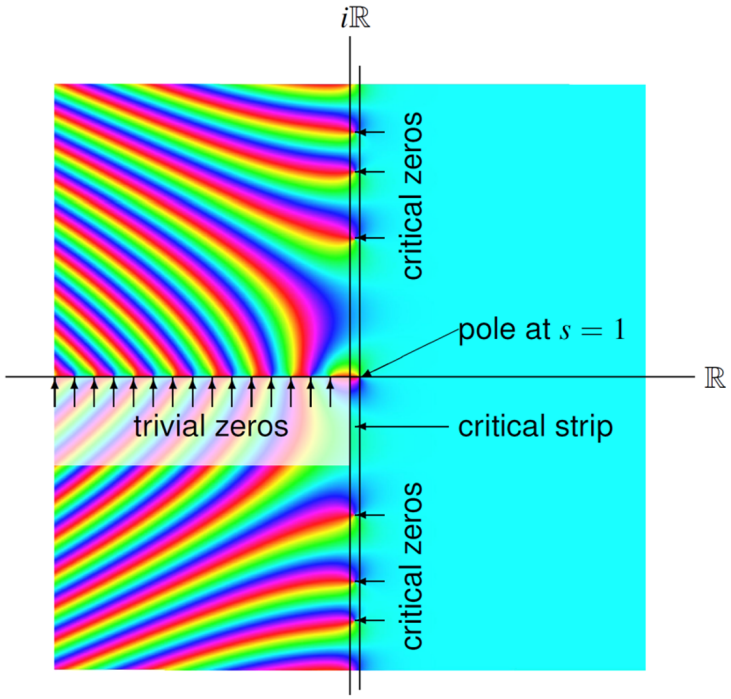

Figure 1: Phase plot of the Riemann zeta function (Breuer, 2013)

7 Application of Fourier Transform

By rearranging the explicit formula for the Chebyshev function and assuming the Riemann Hypothesis to be true (such that $\rho = \frac{1}{2} + it$ for various t) then it is possible to obtain the expression:

$$\psi_0(x) - x + \ln(2\pi) + \frac{1}{2}\ln(1 - x^{-2}) = \sum_{\rho} \frac{x^{\rho}}{\rho} \tag{5}$$

It is also known that the right-hand side of the above equation is approximately equal to a function with the form of a Fourier series:

$$\sum_{\rho} \frac{x^{\rho}}{\rho} \approx \sqrt{x} \sum_{t} \frac{1}{t} \sin\Big(t \log(x)\Big) \tag{6}$$

From these two equations it is found that:

$$\frac{\psi_0(x) - x + \ln(2\pi) + \frac{1}{2}\ln(1 - x^{-2})}{\sqrt{x}} \approx \sum_{t} \frac{1}{t} \sin\Big(t \log(x)\Big) \tag{7}$$

By taking the Fourier transform of this expression it is possible to detect the values of t, revealing the imaginary part of the critical zeroes of the Riemann zeta function. When this process was performed on the Chebyshev function, the following graphs were generated:

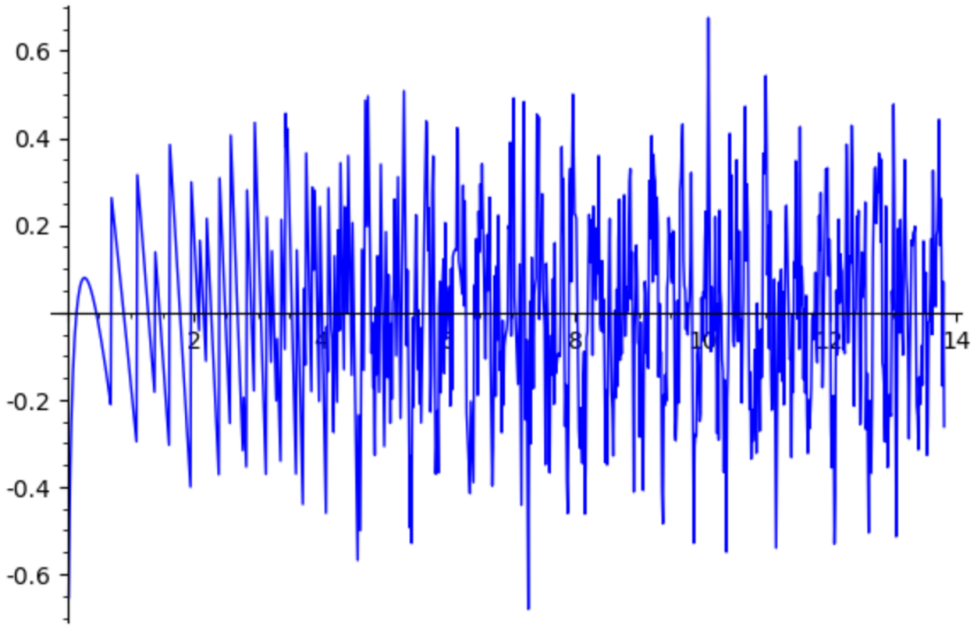

Figure 2: Logarithmic Plot of $\frac{\psi_0(x) - x + \ln(2\pi) + \frac{1}{2}\ln(1-x^{-2})}{\sqrt{x}} \approx \sum_t \frac{1}{t} \sin\Big(t \log(x)\Big)$

