Counting Twin Primes in Residue Classes

Alex Lemann

February 11, 2006

Abstract

It is unknown whether there is an infinite number of twin primes, which are primes differing by two. Another question for which an answer is known for primes, but not for the twin primes, is how they are distributed across residue classes. There are generalized arguments that describe when there are more primes that have one remainder when divided by a particular number than another remainder. For example there are more primes 1 mod 10 than 7 mod 10, that is, end in a 1 than a 7 when written in base 10. For twin primes there are similar questions. Are there more twin primes that end in 1 and 3 or twin primes that end in 7 and 9? We develop an experimental method in order to investigate this question.

Contents

1 Introduction

1.1 The Distribution of Primes in Residue Classes

It is difficult to predict exactly where to find prime numbers. Breaking the integers into subsets and comparing the density of primes in those subsets gives some insight into where the primes fall within the integers. Let us look at the natural numbers mod four. Here the prime numbers are underlined.

- $4x + 0 = 4x = 0, 4, 8, 12, 16, 20, 24, \ldots \equiv 0 \bmod 4$
- $4x + 1 = 1, \underline{5}, 9, \underline{13}, \underline{17}, 21, 25, \underline{29}, 33, \underline{37}, \underline{41}, 45, 49, \underline{53}, 57, \ldots \equiv 1 \bmod 4$
- $4x + 2 = 2(2x + 1) = \underline{2}, 6, 10, 14, 18, 24, 28, \ldots \equiv 2 \bmod 4$
- $4x + 3 = \underline{3}, \underline{7}, \underline{11}, 15, \underline{19}, \underline{23}, \underline{27}, \underline{31}, 35, 39, \underline{43}, \underline{47}, 51, 55, \underline{59}, \ldots \equiv 3 \bmod 4$

This shows the four residue classes mod 4. There is at most one prime that is $4x + 2$ or $4x + 0$ because for $x > 0$ all of the solutions are multiples of two or four respectively. Among the remaining residues there will be an infinite number of prime numbers that will satisfy each equation[Dir37]. Despite this fact, in 1853 Chebyshev noted that within these remaining residues of the natural numbers mod four there are consistently, up to some value, more primes that are of the form $4k+3$ than $4k+1$. This can be seen by graphing the difference between the counts of primes up to x that are 3 mod 4 and those that are 1 mod 4 as in Figure 1. In effect this shows the distance between the lines on a graph which represent the count of primes up to x in a particular residue class. If this distance is ever negative it shows that there are more primes that are in the second term of the subtraction than in the first. The difference is consistently greater than zero meaning there are more primes that are 3 mod 4. In fact, the first value for which the line crosses the x-axis is 26861. Chebyshev's bias is strong for the primes which are 3 mod 4 over 1 mod 4, but comparing other subsets of the integers we do not observe strong biases as in Figure 2. The bias however is never absolute.

It is never the case that there are always more primes of one class up to some value rather than another class[Lit14]. Both of these graphs will cross the x-axis infinitely many times.

This question arose from Chebyshev's experimental observation. Recently there have been analytic results describing in general when there will be more primes congruent to one number rather than another relative to a single modulus[MR94]. Rather than increasing the count by one for each x value that has more primes that are one residue mod the class modulus we increase it by $1/x$. Using this metric it has been found, under the assumption of the Riemann Hypothesis, that whether there is a strong or weak bias is related to whether the residues are quadratic residues for the modulus. That is, if there exists some x, where $0 < x < m$ for some congruence relation modulus m such that $x^2 \equiv q \bmod m$ then q is a quadratic residue modulus m. There will be more primes that are congruent to non-quadratic residues in the class because the squares will always be congruent to quadratic residues.

This begins to scratch the surface of a description of where to look for prime numbers. We will look at the twin primes experimentally with a similar view.

1.2 Twin Primes

In 300 B.C.E. Euclid showed that there are an infinite number of primes. This seems like a good place to start when attempting to describe some set of numbers, asking "How many of them exist?". For twin primes, primes differing by two, nobody knows the answer to this question. Though there have been numerous attempts, there is no complete proof that there are an infinite number of twin primes. A basic questions about twin primes has been left open. Giving some description of how many twin primes there are in different residue classes seems close to giving some explanation of a set which has elluded formal description.

We will approach this question using the weighting described by Rubinstein and Sarnak and determine whether we can use the same guide. Are there more sets of twin primes in pairs of residue classes that are both non-quadratic residues mod m than sets of twin primes which are in pairs of residue classes that are both quadratic residues mod m?

