University of Washington

Number Theory Final Project

Monte-Carlo Approximation of the Prime Counting Function

Author:
Kevin Stueve

Instructor:
Dr. William Stein

Winter 2010
Last edited March 17, 2010

Abstract

We describe an apparently new method of approximating $\pi(x)$, the prime counting function. We name this method the Monte-Carlo method because of its reliance on pseudorandomness in the distribution of the primes. The Monte-Carlo method combines information from the exact distribution of primes surrounding x with information from multiple samples of an analytic approximation surrounding x. The analytic approximation is based on a sum over the nontrivial zeros of the Riemann zeta function. The Monte-Carlo method requires that the mean error of the analytic approximation approaches zero. The Monte-Carlo method would still work under failure of the Riemann hypothesis. It is plausible that a probabilistic exact $\pi(x)$ implementation could be produced from the Monte-Carlo method. We provide a heuristic argument that the Monte–Carlo method reduces the time complexity of the explicit formula from essentially $O(x^{1.38})$ to essentially $O(x^{\frac{7}{8}})$.

Primes

Integers have unique factorization. A whole number greater than one is prime if it cannot be broken up into the product of two smaller whole numbers, otherwise it is composite. The number one is considered a unit, neither prime nor composite. The prime counting function, $\pi(x)$ is the number of primes less than or equal to a given x. The prime counting function grows without bound and its asymptotic behavior can be studied. We need to be careful about discontinuities (at the primes), so we define $\pi_0(x) = \lim_{\varepsilon\to 0} \frac{\pi(x-\varepsilon)+\pi(x+\varepsilon)}{2}$. Analytic formulas will converge to $\pi_0(x)$ rather than $\pi(x)$. This issue is often glossed over, because usually the error of an analytic approximation for the prime counting function is much greater than 0.5–and the issue can be completely ignored by considering x 0.1 greater than an integer.

Asymptotic Growth of the Prime Counting Function

The prime number theorem says that:
$\pi(x) \sim \frac{x}{log(x)} \sim li(x) \sim Li(x) \sim R(x)$, where:
$li(x) = \int_0^x \frac{dt}{\log(t)}$
$Li(x) = \int_2^x \frac{dt}{\log(t)}$
$R(x) = \sum_{n=1}^{\infty} \frac{\mu(n)}{n} li(x^{1/n})$
$\mu(n) = 1$ if n is a square-free positive integer with an even number of distinct prime factors.
$\mu(n) = -1$ if n is a square-free positive integer with an odd number of distinct prime factors.
$\mu(n) = 0$ if n is not square-free.
We call $li(x)$ the logarithmic integral. $Li(x)$ is the offset logarithmic integral. $R(x)$ is the Riemann prime counting function. $R(x)$ is more accurate than $li(x)$, but harder to calculate. $R(x)$ is less than $li(x)$ by approximately $\frac{li(x^{1/2})}{2}$.

The "Smooth" and Periodic Components of the Prime Counting Function

The best "smooth" (not in the sense of being infinitely differentiable) approximation to $\pi(x)$ is due to Andrey V. Kulsha:
$R(x) - \frac{1}{\log x} + \frac{1}{\pi} \arctan \frac{\pi}{\log x}$
The error of the smooth component of the prime counting function fluctuates between positive and negative, with a mean of zero. The error of the smooth component is called the periodic component of the prime counting function. The periodic component of the prime counting function is
$-\sum_\rho R(x^\rho)$
where the sum is taken over nontrivial zeros of the Riemann zeta function. This sum is only conditionally convergent–the zeros must be taken in order of increasing distance from the origin.

The Explicit Formulas for $\pi(x)$

There is an R-based and a li-based explicit formula for $\pi(x)$.
The R based explicit formula is:
$\pi_0(x) = R(x) - \sum_\rho R(x^\rho) - \frac{1}{\log x} + \frac{1}{\pi} \arctan \frac{\pi}{\log x}$
and the li-based formula is:
$\pi_0(x) = li(x) - \sum_\rho li(x^\rho) - log(2) - \sum_{k=2}^{\infty} \pi_0(x^{1/k}) + \int_x^\infty \frac{dt}{t(t^2-1)log(t)}$.
The R-based formula is more commonly quoted (usually without the $o(1)$ terms) and could be considered more elegant. Computing $\pi(x)$ using the

li-based formula requires recursive $\pi(x)$ calls up to $\sqrt{x}$, but if these are available, the calculations and analysis may be easier than for the R-based formula. Each of the $li(x^{0.5+It}) + li(x^{0.5-It})$ terms in the li-based formula is a sinusoid. Each of the $R(x^{0.5+It}) + R(x^{0.5-It})$ terms in the R-based formula is composed of infinitely many sinusoids and not as simple. The connection between the prime counting function and Fourier analysis comes through better in the li-based formula.
For large x, the prime counting function can be approximated by:
$li(x) - \frac{li(\sqrt{x})}{2} - \sum_\rho li(x^\rho)$

The Nontrivial Zeros of the Riemann Zeta Function and The Riemann Hypothesis

The Riemann Zeta function has trivial zeros at the negative even integers. All of the other zeros, the nontrivial zeros, are in the critical strip $0 < Re(s) < 1$. The Riemann Hypothesis (RH) says that the nontrivial zeros of the Riemann hypothesis all lie on the critical line, $Re(s) = \frac{1}{2}$. The nontrivial zeros come in conjugate pairs. Additional, the locations of the nontrivial zeros must be symmetrical about the critical line. It is known that infinitely many, and at least 40% of the zeros are on the critical line. To verify the Riemann Hypothesis numerically up to some bound, the best we can presently do is simply traverse the critical line. We can count the number of zeros in a region of the critical strip, or along an interval of the critical line. The largest number of nontrivial zeros has been calculated using the Odlyzko–Schönhage algorithm, which uses optimizations to evaluate the Riemann zeta function at many points. The prime number theorem results from no zeros having real part 1. The infinitude of primes results from there being a pole at $s = 1$. RH would mean that the size of the periodic component is bounded by essentially $\sqrt{x}$, where "essentially" means that logarithmic factors are ignored. The number of nontrivial zeros with height below T is given by the Riemann–von Mangoldt formula: $N(T) = \frac{T}{2\pi} \log \frac{T}{2\pi} - \frac{T}{2\pi} + O(\log T)$.

Calculating the Logarithmic Integral Based Explicit Formula

Although the logarithmic integral has an exact series based on the exponential integral, in practice the asymptotic series

$$li(x) \sim \frac{x}{\log x} \sum_{k=0}^{\infty} \frac{k!}{(\log x)^k}$$

can be used. The asymptotic series is not strictly convergent and is only valid if a limited number of terms is used. Below is a demonstration of the accuracy of the asymptotic series for real numbers. The error is ~ 1 (slightly more only for small values). See the code listing in appendix B for the implementation of *li_approx* and *li* by Fredrik Johansson.
Code Listing 1:

```
for n in range(1,12):
...      y = 10**n
...      print "%14.3f %14.3f" % (li_approx(y)-li(2), li(y)-li(2))
...
         8.957          5.120
        31.184         29.081
       177.941        176.564
      1246.918       1245.092
      9630.136       9628.764
     78627.536      78626.504
    664918.780     664917.360
   5762209.464    5762208.330
  50849234.805   50849233.912
 455055614.772  455055613.541
4118066400.588 4118066399.576
```

Tomás Oliveira e Silva provided the present author with an implementation of the *li*–based explicit formula in pari–gp. Of the terms in the li–based explicit formula, the $\sum_\rho li(x^\rho)$ terms are the major bottleneck. The asymptotic series is used for li, and the terms are evaluated in conjugate pairs–there is a function $li(x,t)$ that computes $li(x^{0.5+It}) + li(x^{0.5-It})$. The present author rewrote and optimized for speed this code in C with $li(x^{0.5+It}) + li(x^{0.5-It})$ written in terms of sines and cosines (see the code listing in appendix A).

$$li(x) \sim \frac{x}{\log x} \sum_{k=0}^{\infty} \frac{k!}{(\log x)^k}$$

so

$$li(x^{0.5+It}) + li(x^{0.5-It}) \sim Re(2\frac{x^{0.5+It}}{\log x^{0.5+It}} \sum_{k=0}^{\infty} \frac{k!}{(\log x^{0.5+It})^k})$$

$$= Re(2\frac{\sqrt{x}}{\log x} \frac{[\cos(t \log x) + I \sin(t \log x)]}{\frac{1}{2} + It} \sum_{k=0}^{\infty} \frac{k!}{(\log x^{0.5+It})^k})$$

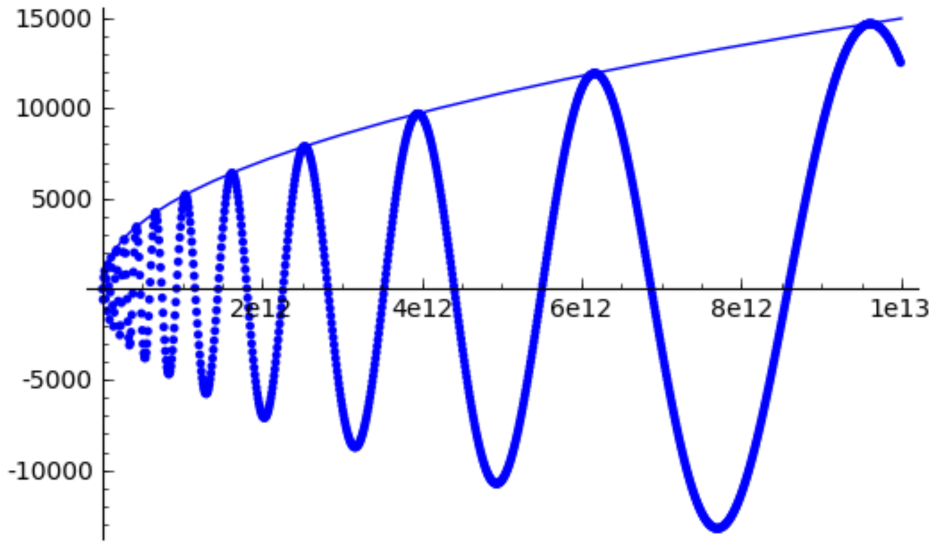
Note that we have symmetry about the real axis and that the imaginary components cancel out. It is worthwhile to study the asymptotic properties of $li(x^{0.5+It}) + li(x^{0.5-It})$ for large x.
For large x,