The logarithmic graph of the expression gives Figure 2, a plot which randomly oscillates around 0 and does not appear to provide any relevant information. However, this changes when the Fourier transform is applied to the data set.

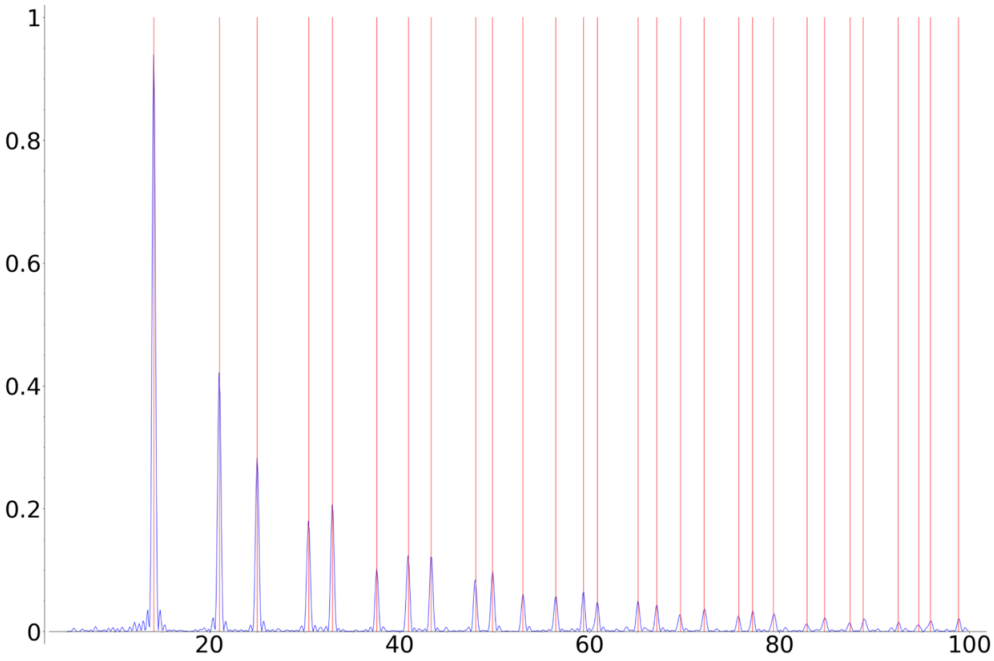

Figure 3: Fourier Transform of Chebyshev Function

The blue line in Figure 3 gives the Fourier transform of the expression $\frac{\psi_0(x)-x+\ln(2\pi)+\frac{1}{2}\ln(1-x^{-2})}{\sqrt{x}}$ with the peaks displaying the values of the imaginary parts of the critical zeroes of the Riemann zeta function, as predicted by the outlined method. The red vertical lines are the known imaginary parts of the critical zeroes of the Riemann zeta function. This graph shows that the predicted and actual values for the imaginary parts of the critical zeroes match almost exactly.

8 The Dirichlet L-Function

The method outlined above has been shown to reveal the imaginary parts of the Riemann zeta function, but it is unknown if it can be used to gain any information about the zeroes of other functions. Of particular interest was whether this method could be used to determine the zeroes of L-functions since the primary goal of this research is to identify whether the case where $d = p \equiv 1 \pmod 4$ and p is a prime has an associated L-function. Therefore, this method was applied to a Dirichlet L-function where the zeroes are already known to test the method's applicability to other functions, particularly those of the form of the data set that was of interest for this research.

The following Dirichlet L-function was considered to determine if the outlined method was appropriate to find the zeroes of the function. It had the form:

$$L(s,\chi) = \sum_{n=1}^{\infty} \frac{\chi(n)}{n^s} \tag{8}$$

where

$$\chi(n) = \begin{cases} 1 & \text{if } n \equiv 1 \pmod 4 \\ 0 & \text{if } n \equiv 0 \pmod 4 \text{ or } n \equiv 2 \pmod 4 \\ -1 & \text{if } n \equiv 3 \pmod 4 \end{cases} \tag{9}$$

We also define a function

$$\psi_\chi(x) = \sum_{p^n \leq x} \chi(p^n) \ln(p) \tag{10}$$

which has an associated explicit formula of

$$\psi_0(x, \chi) = -\sum_{\rho_*} \frac{x^{\rho_*}}{\rho_*} - \frac{L'(0, \chi)}{L(0, \chi)} \tag{11}$$

where ρ_* ranges over the critical zeros of $L(s, \chi)$.