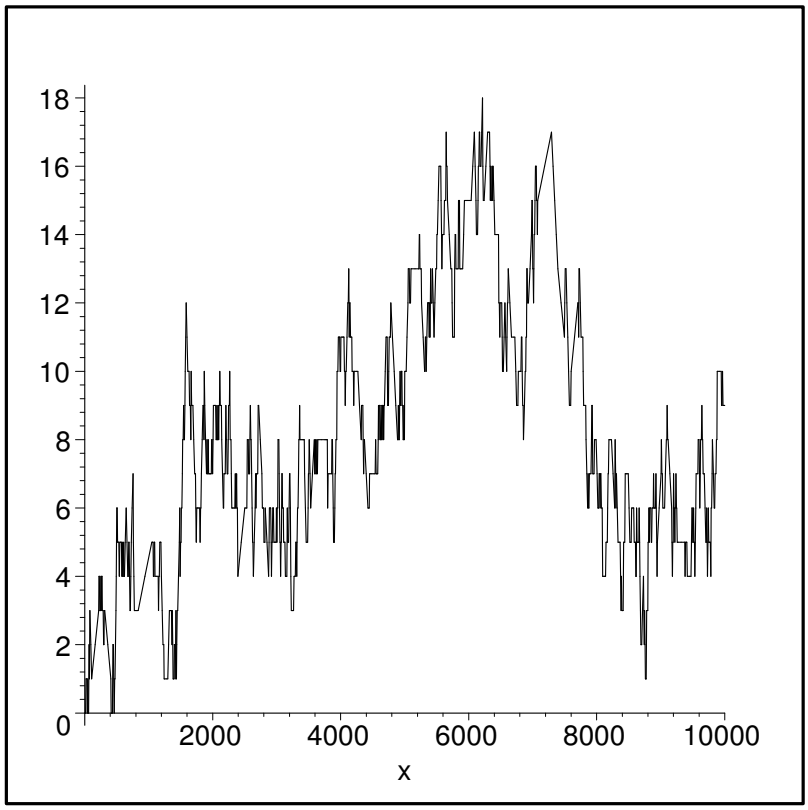

Figure 1: The difference between the number of primes which are 3 mod 4 and 1 mod 4 up to x

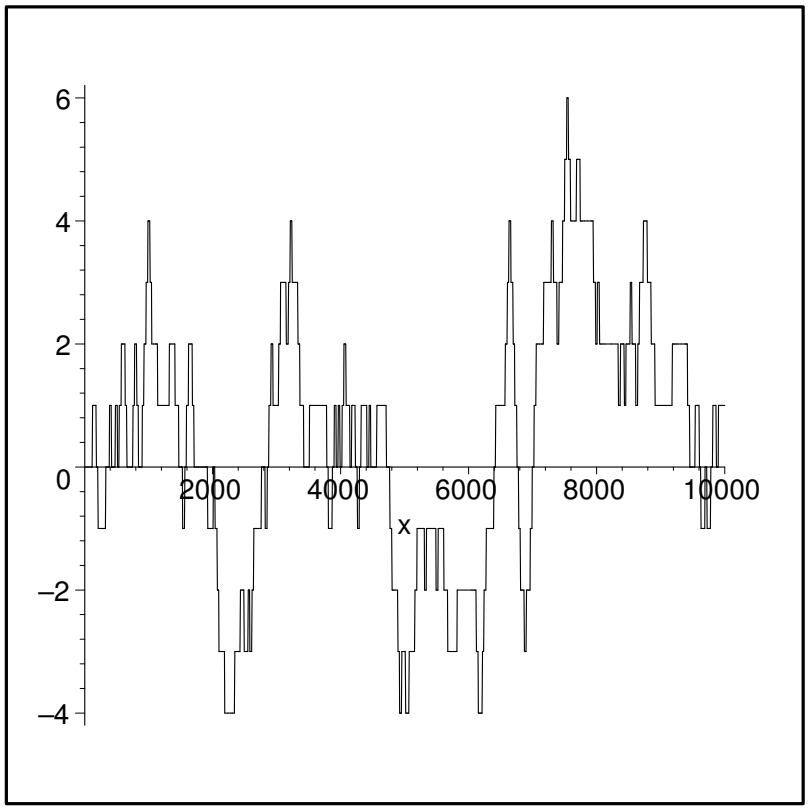

Figure 2: The difference between the number of primes which are 7 mod 30 and 13 mod 30 up to x