$$\sum_{k=0}^{\infty} \frac{k!}{(\log x^{0.5+It})^k}$$

will approach one.
We are left with $li(x^{0.5+It}) + li(x^{0.5-It}) \sim Re(2\frac{\sqrt{x}}{\log x}\frac{[\cos(t\log x)+I\sin(t\log x)]}{\frac{1}{2}+It}) = 2\frac{\sqrt{x}}{\log x}\frac{0.5\cos(t\log x)+t\sin(t\log x)}{0.25+t^2} \sim 2\frac{\sqrt{x}}{\log x}\frac{\sin(t\log x)}{t}$.
Now we can see a hint of why RH implies that the error of the smooth approximations is bounded by essentially $\sqrt{x}$–the real part of the nontrivial zeros translates directly into the exponent in the explicit formula. $\sum_\rho li(x^\rho)$ is in some sense very roughly an infinite series of sinusoids in the logarithmic domain where the amplitudes are (assuming RH) $\frac{2\sqrt{x}}{t\log x}$ and the frequencies are $\frac{t}{2\pi}$, the heights of the nontrivial zeros of the Riemann zeta function divided by 2π. Below is a plot of $li(x^{0.5+It}) + li(x^{0.5-It})$ for the first nontrivial zero, along with the amplitude, $\frac{2*sqrt(x)}{tlog(x)}$. Take special note of the exponential change of variable.
Code Listing 2:

```
 def f(x):
    return li(x,zz[1])
li_list=[]
for x in range(int(10),int(100*10**11),int(10**10)):
    li_list.append((x,f(x)))
def g(x):
    return 2*sqrt(x)/zz[1]/log(x)
list_plot(li_list)+plot(g,(1,100*10**11))
```

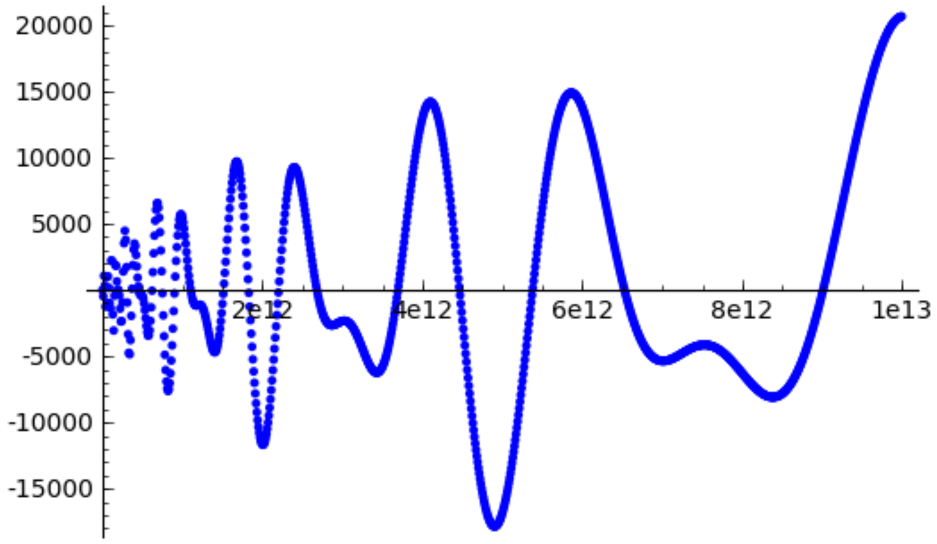
Figure 1:



The perfect symmetry is broken as soon as we add the second term:
Code Listing 3:

```
 def f(x):
    return li(x,zz[1])+li(x,zz[2])
li_list=[]
for x in range(int(10),int(100*10**11),int(10**10)):
    li_list.append((x,f(x)))
list_plot(li_list)
```

Figure 2:



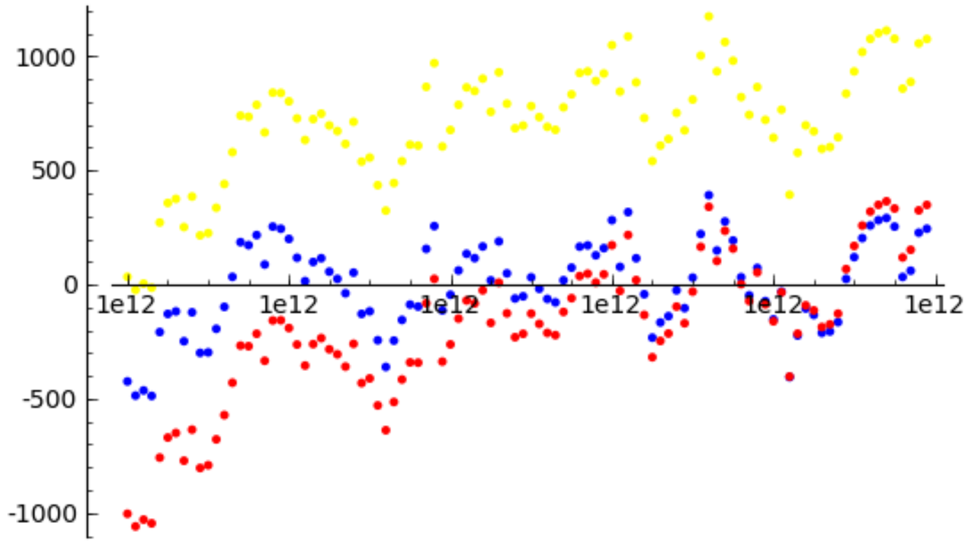
With more and more terms, we will more closely approximate the prime counting function. We may call the explicit formula the analytic approximation when truncated after some finite number of terms.

The Accuracy of the Logarithmic Integral Based Explicit Formula

Below are plots of the error of the logarithmic integral based explicit formula with 1000 (yellow), 10000 (red), and 100000 (blue) zeros. It can be seen that the error decreases when more zeros are used.
Code Listing 4:

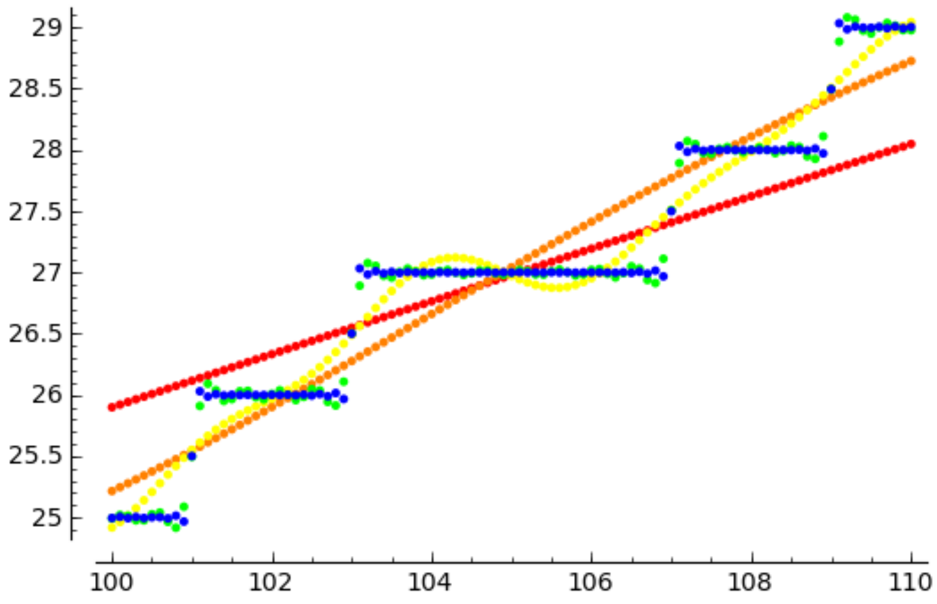
```
 l=[]
l2=[]
l3=[]
start=int(10^12)
step=int(10^6)
stop=int(start+step*100)
for x in range(start,stop,step):
    l.append((x,riemann_pi(x,100000)-prime_pi(x)))
    l2.append((x,riemann_pi(x,10000)-prime_pi(x)))
    l3.append((x,riemann_pi(x,1000)-prime_pi(x)))
list_plot(l)+list_plot(l2,color="red")+list_plot(l3,color="yellow")
```

Figure 3:



Below are plots of the logarithmic integral based explicit formula with 0 (red), 10 (orange), 100 (yellow), 1000 (green), and 10000 (blue) zeros. Notice the convergence to the true prime counting function.

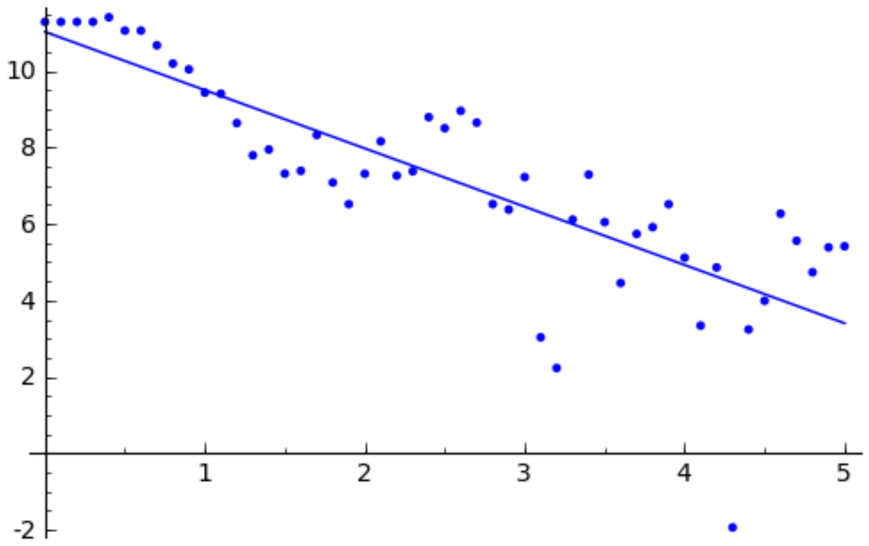
Figure 3b:



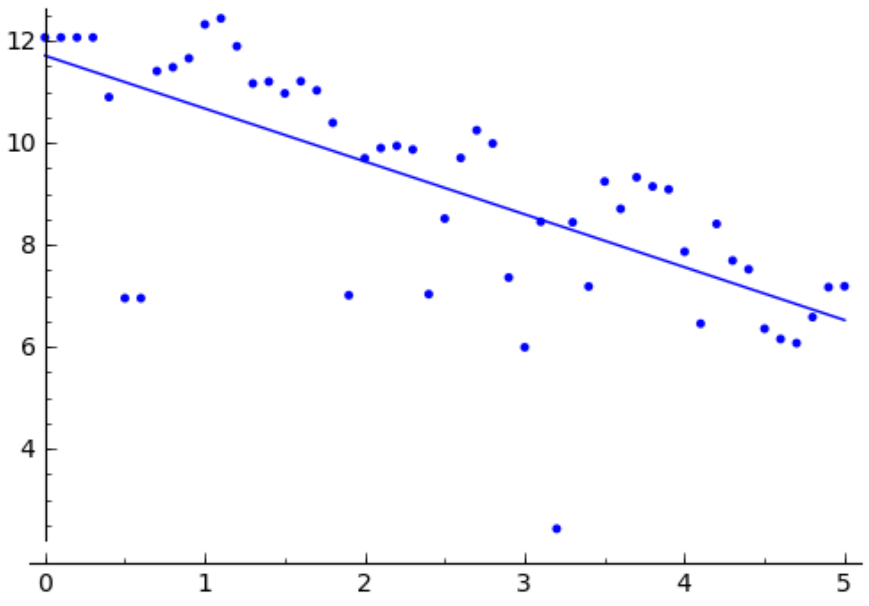
Let's see how the error decreases as the number of zeros increases.

Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to

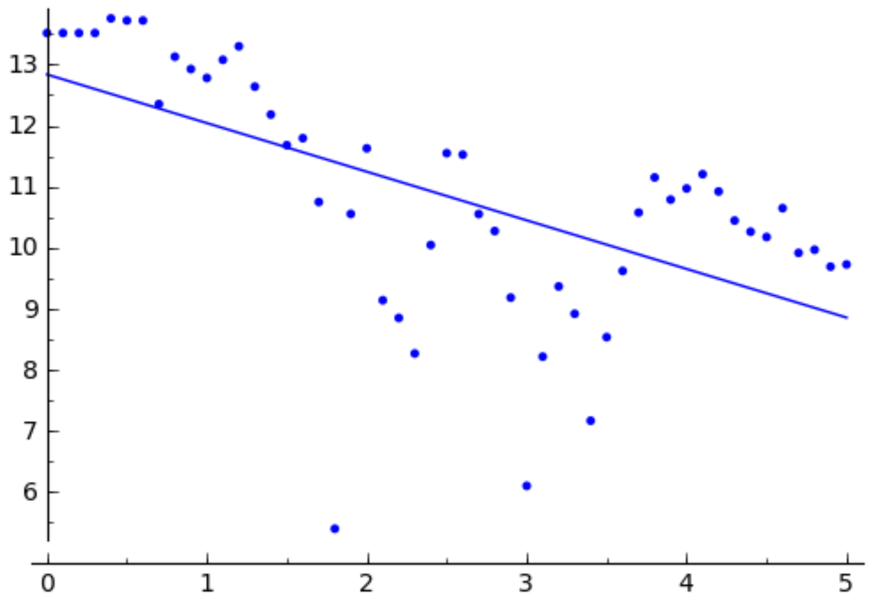
store the error of the approximation as a function of the base 10 logarithm of the number of zeros used for $x = 10^{10}$:
Line of best fit: $y = -1.5253066915024995x + 11.028726382380231$
Increasing the number of zeros by a factor of ten decreases error by a factor of 2.87.
Figure 4:



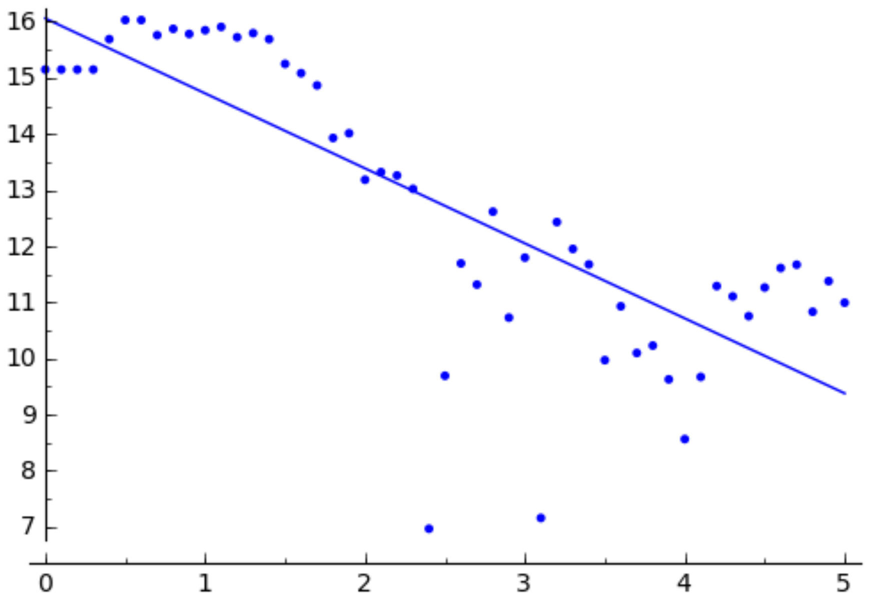
Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the approximation as a function of the base 10 logarithm of the number of zeros used for $x = 10^{11}$:
Line of best fit: $y = -1.0368869521503168x + 11.705300338338557$
Increasing the number of zeros by a factor of ten decreases error by a factor of 2.05.
Figure 5:



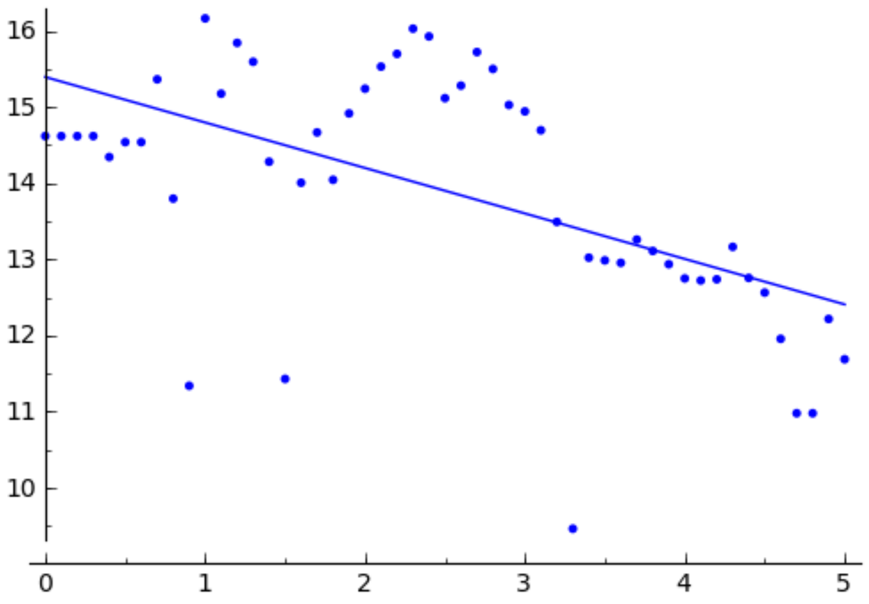
Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the approximation as a function of the base 10 logarithm of the number of zeros used for $x = 10^{12}$:
Line of best fit: $y = -0.7960336635220191x + 12.837456191781575$
Increasing the number of zeros by a factor of ten decreases error by a factor of 1.73.
Figure 6:



Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the approximation as a function of the base 10 logarithm of the number of zeros used for $x = 10^{13}$:
Line of best fit: $y = -1.3366081667372423x + 16.060337604210417$
Increasing the number of zeros by a factor of ten decreases error by a factor of 2.53.
Figure 7:



Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the approximation as a function of the base 10 logarithm of the number of zeros used for $x = 10^{14}$:
Line of best fit: $y = -0.59705527779850254x + 15.392726387621201$
Increasing the number of zeros by a factor of ten decreases error by a factor of 1.51.
Figure 8:

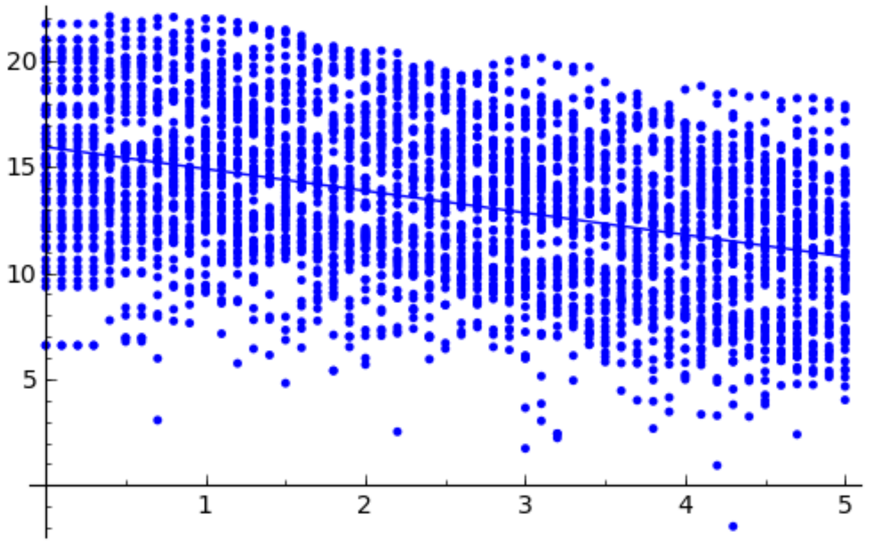


The benefit of using additional zeros seems to depend strongly on the value of x. Let's calculate an average slope for many values of x. See Appendix C for the linear regression Code used.
Code Listing 5:

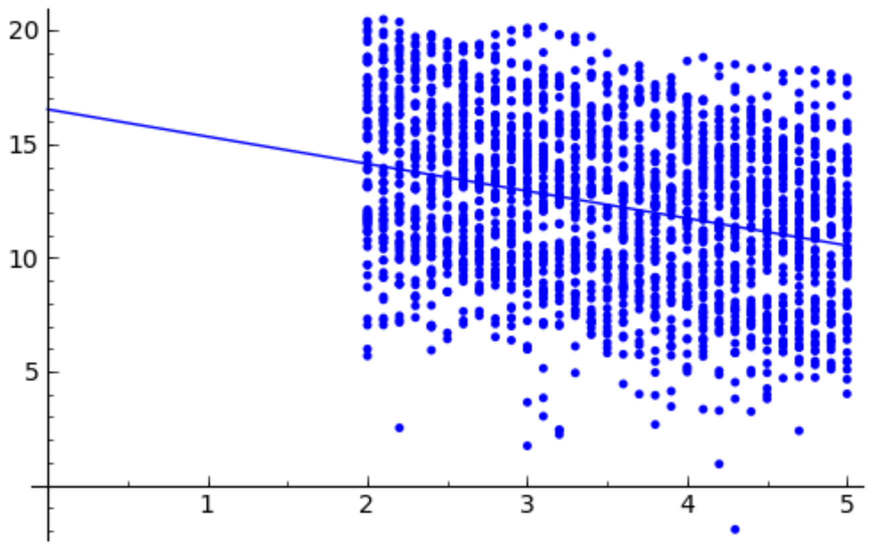
```
 data=[]
list1=[10^10,10^11,10^12,10^13,10^14,10^15,10^16]
list2=[]
for x in list1:
    for multiplier in (1,2,3,4,5,6,7,8,9):
        list2.append(x*multiplier)
for n in (0,.1,..5):
    for x in (list2):
        data.append((n,log(abs(riemann_pi(x,10^n)-prime_pi(x)))/log(2)+1))
thefit=polyfit(data,1)
a=thefit['polynomial'][0]
b=thefit['polynomial'][1]
def f(x):
    return a*x+b
list_plot(data)+plot(f,(0,5))
```

Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the approximation as a function of the base 10 logarithm of

the number of zeros used. Values of x between 10^{10} and $9*10^{16}$ and numbers of zeros between 1 and 10^5 are used:
Line of best fit: $y = -1.02884846072x + 15.8755002601$
Increasing the number of zeros by a factor of ten decreases error by a factor of 2.04.
Figure 9:



Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the approximation as a function of the logarithm of the number of zeros used. Values of x between 10^{10} and $9*10^{16}$ and numbers of zeros between 100 and 10^5 are used (to remove outliers from the previous plot):
Line of best fit: $y = -1.24079579286x + 16.6412116696$
Increasing the number of zeros by a factor of ten decreases error by a factor of 2.36, consistent with Patrick Demichel's value of 2.3:
Figure 10:

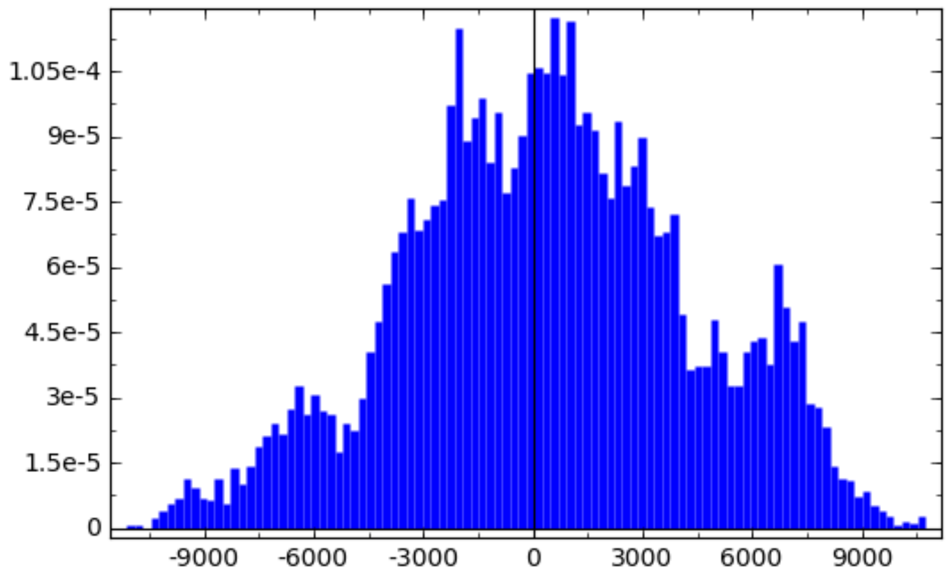


With this figure, we can make the very very rough heuristic estimate that to decrease the error of the li-based analytic approximation from essentially $\sqrt{x}$ to about 0.5, we need essentially about $O(x^{1.38})$ terms–not a very efficient way to calculate the prime counting function.

The Monte-Carlo Approximation of the Prime Counting Function

Wikipedia says:
"Monte Carlo methods are a class of computational algorithms that rely on repeated random sampling to compute their results."
The Monte–Carlo method takes advantage of the observation that the error of the analytic approximation is normally distributed with mean 0. By using a formula that samples the error of the analytic approximation, we can obtain better accuracy from the explicit formula with the same number of zeros.
Tomás Oliveira e Silva says:
"Assuming that the values of t log(x) are uniformly distributed modulo 2pi, where t is, as before, the imaginary part of the zero, it is possible to model the terms not used in the formula as a Gaussian random variable with zero mean and an approximate standard deviation which is easy to compute. Unfortunately, this standard deviation decreases slowly with the number of zeros used, to the point that one hundred times the number of zeros gives us less than one extra digit of precision."

Patrick Demichel says:
"The worst theoretical error is much greater, but will occur only if a large number of the millions of sinusoids would coincide in one place. Otherwise they have a natural tendency to cancel each other. Thats why the worst observed error, is orders of magnitude smaller than the worst theoretical maximum. Then without any special property of the distribution of the zeros, we can consider the values as random. And as it seems to be the case, the errors distribution follows perfectly the normal distribution."
Below is a histogram of the error of the li-based analytic approximation without any of the zeros being used for 8000 values of x just below 10^{12}. The values of x are in an arithmetic progression separated by a stepsize 10^7:
Figure 10b:



By evaluating the analytic approximation at many points in an interval around some x, and also calculating the fluctuations of the prime counting function in that interval (with local sieving), we can obtain a more accurate approximation of the prime counting function than the analytic approximation. Statistics says that if the number of samples from a normal distribution increases by a factor of m, then the standard error of the mean decreases by a factor of $\sqrt{m}$. In practice we might not obtain such good results. Although a normally distributed error in the analytic approximation may allow faster convergence, the Monte–Carlo method does not require a normally distributed error, it will work as long as the mean error approaches zero for a larger number of samples.
Define:

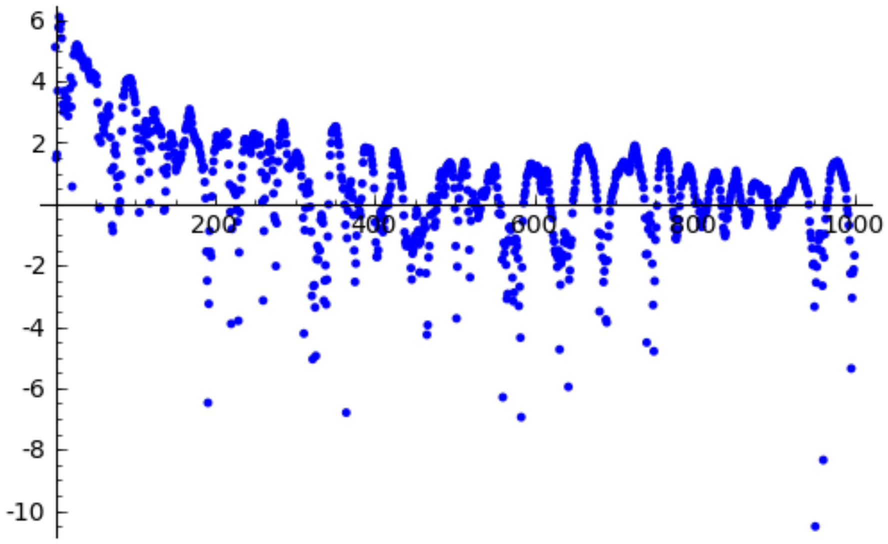
$Li_n(x) = li(x) - log(2) - \sum_{k=2}^{\infty} \pi(x^{1/k}) + \int_x^{\infty} \frac{dt}{t(t^2-1)log(t)} - \sum_{\rho} li(x^{\rho})$ (over the first n nontrivial zeros) so that
$\pi_0(x) \approx Li_n(x)$ and
$\pi_0(x) = Li_n(x) + E(x)$
where $E(x)$ is the error of $Li_n(x)$.
Algorithm:
1) Determine $\pi(x_1) - \pi(x_0)$, $\pi(x_2) - \pi(x_1)$, $\pi(x_3) - \pi(x_2)$, etc for many values in an interval around x.
2) Use the formula:
$E(b) - E(a) = \pi(b) - \pi(a) - Li_n(b) + Li_n(a)$ to calculate $E(x_1) - E(x_0)$, $E(x_2) - E(x_1)$, $E(x_3) - E(x_2)$, etc for many values in an interval around x.
3) Use some x as a reference point. Calculate $E(x) - E(x_0)$, $E(x) - E(x_1)$, $E(x) - E(x_2)$, $E(x) - E(x_3)$, etc based on the data from the previous step. Calculate the mean of this list of numbers and call it $E_{est}(x)$.
Because the mean of $E(x)$ approaches zero for larger intervals, the mean we just calculated is an approximation to $E(x)$.
4) Use $Li_n(x) + E_{est}(x)$ as an approximation to $\pi_0(x)$ that is better (on average) than $Li_n(x)$.
Note that it is really $E(x) * \frac{log(x)}{\sqrt{x}}$ that has mean approaching zero, not $E(x)$, so it is desirable for the interval used to be small, ideally on the order of $\sqrt{x}$ so that the change in $\frac{log(x)}{\sqrt{x}}$ over the interval is negligible–that way it is valid to consider $E(x)$.