When the same process of rearrangement of the explicit formula and approximation to a Fourier series was applied to this function as was applied to the Chebyshev function, then the plot on the logarithmic graph in Figure 4 is generated. Like the logarithmic graph produced for the Chebyshev function, this plot randomly oscillates around a certain value. In this case, it oscillates around the value of $\frac{L'(0,\chi)}{L(0,\chi)} \approx -0.783$ rather than 0, as occurred in the Chebyshev function.

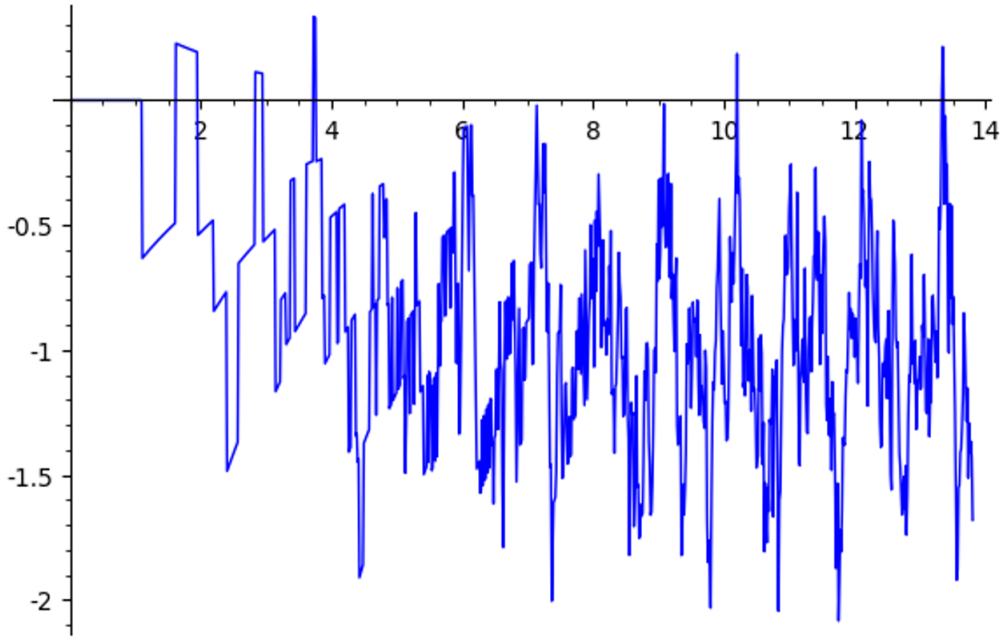

Figure 4: Logarithmic Plot of $\frac{\psi_\chi(x)}{\sqrt{x}}$

When the Fourier transform is applied to the data set, Figure 5 was created. Like the Chebyshev function, the blue plot gives the Fourier transform of $\frac{\psi_\chi(x)}{\sqrt{x}}$ with the peaks being the expected values of the zeroes of the