2 Computational Methods

2.1 Atkin's sieve

Atkin's sieve is based on a number of theorems similar to "Some integer,p , with unique prime factors of the form 1 mod 4 is prime if and only if the equation $4x^2 + y^2 = p$ has an odd number of solutions." It uses a set of equations like this which together describe all of the integers. Next, it describes a method for finding all solutions to the equations which is based on algorithms for enumerating lattice points on a curve. Using these key principles, Atkin and Berstein develop a method of enumerating all of the primes up to some integer, n using $O(\frac{n}{\log(\log(n))})$ operations and $n^{(1/2)+o(1)}$ bits[AB04]. Berstein has released an implementation of this algorithm, the primegen[Ber99] library which we use to generate primes.

2.2 Parallel approach

In order to decrease the time to calculate the large list of twin primes with which will be working we use a parallel adaptation of the primegen package. This uses principles developed for other embarrassingly parallel applications[WA05]. Since sieving one block of the integers for primes does not relate to sieving any other block of integers the integers can be split up into sections and the sieve operation can be performed on each of these blocks in parallel.

Each node starts and reads in a configuration file. This file specifies the output file name, size of a work unit and the maximum value up to which we will output twin primes. One node is started as the assignment server and the rest as worker clients. The assignment server sends a single value to each worker, the beginning of its work unit. A work unit consists of a portion of the integers from the beginning of a block of integers. These blocks of integers have a fixed size. The workers have already read in the fixed work unit size, so a beginning of the unit is all that is needed to designate the block on which they will sieve.

Each worker node now has a portion of the integers in which to find all of the twin primes. Once the worker has calculated this list of twin primes in its block it sends the list back to the assignment server. The server determines if there are more work units to be assigned. If there are, the server will send a new block and then write the received twin prime list to disk. If there

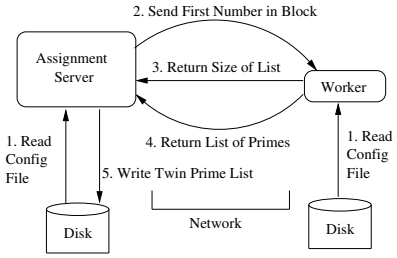

Figure 3: Interactions between the assignment server and one worker. The steps are labeled in order. Steps 2 through 5 repeat until there are no blocks remaining

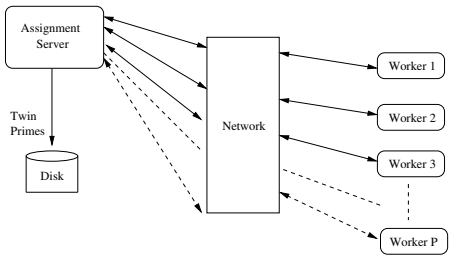

Figure 4: Interactions between the assignment server and workers

are no units left the assignment server sends out a termination message. At this point the client cleans up and terminates. If the work unit size does not evenly divide the number up to which we are listing the twin primes the server will finish by completing the last work unit, the only one that will not have the same fixed size.

2.3 Speedup and Efficiency

The timing data was created on a Beowulf cluster of 10 nodes. Each node has two 500Mhz Pentium III processors and 512MB of primary memory. The network interface card on the nodes have a transfer rate of 100MBit/s and the switch to which the cards are connected is a has 100MBit ports. The data is based on an average of five runs. It is possible to manage the computation versus communication ratio by modifying the block size. For these runs the block size was optimized for 19 worker processors. The speedup factor is calculated by dividing the amount of time it took to run with one worker node by the timing data for a particular number of worker processors. The

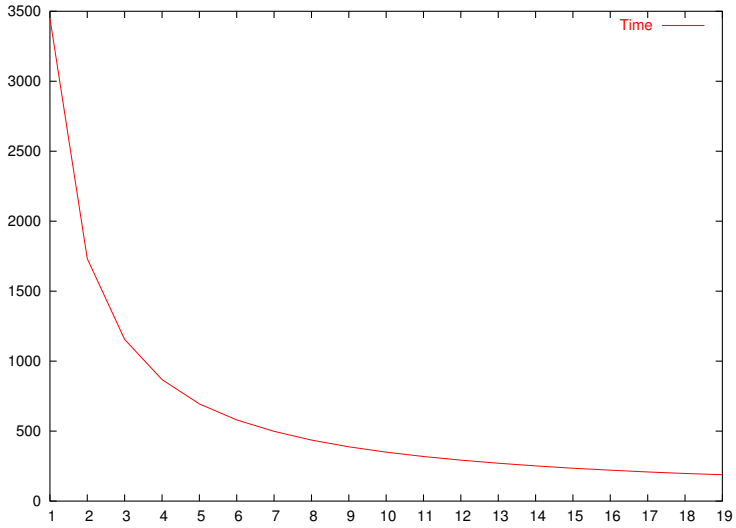

Figure 5: Time in seconds to calculate twin primes out to 10^{11} for 2 to 20 processors

efficiency is the serial time divided by the run time at p processors multiplied by p.