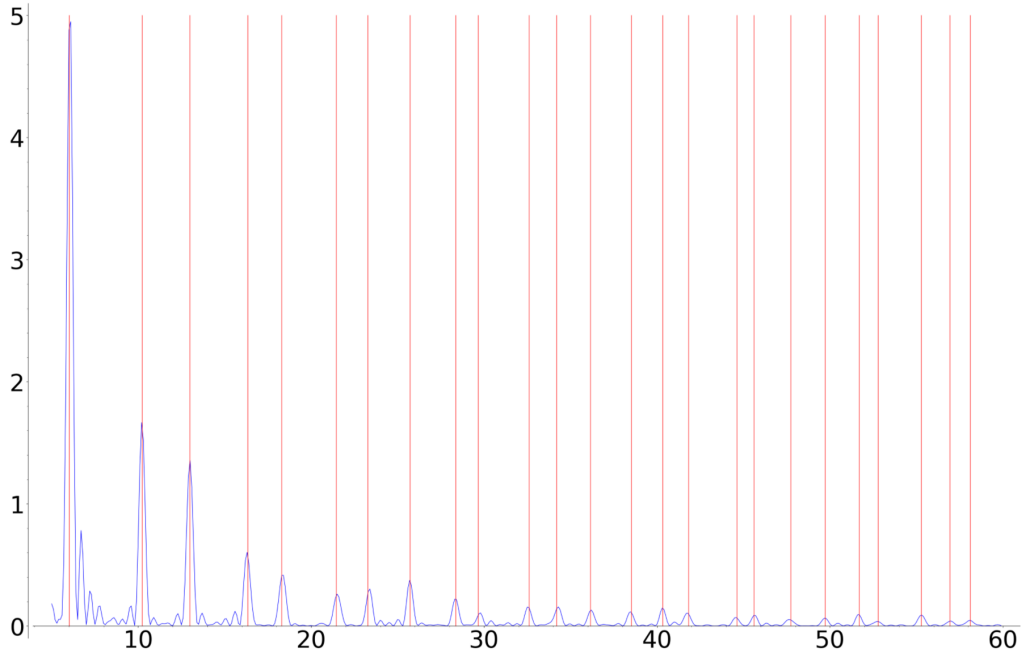

Figure 5: Fourier Transform of Dirichlet L-Function

Dirichlet L-function defined above, and the red vertical lines are the known zeroes of the Dirichlet L-function. The graph shows that the predicted and actual values for the zeroes match almost exactly.

This result indicates that the method can also be used for determining the zeroes of L-functions, and so can be used in the case of the data set of $d = p \equiv 1 \pmod 4$ where p is a prime to identify if there is an L-function associated with this data set and, if so, determine the corresponding zeroes.

9 ψ-Function for Research

The ψ-function that arises from the set of numbers $d = p \equiv 1 \pmod 4$ where p is a prime is the function of main interest within the research. It has the form:

$$\psi_{comb}(x) = \sum_{p^n \leq x} \chi(p^n) \ln(p) \tag{12}$$

where p is a prime and:

$$\chi(p^n) = \begin{cases} -\frac{1}{2} & \text{if } p \equiv 1 \pmod 4 \text{ and } x^2 - dy^2 = -4 \text{ has only even solutions} \\ 1 & \text{if } p \equiv 1 \pmod 4 \text{ and } x^2 - dy^2 = -4 \text{ has only odd solutions} \end{cases} \tag{13}$$

However, the form of the formula was changed to allow for implementation of a faster algorithm into Sage. Specifically, separating the function into two different functions denoted $\psi_{even}(x)$ and $\psi_{odd}(x)$.

$\psi_{even}(x)$ was defined as:

$$\psi_{even}(x) = \sum_{p^n \leq x} \ln(p) \tag{14}$$

where the sum ranges over primes $p \equiv 1 \pmod 4$ and $x^2 - py^2 = -4$ has only even solutions.

And $\psi_{odd}(x)$ was similarly defined as:

$$\psi_{odd}(x) = \sum_{p^n \leq x} \ln(p) \tag{15}$$

where the sum ranges over primes $p \equiv 1 \pmod 4$ and $x^2 - py^2 = -4$ has only odd solutions.

A linear combination of these two functions can then be formed to reproduce the original function:

$$\psi_{comb}(x) = \psi_{odd}(x) - \frac{1}{2}\psi_{even}(x) \tag{16}$$

When $\psi_{even}(x)$ and $\psi_{odd}(x)$ are graphed separately over the data set of primes less than 2×10^8, as shown in Figure 6, the plots indicate that $\psi_{even}(x)$ grows at approximately double the rate of $\psi_{odd}(x)$. It also gives the impression that both $\psi_{even}(x)$ and $\psi_{odd}(x)$ increase linearly. However, this is not the case because there are slight variations within the increase of $\psi_{even}(x)$ and $\psi_{odd}(x)$ as x increases, and it is only because of the large size of the data set considered that these variations are comparatively small and so are not seen when plotted.