$$\text{Speedup}(p) = \frac{\text{time for one worker}}{\text{time for } p \text{ worker processors}}$$

$$\text{Efficiency}(p) = \frac{\text{time for one worker}}{(\text{time for } p \text{ worker processors}) \times p}$$

2.4 Concise Storage

Reducing the size of the data points we store in turn reduces the amount of time taken to store the output file since disk writes are a probable bottleneck. The assignment server stores a block of prime values when it is returned from the prime generating clients. The clients return the twin prime list in the format which the assignment server will store the list. Since it is already optimized for size, it will take less time to transmit over the network. Furthermore, the assignment server has no work to do on the data other than putting it into a file. If the server had to processes the returned list it would increase the time another client could be waiting to send its list. More

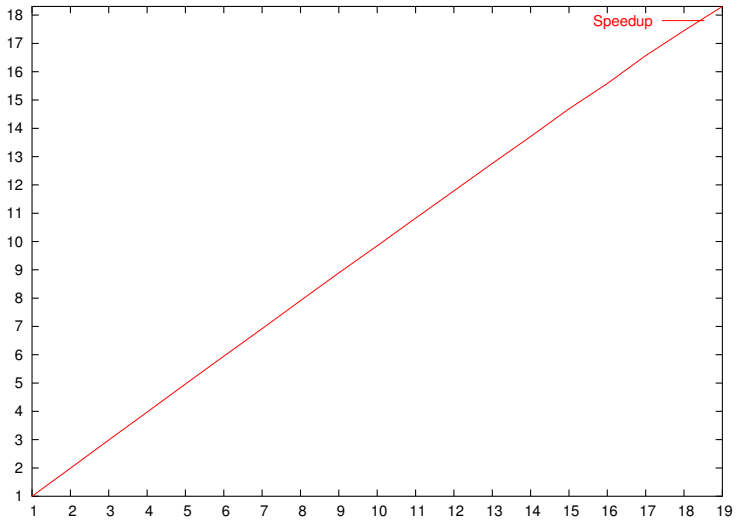

Figure 6: Speedup for 1 to 19 worker processors

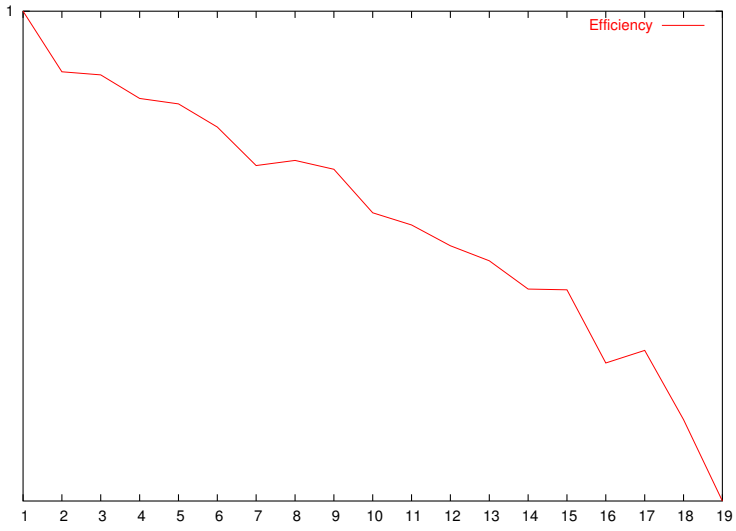

Figure 7: Efficiency for 1 to 19 worker processors

importantly, it would increase the time the client could be waiting to get another block in which to start calculating twin primes.

The size of the storage is reduced first by storing only the smaller prime in a pair of twin primes. We can reverse this process by simply adding 2 to the stored prime value. We know for $\pi_2(n)$ the number of pairs of twin primes up to n, for constants c and Π_2, that[Wei]:

$$\pi_2(n) \leq c\Pi_2 \frac{n}{(\ln n)^2}(1 + O\frac{\ln \ln n}{\ln n}) \approx c\Pi_2 \frac{n}{(\ln n)^2}$$

If we simply stored a list of smaller primes in a set of twin primes, the prime list entries could be stored as 64 bit values, one for each pair of primes. That is that storage(n), the number of bits needed to store the primes up to n, is $64\pi_2(n)$. This is an upper bound on storage using this method.