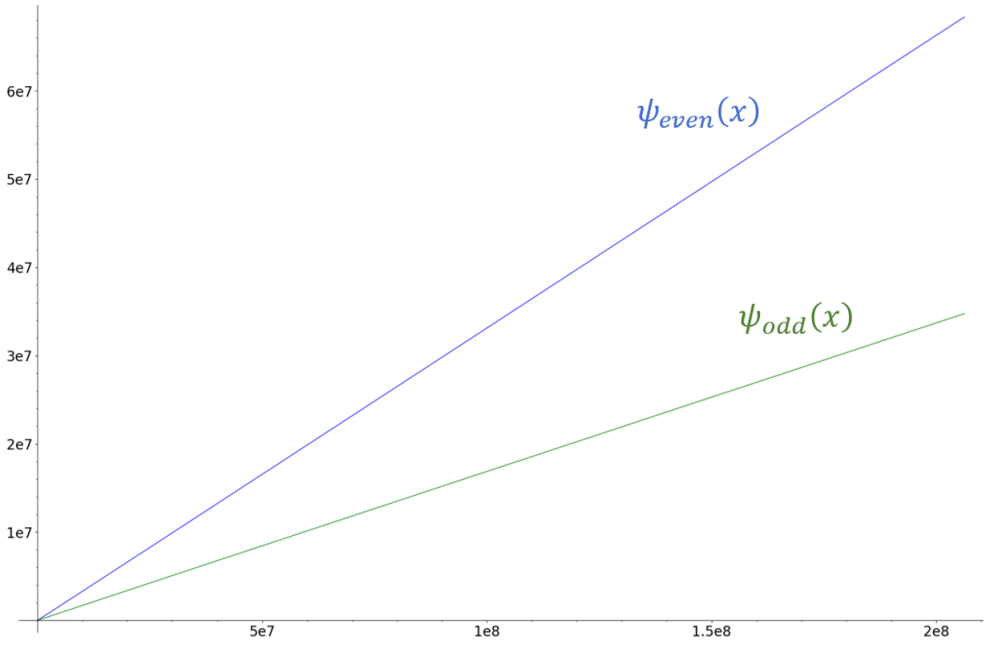

Figure 6: Plot of $\psi_{even}(x)$ and $\psi_{odd}(x)$

When $\psi_{comb}(x)$ is plotted over the same range, as in Figure 7, these slight variations become apparent. Also, the value of the $\psi_{comb}(x)$ is much smaller than the values of the $\psi_{even}(x)$ and $\psi_{odd}(x)$ functions at the same value of x.

Using the same process for finding a Fourier series that approximates the function applied to the Chebyshev and Dirichlet L-function, it is possible to plot this function on a logarithmic graph, as presented in Figure 8. The plot of this function shows an asymptotic structure going to 0 as x increases. This is much different from that of both the Chebyshev and Dirichlet L-function, which oscillated randomly around particular values. Since

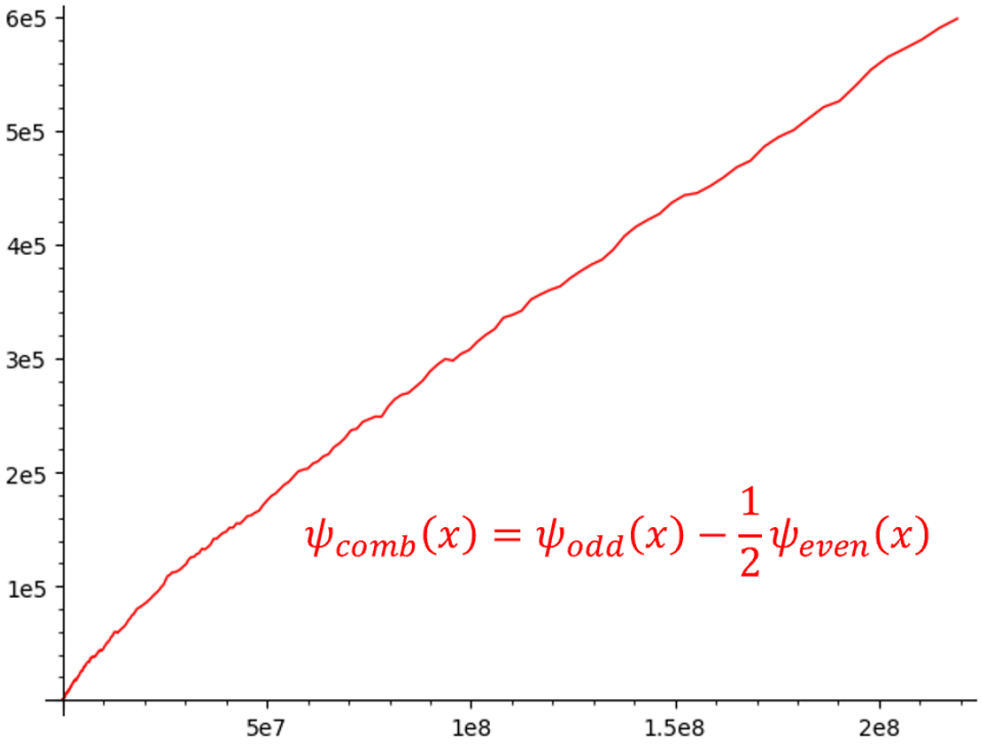

Figure 7: Plot of $\psi_{comb}(x)$

the plot corresponding with $\psi_{comb}(x)$ has a different shape to functions where this method is known to work, it gives an indication that there may not be an L-function associated with $\psi_{comb}(x)$.

The Fourier transform was applied to the data set to provide more conclusive evidence about the presence of an L-function for the investigated function. Figure 9 shows that the plot exhibits an asymptotic structure with the asymptote at zero. There are no obvious peaks, indicating that there are no observable zeroes of an associated L-function. Therefore, there does not appear to be an L-function associated with the function and data set corresponding to the numbers $d = p \equiv 1 \pmod 4$ where p is a prime.