If, however, we were to store a variable number of bits depending on the difference between the smaller primes in two sets of twin primes we would get a slightly lower storage size[Rie85]. That is for p_k, the kth first prime in a pair of twin primes, we would store $p_k - p_{k-1}$ varying the number of bits used based upon the value of $p_k - p_{k-1}$. Since the client does not know the last prime prior to the block it is working on, the first value stored in the list for a block would always be the actual prime value rather than a difference of primes. In reading back the file, we must know what p_{k-1} is before determining the value p_k. This is not a problem for our purposes since we can iterate through the list sequentially to calculate the weighted counts of primes in residue classes.

Using the estimate of the ith prime as $i \ln i$ and the estimation that twin primes are $\frac{1}{\ln n}$ of the standard primes, we determine that the nth twin prime is equal to about $n \ln(n)^2$. Figure 8 shows that our estimate is close up to a constant factor. We can show the *log*, number of bits stored, of the sum up to n of the differences between the kth and $(k-1)th$ twin prime is about $2n(\ln(\ln n))$. That is:

$$\text{storage}\prime(n) = \sum_{k=1}^{n} (\ln(p_k - p_{k-1})) \approx$$

$$\sum_{k=1}^{n} (\ln(k(\ln k)^2 - (k-1)(\ln(k-1))^2) \approx 2n(\ln(\ln(n))$$

This gives us an improvement over our original method of just storing each twin prime by about $\frac{1}{\ln(n)\ln(ln(n))}\%$. We need to add a fixed width bit

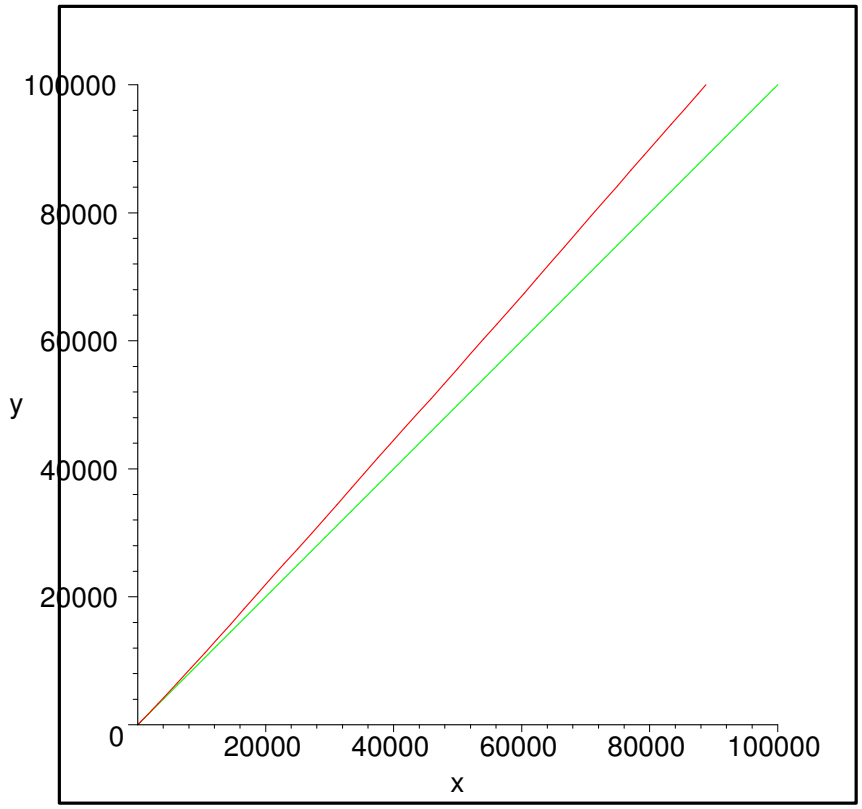

Figure 8: A graph of the number of actual twin primes up to our estimate of the nth twin prime,$1.5n(\log n)^2$, shown in red. $y = x$ is shown as a reference in green.

field of information for each prime record conveying how much storage it will need. In order to accomplish this we use the first two bits of each record as the width of the data field in the record. This gives us four possible sizes of data fields. We chose this based on the C99 defined maximum integer value in C, 64-bits[Int99]. Thus, we use these four possible values to represent the record size as 8, 16, 32, or 64 bits. We must also store a fixed width block size designator at the beginning of each block that is written to disk. This is because we must treat the first record in each block differently since it is not a difference of primes, but an absolute prime value. See Figure 9 for an example of this. Note that in this example of storing two primes and a block size designator we store only 88 bits. If we were storing the full primes we would have to store 64 bits per prime equaling a total of 128 bits without a designator.