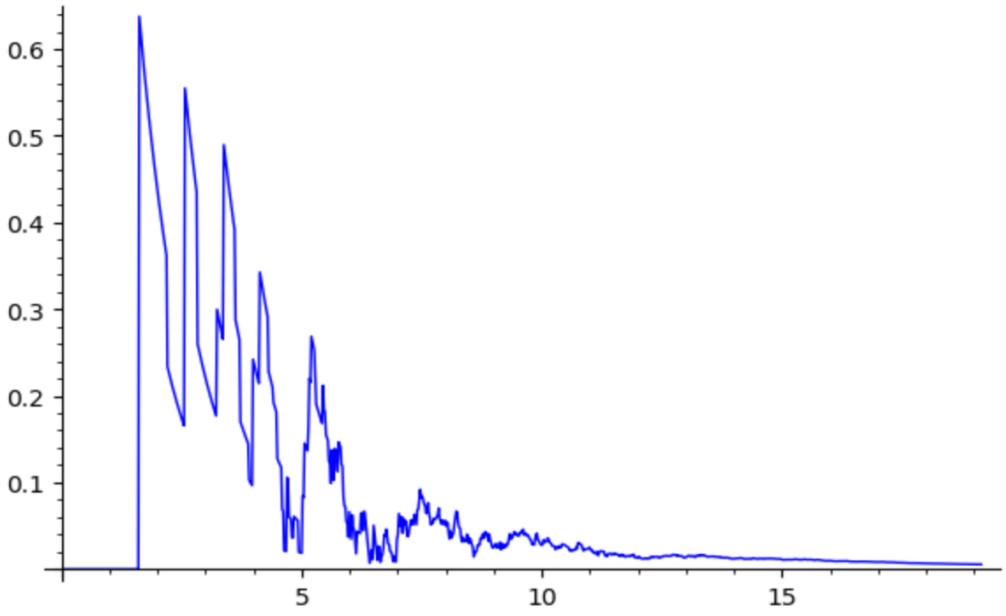

Figure 8: Logarithmic Plot of $\frac{\psi_{comb}(x)}{\sqrt{x}}$

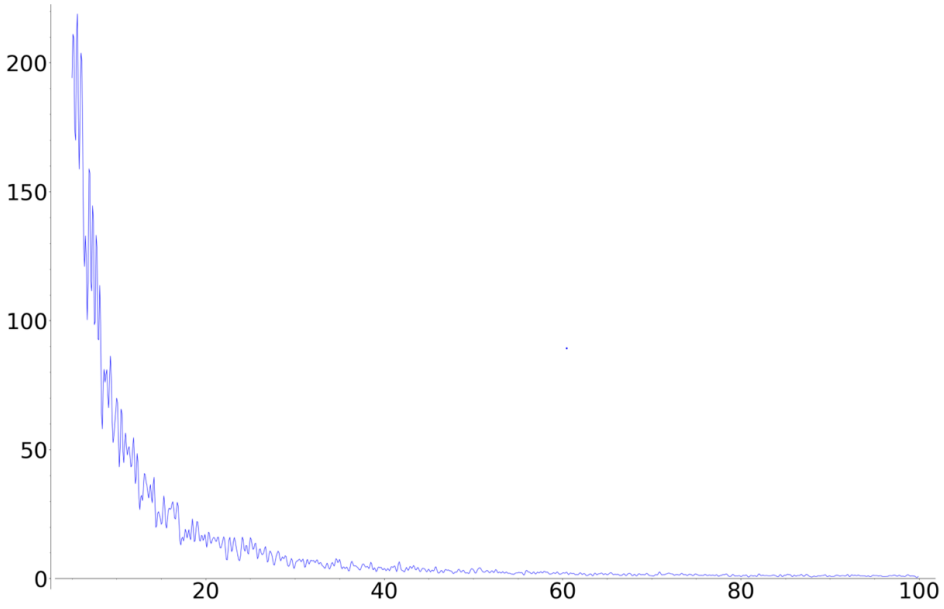

Figure 9: Fourier Transform of $\frac{\psi_{comb}(x)}{\sqrt{x}}$

10 Conclusion and Further Research

The data shows that there is no clearly observable associated L-function to describe the occurrence of the primes congruent to 1 (mod 4) less than to 2×10^8. Although from the data considered during the research there appears to be no associated L-function, it is possible that if a larger data set is considered, evidence for an L-function would become more apparent and so this could be an extension of the current research. To do so would require developing a faster algorithm in Sage for calculating the $\psi_{comb}(x)$ function for each value of x tested. Based on the data obtained, though, it seems unlikely that an L-function does exist for the data set of interest.

Although the method in this research did not work for the particular data set considered, it is possible that a different data set might be more successful. Therefore, further research could be done on other data sets of prime congruences, such as those proposed by Stevenhagen of $d = p \equiv 5 \pmod 8$ where p is prime.