A simple method for saving a constant factor would be to divide the differences by six. Each twin prime is either 1 mod 6 or 5 mod 6. We know that the smaller of the twin primes will always be 5 mod 6 so looking at the difference between the lesser of the twin primes: $(6x + 5) - (6y + 5) = 6x - 6y = 6(x - y) \equiv 0 \bmod 6$. Thus, six will divide the difference evenly.

3 Results

3.1 Methods of Generating Graphs

In order to generate weighted counts for the data we produced using the parallel algorithm we must first sort the returned data. Since it was returned from the workers as it was produced it is not guaranteed to be in order already. One node may have other processes taking up CPU cycles which would mean that though it received a block first it may not return its data until much later.

To sort the data we run through the prime list, looking at the beginning of blocks. From that data we produce an list of block headers which contain the first prime in the block, an absolute prime value, the size of the block and the file position of the block header. We can then sort this list of headers in memory by the first prime in the block. If we iterate over this sorted list, pick out file positions, and read from the file, we get the sorted list of twin primes.

The weights are calculated using the MPFR, multi-precision floating point

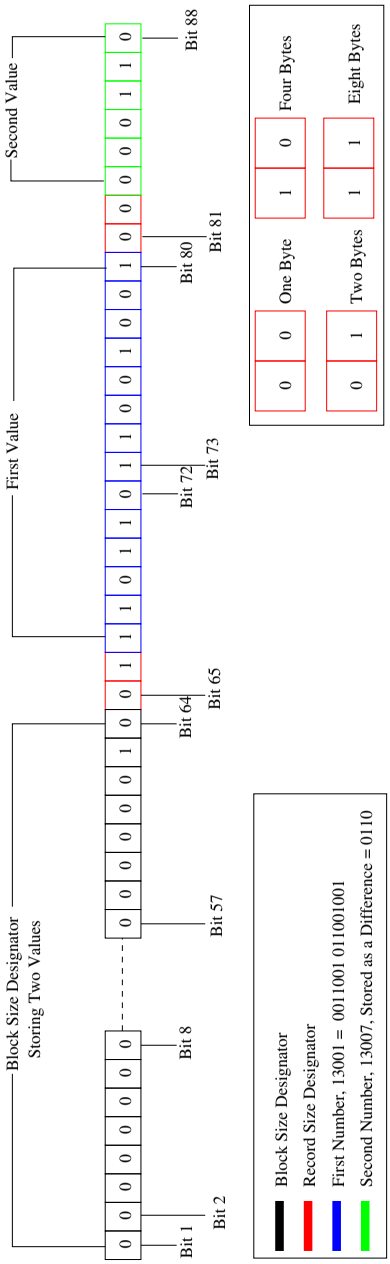

Figure 9: View of a block in bits starting at 13000, storing 13001 and 13007

rounding,[Tea05] library. This allows us to make the precision of the calculated weights a parameter to the analysis program. Rather than calculate $1/x$ for each x where there are more primes in one class than another we approximate the sum of the $1/x$ values between two primes with integration and get the result:

$$\sum_{i=p_k}^{p_{k+1}} \frac{1}{i} \approx \ln(p_{k+1}) - ln(p_k) + \frac{1}{2}(\frac{1}{p_k} - \frac{1}{p_{k+1}})$$

This replaces a number of operations required to calculate the weight at each prime number which was $O(p_{k+1} - p_k)$ with constant time. Figure 10 shows that the estimated weight is close to the actual weight.

3.2 Validation and Verification

The results of the estimated weights were verified to ensure that the estimates being made followed the actual data being produced. The weight estimation equation was validated by graphing the output of the normal weighting[MR94] and the estimated weighting using thirty decimal positions of precision. These graphs looked close for all of the tests we performed. We were able to compare the graphs as far out as 10^7. After this the analysis would take a long time without the estimation.

Other methods were used to validate the code meaning looking for logic errors that may produce erroneous output. This was done through empirical analysis of the output. In order to validate the production of the correct twin primes we first compared small output values to those produced by code written to generate twin primes in Maple. This code was serial and considerably less complex and thus less prone to error. For the large values we compared the counts of twin prime numbers up to various values, again using Maple and verified that various output values were in fact twin primes.

The weight calculations were validated by stepping through the code with a debugger and comparing the output to hand calculated weighted outputs from Maple. The final graphs were compared for small values to graphs generated in Maple.

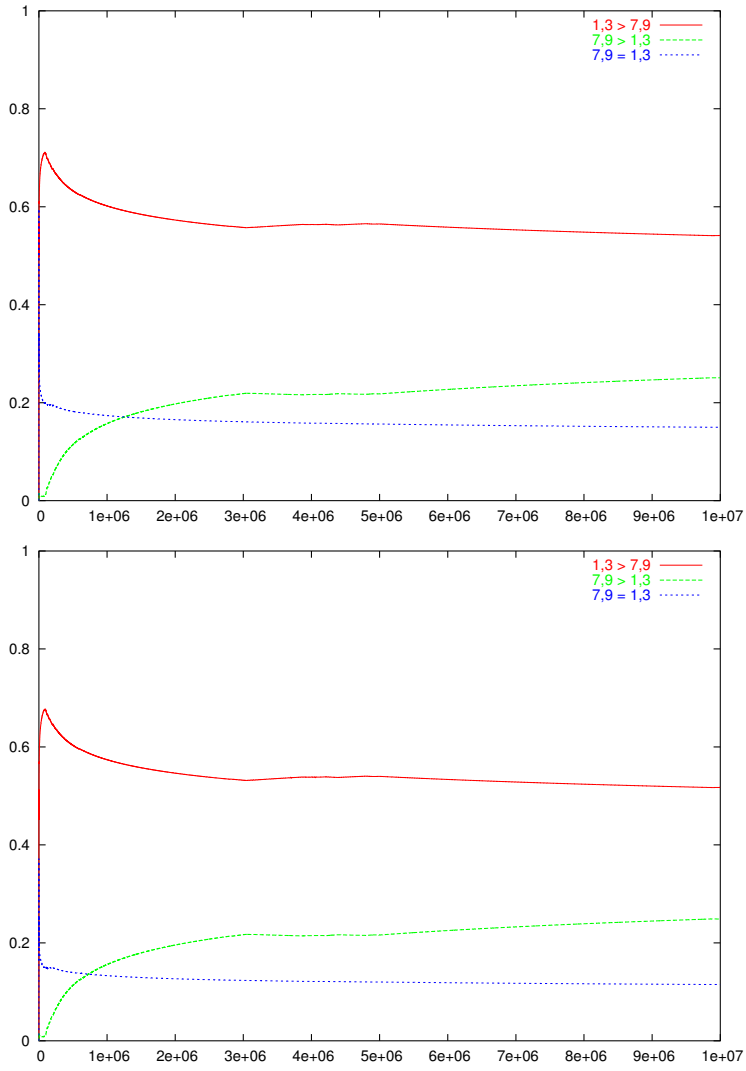

Figure 10: Weighted counts using an estimation(top). And calculating the sum of $\frac{1}{x}$ (bottom).

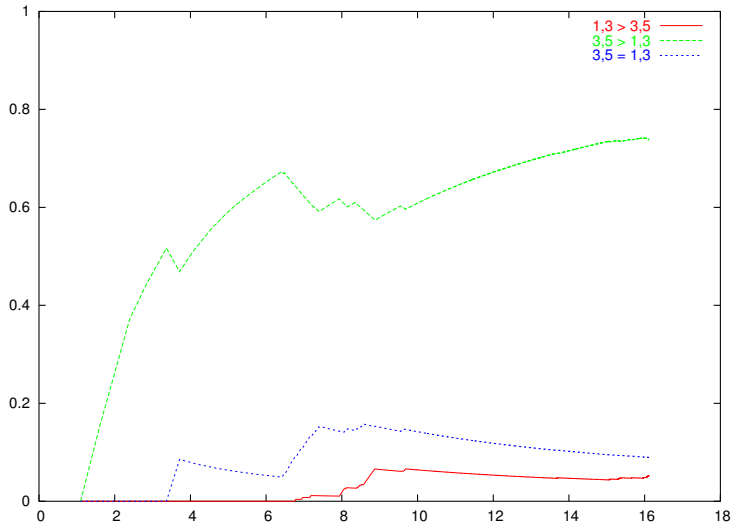

Figure 11: Counts of twin primes that are 1 mod 8 and 3 mod 8 versus those that are 3 mod 8 and 5 mod 8