References

Breuer, F 2019, 'Periods of Ducci Sequences and odd solutions to a Pellian Equation', *Bulletin of the Australian Mathematical Society*, vol. 100, pp. 201-205.

Breuer, F 2013, *The Parallel Worlds of Number Theory*. Inaugural lecture, Department of Mathematical Sciences Stellenbosch University.

Granville, A & Martin, G 2006, 'Prime Number Races', *The Mathematical Association of America*, Monthly vol. 113, pp. 1-33.

Hutama, D.J 2017, *Implementation of Riemann's Explicit Formula for Rational and Gaussian Primes in Sage*. Honours Dissertation, University of Montreal.

Jacobson, M.J & Scheidler, R 2014, 'Infrastructure: structure inside the class group of a real quadratic field', *Notices of the AMS*, vol. 61, no. 1, pp. 36-46.

Jacobson, M.J & Williams, H 2009, *Solving the Pell Equation*, Canadian Mathematical Society, Ottawa.

Stevenhagen, P 1996, 'On a problem of Eisenstein', *Acta Arithmetica*, vol. 74, no. 3, pp. 259-268.

11 Appendix

The Sage code used to develop the results and graphs in the chapter 9 are presented here. The results of chapters 7 and 8 were obtained by the same methods as in chapter 9, and so the code is almost identical and will not be presented.

11.1 Sage Code for Finding the Fundamental Solutions of the Pellian Equation

```
from sage.parallel.decorate import parallel
from sage.arith.misc import is_prime
from sage.misc.functional import N
from sage.functions.other import sqrt
from sage.functions.other import floor

def Pell_sol_mod_M(p, M, test_prime=0, test_Q_iterations=1):
    if test_prime:
        if not is_prime(p):
            return()

    num_iterations = 1 # for testing only
    Q_max = 2 # largest value of Q seen; for testing only

    n = 30    # Number of significant figures for the square root of p (to start with)
    n1 = n
    rtp = N(sqrt(p),digits=n)

    delta = (1 + rtp)/2

    P = 1
    Q = 2
    q = floor(delta)

    # initialise B's anf G's, one for each l:
    B = 1
    B_old = 0
    G = ((2*q)-1) % M
    G_old = 2 % M


    # First iteration
    P = (q*Q)-P
    Q = (p-(P**2))/Q
    q = floor((P+rtp)/Q)
    G_old, G = G, ((q*G+G_old) % M)
    B_old, B = B, ((q*B+B_old) % M)

    if test_Q_iterations:  # slower, more careful loop with tests
        while Q != 2:
```

```
            # Subsequent iterations of the recurrence relation
            num_iterations += 1
            P = (q*Q)-P
            Q = (p-(P**2))/Q

            if Q > Q_max:
                Q_max = Q

            while Q > 10**n1:
                n1 = n1 + 10

            if n1>n:
                rtp = N(sqrt(p),digits=n1)
            q = floor((P+rtp)/Q)
            G_old, G = G, ((q*G+G_old) % M)
            B_old, B = B, ((q*B+B_old) % M)
    else:  # fastest loop
        while Q != 2:
            # Subsequent iterations of the recurrence relation
#             num_iterations += 1
            P = (q*Q)-P
            Q = (p-(P**2))/Q

            q = floor((P+rtp)/Q)
            G_old, G = G, ((q*G+G_old) % M)
            B_old, B = B, ((q*B+B_old) % M)

    if test_Q_iterations:
        print('iterations = ',num_iterations, '; Q max = ', Q_max, '; n1 = ', n1)
    return((G_old, B_old))
```

11.2 Sage Code for Calculating $\psi_{even}(x)$ and $\psi_{odd}(x)$

```
load("Pell_Sol_ModM.spyx")

print(sage.parallel.ncpus.ncpus())
os.environ['SAGE_NUM_THREADS'] = '2'
print(sage.parallel.ncpus.ncpus())

@parallel(ncpus=2)
def contribu(k):
    pp = is_prime_power(k,get_data=True)
    if pp[1] > 0:
        p = pp[0]
        P= Pell_sol_mod_M(p,2,0,0)
        if P[0] == 0:
            if pp[1]>1:
                return(log(p), 0, 0, 0)
            else:
                return(log(p), 0, 1, 0)
        else:
            if pp[1]>1:
                return(0, log(p), 0, 0)
            else:
                return(0, log(p), 0, 1)
    else:
        return(0,0,0,0)

t = walltime()
output = []
c = {}
psi_even = 0
psi_odd = 0
pi_even = 0
pi_odd = 0
psi = 0
x_start = 0
x_end = 10^3
N = 100
ch_length = 10^2

print('N = ',N,'; x_end = ', x_end)