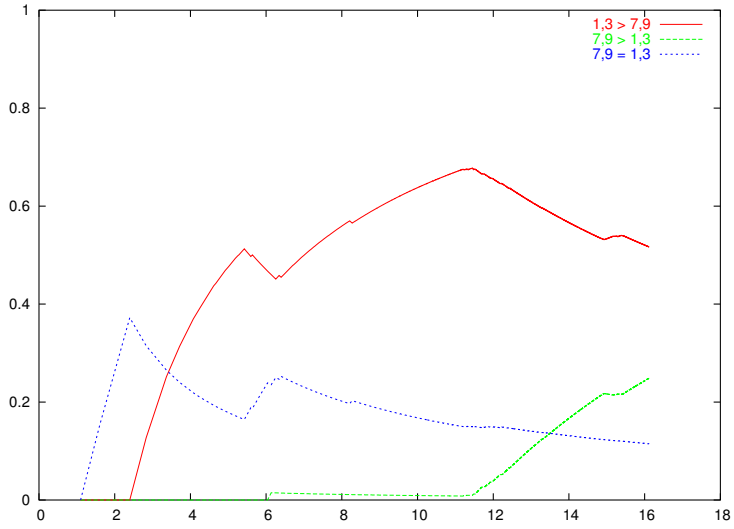

Figure 12: Counts of twin primes that are 1 mod 8 and 3 mod 8 versus those that are 7 mod 8 and 1 mod 8

3.3 Generated Graphs and Analysis

The results of this project are inconclusive. The twin primes are sparse amongst the primes, a set which is already sparse in the integers. In order to get good data it will require going further out along the integers. The data for the counts of twin primes that are 1 mod 8 and 3 mod 8 versus those that are 3 mod 8 and 5 mod 8 seems to be consistent with our expectations1112. The quadratic residues mod 8 are 0, 1, and 4. The count of set twin primes that has an element that is 1 mod 8 seems to be much lower than the set of twin primes in which there is no element congruent to a quadratic residue. This seems to hold for many of the output graphs. Though, counts of twin primes that are 1 mod 8 and 3 mod 8 versus those that are 7 mod 8 and 9 mod 8 seem to not coincide with expectations. Note that $1 \equiv 9 \bmod 8$ and 1 is a quadratic residue. It would seem that since both of these sets of twin primes have an element that is always a quadratic residue we would see them being somewhat equal, but for a large portion of the graph the primes that are 1 mod 8 and 3 mod 8 seem to be ahead. This trend seems to be leveling off as they come close at the end, but this is far from conclusive. More tests need to be run with different moduli and going further out along the number line.

Acknowledgments

- This project is the result of a Discrete II class and an independent study with Tim McLarnan. He helped me work through the background papers which led us to this project. His help is also found within the precise definition of the concise storage, see 2.4 as well as the estimation weighted count, see 3.1.

- Charlie Peck gave advice on parallel computation, concise storage, commented on the paper.

- My CS488 class was helpful in providing feedback on my presentation and abstract. They also lent an extra pair of eyes to my code.

References

[AB04] A. Atkin and D. Bernstein. Prime sieves using binary quadratic forms. *Mathematics of Computation*, 73:1023–1030., 2004.

[Ber99] D.J. Berstein. primegen 0.97. http://cr.yp.to/primegen.html, 1999.

[Dir37] L Dirichlet. Beweis des satzes, dass jede unbegrenzte arithmetische progression, deren erstes glied und differenz ganze zahlen ohne gemeinschaftlichen factor sing, unendlich viele primzahlen erhlt., 1837.

[Int99] International Organization for Standardization. *ISO/IEC 9899:1999: Programming Languages — C*, chapter 5.2.4.2.1. International Organization for Standardization, Geneva, Switzerland, December 1999.

[Lit14] J. E. Littlewood. Distribution des nombres premiers. *C. R. Acad. Sci. Paris*, 158:1869–1872, 1914.

[MR94] Peter Sarnak Michael Rubinstein. Chebyshev's bias. *Experimental Math*, 3:173–197, 1994.

[Rie85] Hans Riesel. *Prime numbers and computer methods for factorization*, chapter 1. Birkhauster, 1985.

[Tea05] MPFR Team. Mpfr 2.2.0. http://www.mpfr.org/, September 2005.

[WA05] Barry Wilkinson and Michael Allen. *Parallel Programming*. Pearson Education, Ltd., Upper Saddle River, NJ 07458, 2005.

[Wei] Eric W. Weisstein. Twin primes. MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/TwinPrimes.html.