```

```
x_unit = ((x_end)-x_start)^(1/N)
x_vals = {}

for i in range(1,N+1):
    x_vals[i] = x_unit^i

for i in range(1,N):

    a1 = floor(x_vals[i])+1
    r = a1%4
    if r == 2:
        a = a1 + 3
    elif r == 3:
        a = a1 + 2
    elif r == 0:
        a = a1 + 1
    else:
        a = a1
    b1 = floor(x_vals[i+1])+1
    s = b1%4
    if s == 2:
        b = b1 + 3
    elif s == 3:
        b = b1 + 2
    elif s == 0:
        b = b1 + 1
    else:
        b = b1

    print('i =',i,'; a = ',a,'; b = ',b)

    aa = a
    bb = min(b, aa + ch_length)
    output = [n for n in range(aa,bb) if n%4==1]
    c = contribu(output)
    for m in c:
        psi_even = psi_even + numerical_approx(m[1][0])
        psi_odd = psi_odd + numerical_approx(m[1][1])
        pi_even = pi_even + m[1][2]
        pi_odd = pi_odd + m[1][3]
```

```
    while bb < b:
        aa = aa + ch_length
        bb = min(b,aa + ch_length)
        output = [n for n in range(aa,bb) if n%4==1]
        c = contribu(output)
        for m in c:
            psi_even = psi_even + numerical_approx(m[1][0])
            psi_odd = psi_odd + numerical_approx(m[1][1])
            pi_even = pi_even + m[1][2]
            pi_odd = pi_odd + m[1][3]

    with open('psi_even10.txt',"a") as f:
        f.write(str((numerical_approx(x_vals[i+1]),psi_even))+',\n')
    with open('psi_odd10.txt',"a") as f:
        f.write(str((numerical_approx(x_vals[i+1]),psi_odd))+',\n')
    with open('pi_even10.txt',"a") as f:
        f.write(str((numerical_approx(x_vals[i+1]),pi_even))+',\n')
    with open('pi_odd10.txt',"a") as f:
        f.write(str((numerical_approx(x_vals[i+1]),pi_odd))+',\n')

print(walltime(t))
```

11.3 Sage Code for Generating Figures 6 and 7

```
with open('psi_even-2021-02-5.txt',"r") as f:
        #f.write(str((numerical_approx(x_vals[i+1]),psi_even)))
    E = [eval(l.strip()) for l in f]
    #L = [l.strip() for l in f]

with open('psi_odd-2021-02-5.txt',"r") as f:
        #f.write(str((numerical_approx(x_vals[i+1]),psi_odd)))
    O = [eval(h.strip()) for h in f]

E_x = []
E_y = []
E_list = []
O_x = []
O_y = []
O_list = []

for i in E:
    E_x.append(i[0][0])
    E_y.append(i[0][1])
    E_list.append((i[0][0],i[0][1]))

for j in O:
    O_x.append(j[0][0])
    O_y.append(j[0][1])
    O_list.append((j[0][0],j[0][1]))

show(list_plot(E_list, plotjoined=true, color="blue")+
     list_plot(O_list, plotjoined=true, color="green"),
     figsize=20, title="Plot of Psi_even and Psi_odd", fontsize=20)

t = 0
comb_list = []
while t < len(E):
    comb_list.append((E_list[t][0],O_list[t][1]-0.5*E_list[t][1]))
    t = t + 1

show(list_plot(comb_list, plotjoined=true, color="red"))
```

11.4 Sage Code for Generating Figure 8

```
PellPsi_eo = [(E_list[i][0], (O_list[i][1]*2 - E_list[i][1])/sqrt(E_list[i][0]))
              for i in range(len(E))]
list_plot([(log(x),y/sqrt(x)) for (x,y) in PellPsi_eo], plotjoined=true)
```

11.5 Sage Code for Generating Figure 9

```
from numpy import arange
z_start = 5
z_end = 100
z_step = 0.1

x_start = 0
x_end = 10^9
N = 1024

x_unit = (x_end-x_start)^(1/N)

M = len(PellPsi_eo)

z = var('z')
dx = numerical_approx(log(x_unit))
print('dx = ', dx)

logx = [numerical_approx(log(x)) for (x,y) in PellPsi_eo]

Fhat_sin = lambda z : sum(PellPsi_eo[i][1]*sin(z*logx[i]) for i in range(M-2))*dx
Fhat_cos = lambda z : sum(PellPsi_eo[i][1]*cos(z*logx[i]) for i in range(M-2))*dx

t = walltime()
Pell_power_spectrum_eo = [(z, Fhat_sin(z)^2 + Fhat_cos(z)^2) for z in arange(z_start,z_end, step=z_step)]

print(walltime(t))

show(list_plot(Pell_power_spectrum_eo, plotjoined=true, color="blue", figsize=30, title="M = "+str(M), fontsize=50))
```