The Accuracy of the Monte-Carlo Approximation

The error of the Monte-Carlo Approximation is the error in $E_{est}(x)$, i.e. the error is the value $E_{est}(x) - E(x)$.
$E_{est}(x) = Avg[E(x) - E(x_i)] = Avg[\pi_0(x) - \pi_0(x_i) - Li_n(x) + Li_n(x_i)]$ where the average is over i.
Now $E_{est}(x) = Avg[E(x) - E(x_i)] = E(x) - Avg[E(x_i)]$, so
$E_{est}(x) - E(x) = -Avg[E(x_i)]$
In other words, the error of the Monte–Carlo approximation is the average of the error of the analytic approximation. This is why it is important for the analytic approximation used to have mean error 0. This reliance on the pseudorandom properties of the prime counting function is the reason this method is given the name the Monte–Carlo approximation.
Let's look at some examples of the accuracy of the Monte–Carlo approximation.
Code Listing 6:

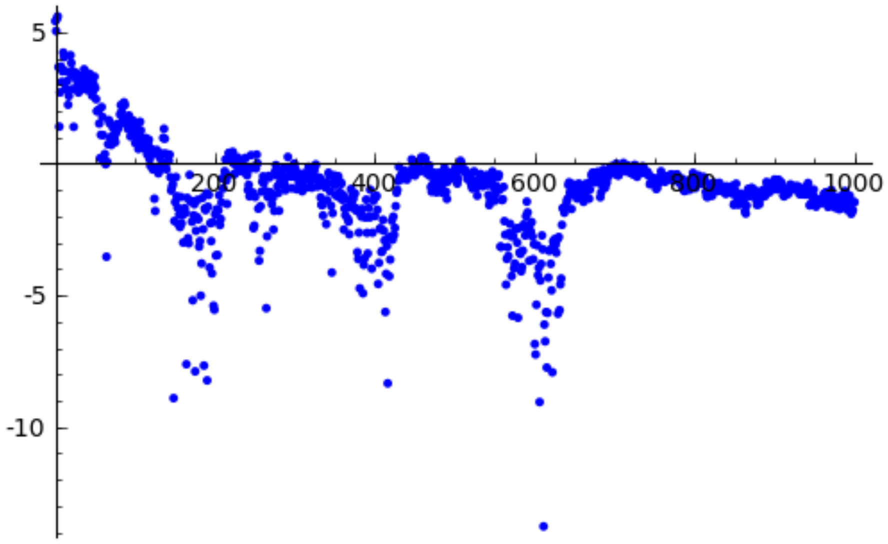
```
 start=int(10^10)
stepsize=int(10^5)
numchanges=0
last=0
differences=[]
plotpoints=[]
for steps in range(0,1000+1,1):
    x=start-steps*stepsize
    difference=riemann_pi(x,100000)-prime_pi(x)
    if value*last<0:
        numchanges+=1
    differences.append(difference)
    plotpoints.append((steps,log(abs(mean(differences)))/log(2)+1))#add 1 for
sign bit
list_plot(plotpoints)
```

Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{10})$ as a function of the number of evaluations (separated by 10^5) of $Li_{10^4}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^5) required to calculate this approximation:
Figure 11:

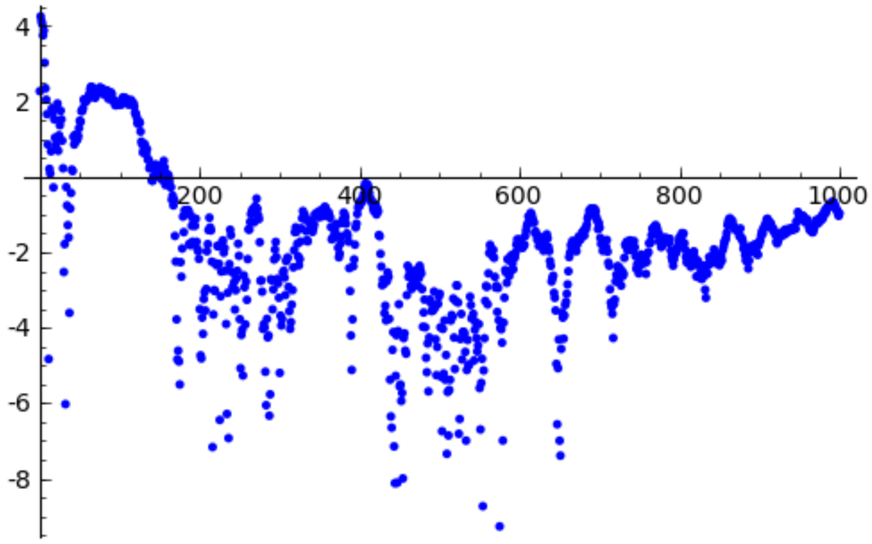


Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{10})$ as a function of the number of evaluations (separated by 10^5) of $Li_{10^5}(x_i)$–the x–axis also

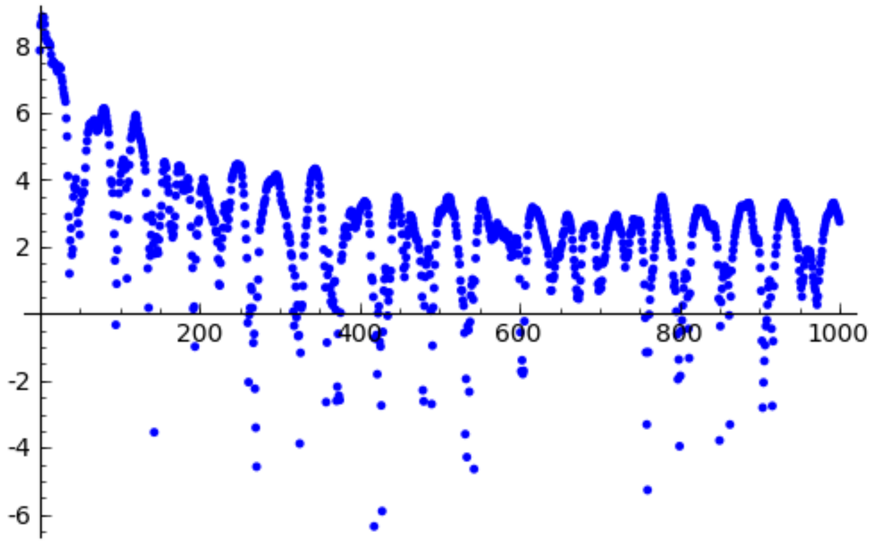
represents the amount of sieving (in units of 10^5) required to calculate this approximation:
Figure 12:



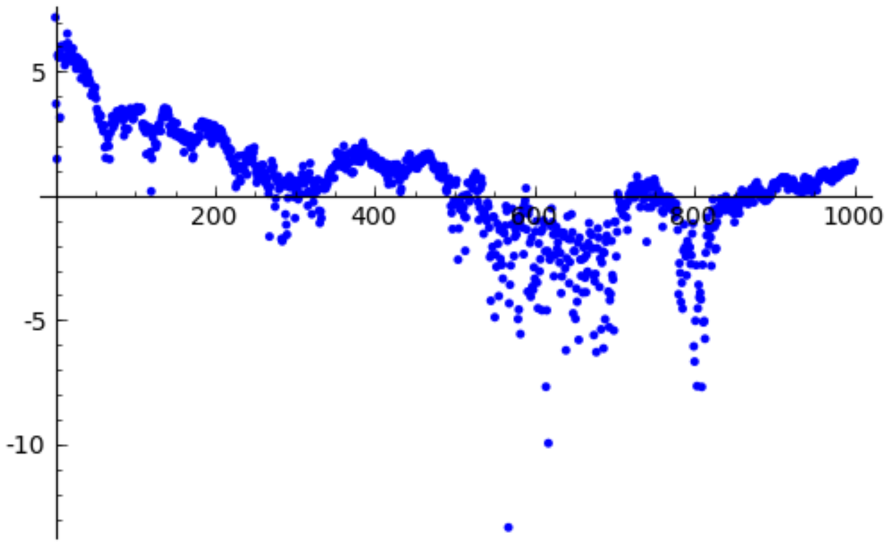
Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{10})$ as a function of the number of evaluations (separated by 10^5) of $Li_{10^6}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^5) required to calculate this approximation:
Figure 13:



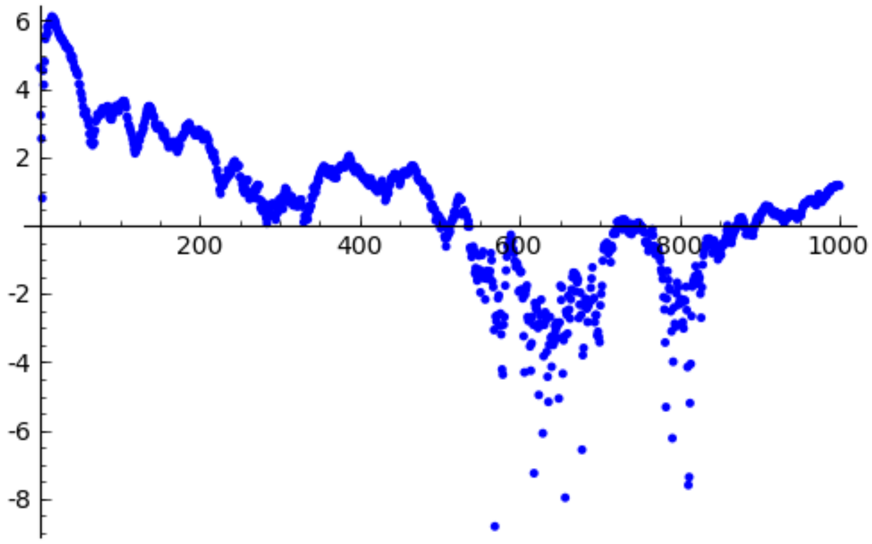
Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{11})$ as a function of the number of evaluations (separated by 10^6) of $Li_{10^4}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^6) required to calculate this approximation:
Figure 14:



Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{11})$ as a function of the number of evaluations (separated by 10^6) of $Li_{10^5}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^6) required to calculate this approximation:
Figure 15:

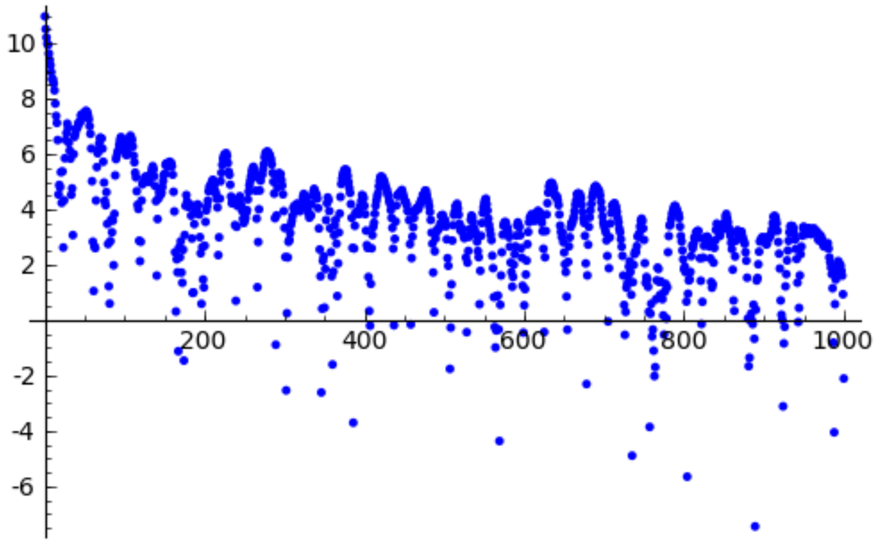


Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{11})$ as a function of the number of evaluations (separated by 10^6) of $Li_{10^6}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^6) required to calculate this approximation:
Figure 16:

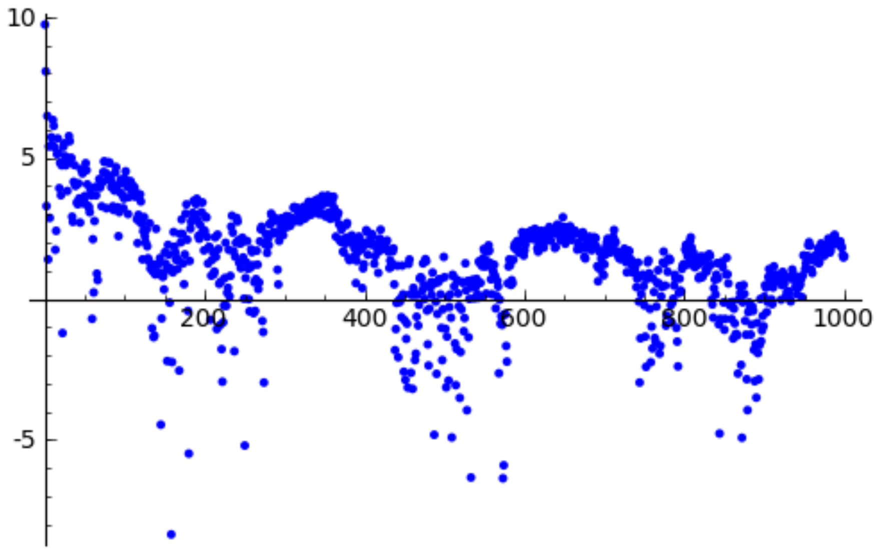


Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{12})$ as a function

of the number of evaluations (separated by 10^7) of $Li_{10^4}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^7) required to calculate this approximation:
Figure 17:

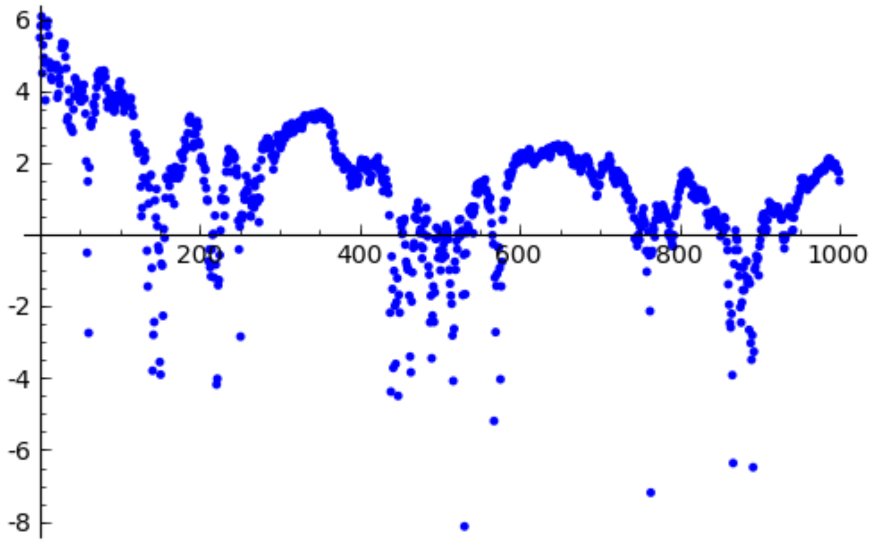


Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{12})$ as a function of the number of evaluations (separated by 10^7) of $Li_{10^5}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^7) required to calculate this approximation:
Figure 18:

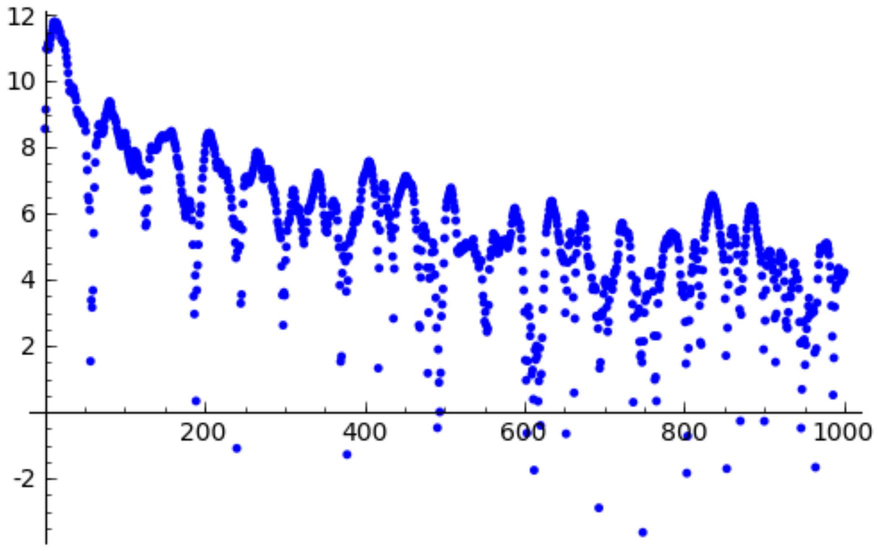


Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{12})$ as a function of the number of evaluations (separated by 10^7) of $Li_{10^6}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^7) required to calculate this approximation:

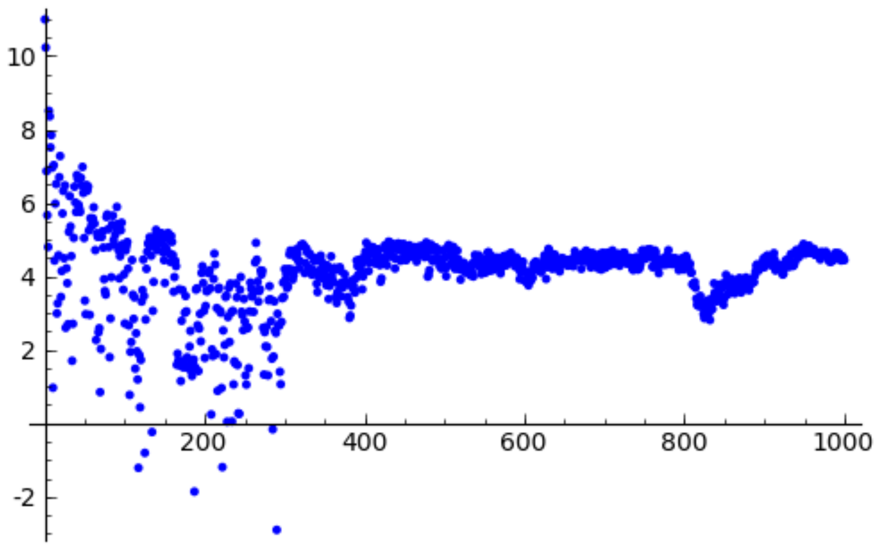
Figure 19:



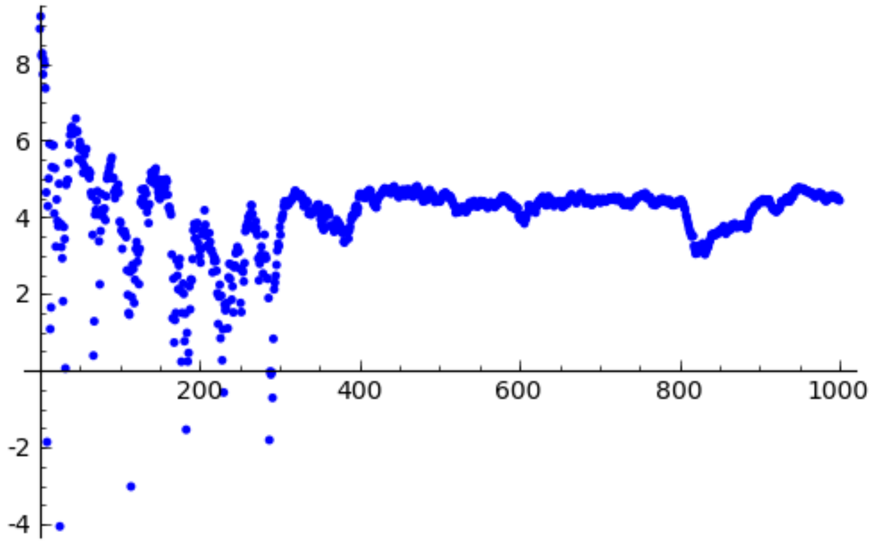
Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{13})$ as a function of the number of evaluations (separated by 10^8) of $Li_{10^4}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^8) required to calculate this approximation:
Figure 20:



Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{13})$ as a function of the number of evaluations (separated by 10^8) of $Li_{10^5}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^8) required to calculate this approximation:
Figure 21:

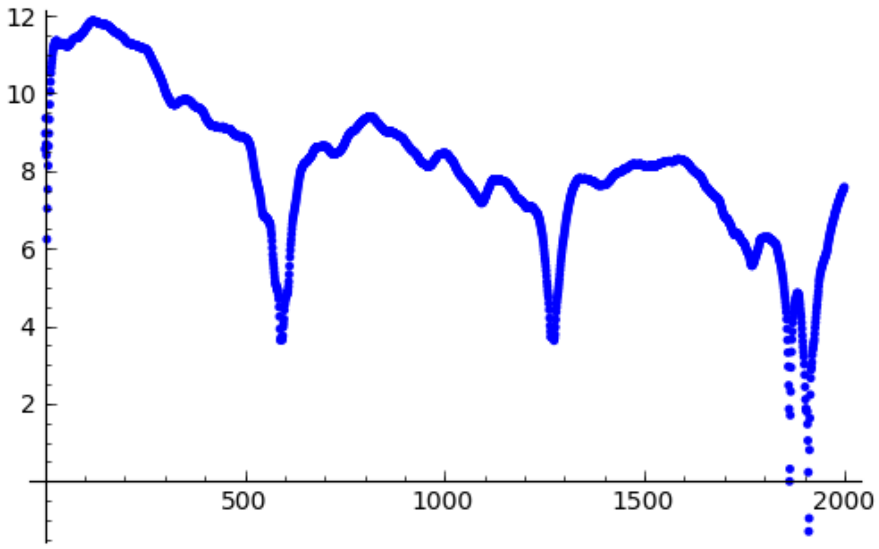


Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{13})$ as a function of the number of evaluations (separated by 10^8) of $Li_{10^6}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^8) required to calculate this approximation:
Figure 22:



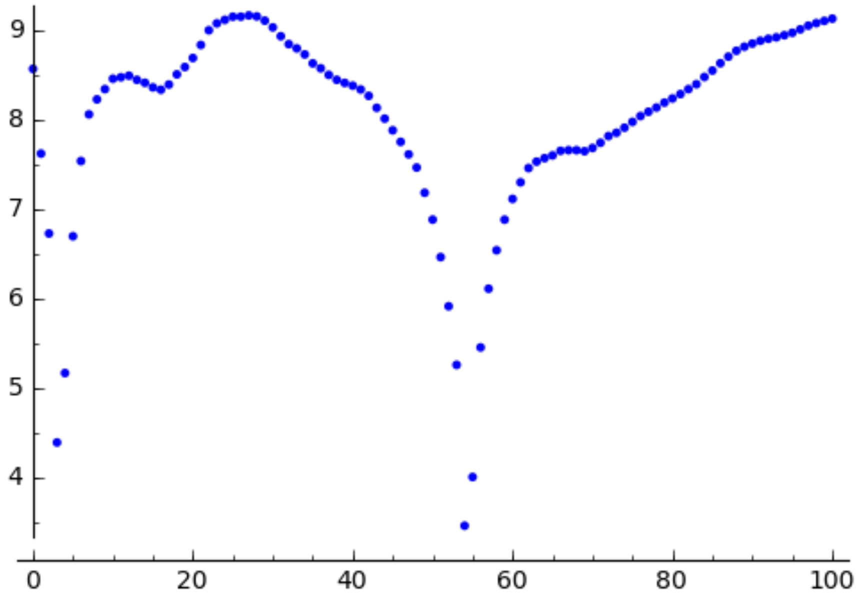
The error of the Monte–Carlo method with respect to the size of the interval used for the sieving and analytic approximation evaluations has an

apparent periodic structure. For simplicity and ease of calculation, we will direct our focus on the approximation based on $Li_{10^4}(x_i)$.
Let's take a closer look at Figure 20.
Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{13})$ as a function of the number of evaluations (separated by 10^7) of $Li_{10^4}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^7) required to calculate this approximation:
Figure 23:

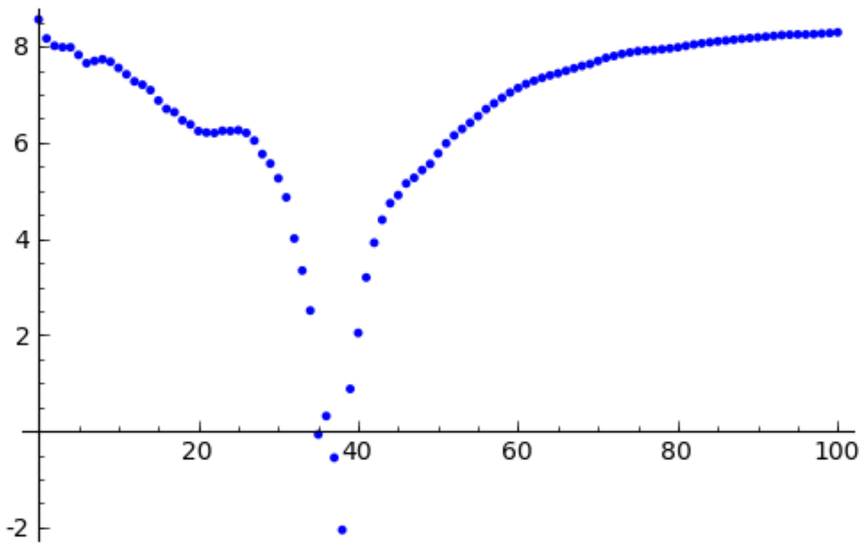


This periodic structure comes from the periodic nature of the error of $Li_{10^4}(x)$. The error of $Li_{10^4}(x)$ is equal to the contributions from the zeros past the first 10^4. Patrick Demichel has observed that "In practice a higher group of zeros 10 times bigger tends to offer 2 times smaller contribution to the sum". The main contributors to the error of $Li_{10^4}(x)$ are from the sinusoids from the first zeros after the first 10^4. The period of each sinusoid is $\frac{2\pi}{t}$ (in the logarithmic domain), i.e. one wavelength is completed each time $\log x$ increases by $\frac{2\pi}{t}$. The value of zero number $10^4 + 1$ is 9878.65477238, so one wavelength of the contribution from this zero is completed each time $\log x$ increases by 0.000630365305. Because the code in code listing 6 sieves down from 10^{13}, we calculate that the contribution from zero $10^4 + 1$ completes its first period after an interval of size $6.358343021 * 10^9$. This is reasonably close to the first wavelength completion between $5 * 10^9$ and $6 * 10^9$ (between tick-marks 500 and 600 in figure 23. It is also an overestimate, which is reasonable considering that the periods of the sinusoids are smaller for later

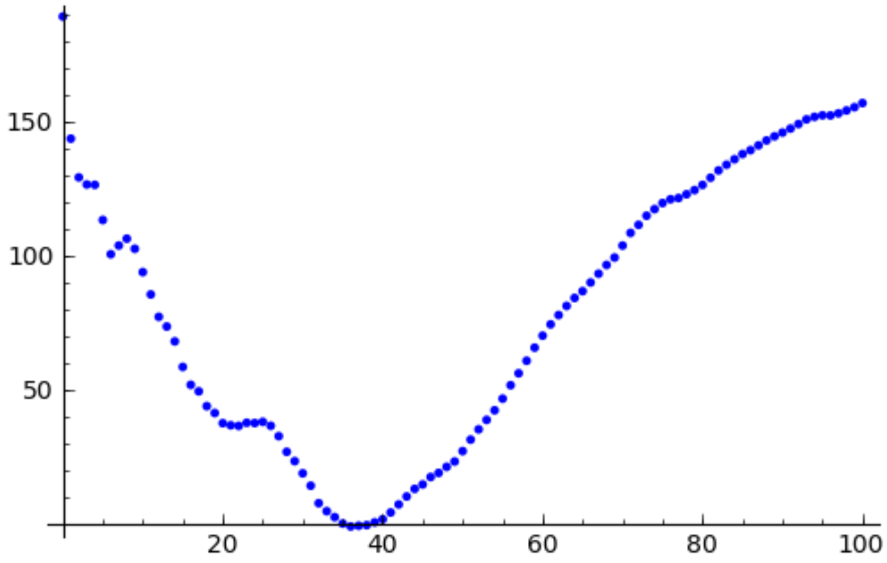
zeros. Maybe a better estimate could be made by considering the periods of the later zeros and their predicted contribution. Let's take a closer look at figure 23.
Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{13})$ as a function of the number of evaluations (separated by 10^6) of $Li_{10^4}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^6) required to calculate this approximation:
Figure 24:



In figure 24 the first wavelength is completed between $5.0 * 10^9$ and $5.5 * 10^9$ (between tick-marks 50 and 55). If we approximate the location as $5.3 * 10^9$, then the frequency in the logarithmic domain is about the same as that of zero number 12341.
Let's zoom into the first region of high accuracy in figure 24.
Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{13})$ as a function of the number of evaluations (separated by 10^5) of $Li_{10^4}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^5) required to calculate this approximation:
Figure 25:

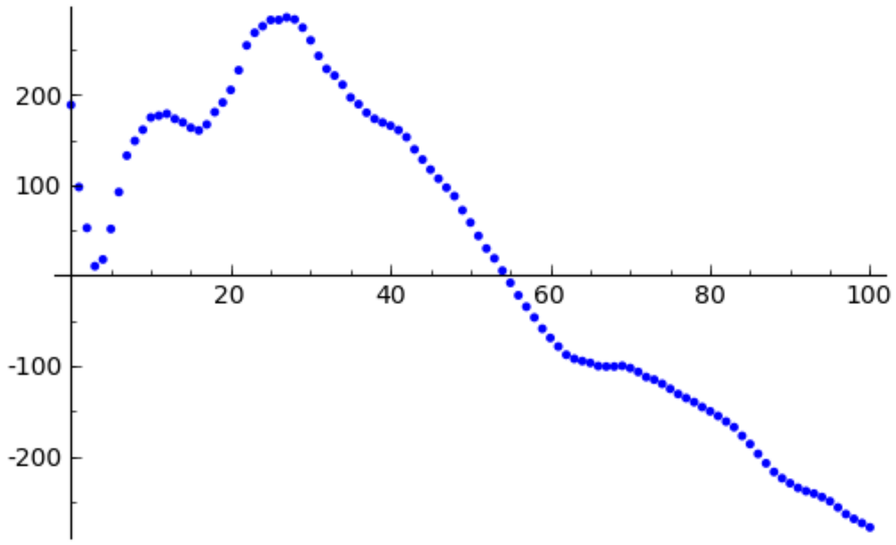


Let's look at the absolute error of the Monte-Carlo approximation instead of the number of bits (i.e. the log base 2 plus 1) needed to store the error. Below is a plot of the absolute error of the Monte-Carlo approximation of $\pi(10^{13})$ as a function of the number of evaluations (separated by 10^5) of $Li_{10^4}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^5) required to calculate this approximation:
Figure 26:

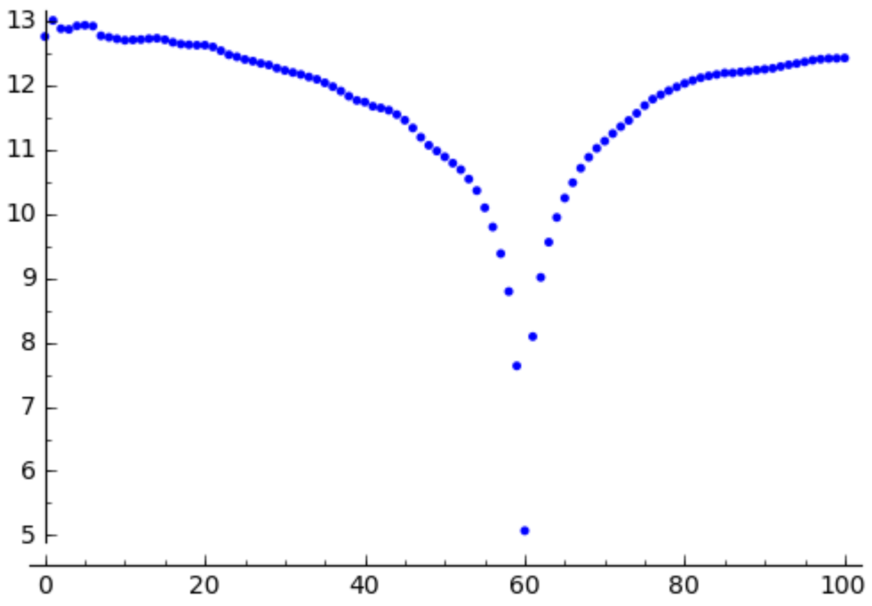


This behavior may be a fluke from the fact that $Li_{10^4}(x)$ has a below average

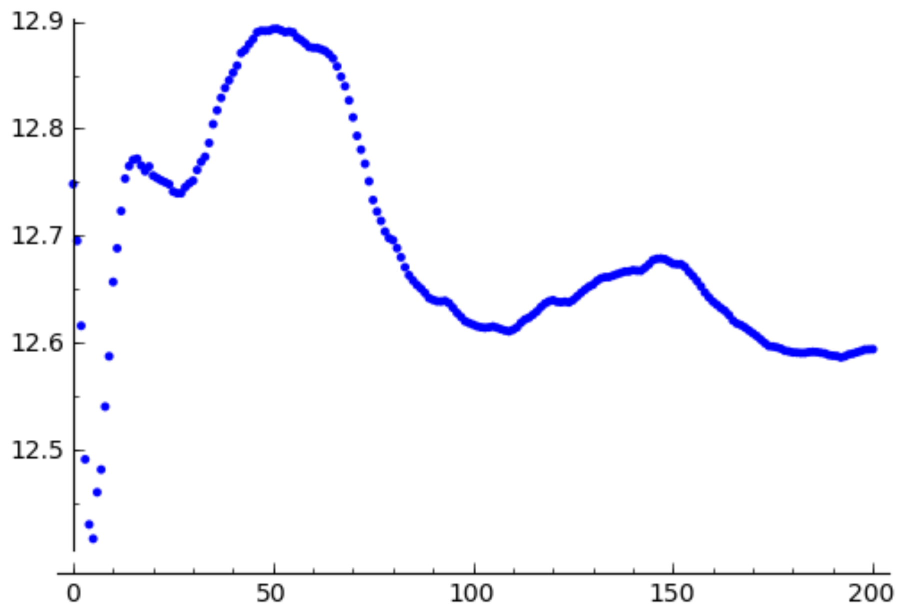
error at 10^{13}. The error of $Li_{10^4}(10^{13})$ is only 189 (8.56 bits) significantly less than errors of 2002, 1196, and 1620 for $Li_{10^4}(0.9 * 10^{13})$, $Li_{10^4}(1.1 * 10^{13})$, and $Li_{10^4}(1.2 * 10^{13})$, respectively. Perhaps random fluctuations of the value of $Li_{10^4}(x)$ cause the Monte–Carlo method's error to be zero here just by chance. The region of high accuracy after one wavelength can be seen to be a crossing of the x-axis rather than a "kissing" in the plot below. Perhaps this is another hint that the region of high accuracy in figure 25 is a fluke.
Below is a plot of the absolute error of the Monte-Carlo approximation of $\pi(10^{13})$ as a function of the number of evaluations (separated by 10^6) of $Li_{10^4}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^6) required to calculate this approximation:
Figure 27:



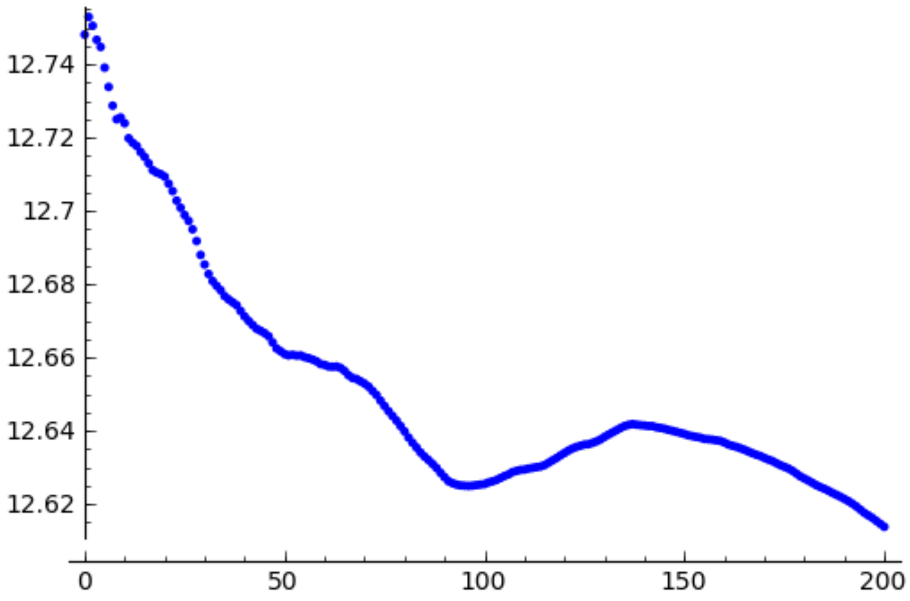
A region of high accuracy before the first period is not seen for the Monte–Carlo approximation of $\pi(10^{14})$.
Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{14})$ as a function of the number of evaluations (separated by 10^8) of $Li_{10^4}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^8) required to calculate this approximation:
Figure 28:



Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{14})$ as a function of the number of evaluations (separated by 10^7) of $Li_{10^4}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^7) required to calculate this approximation:
Figure 29:



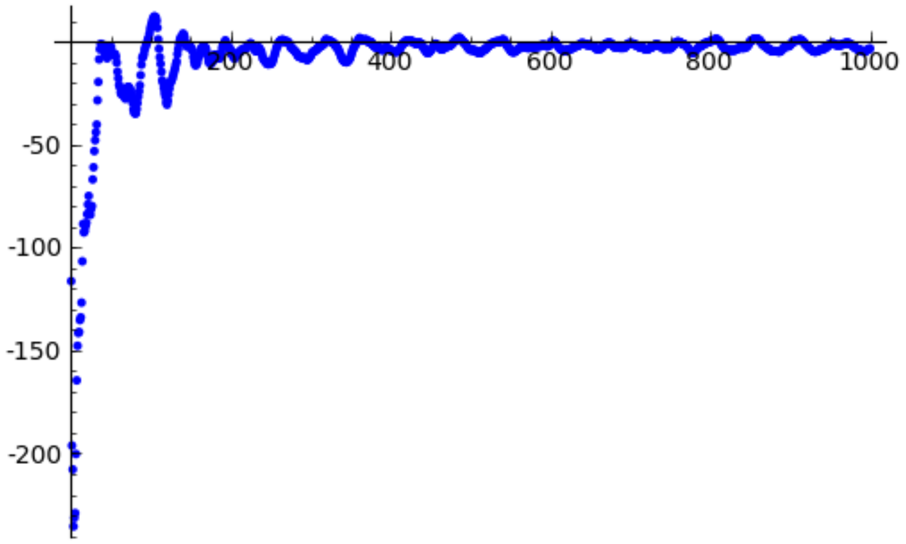
Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{14})$ as a function of the number of evaluations (separated by 10^5) of $Li_{10^4}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^5) required to calculate this approximation:
Figure 30:



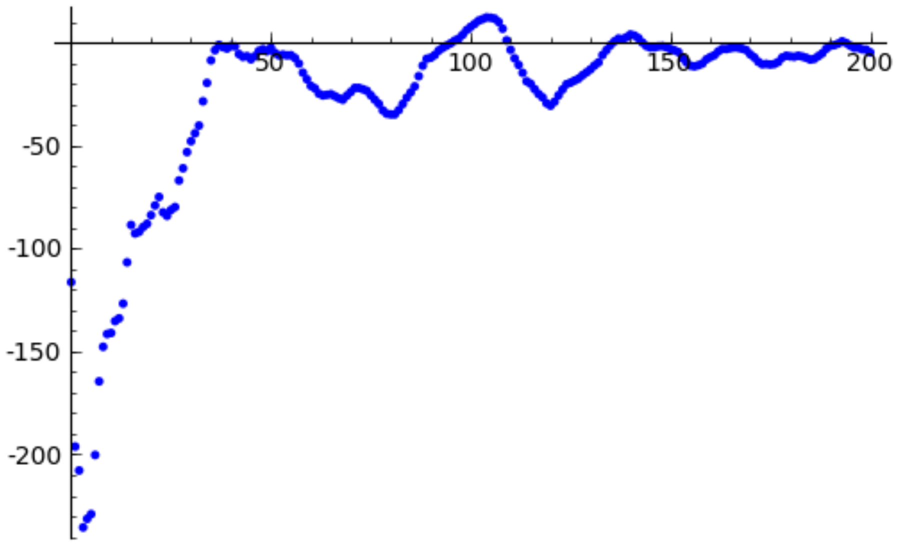
Let's look more at the Monte–Carlo approximation of $\pi(10^{11})$. Let's look at the absolute error for figure 14.

Below is a plot of the absolute error of the Monte-Carlo approximation of $\pi(10^{11})$ as a function of the number of evaluations (separated by 10^6) of $Li_{10^4}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^6) required to calculate this approximation:

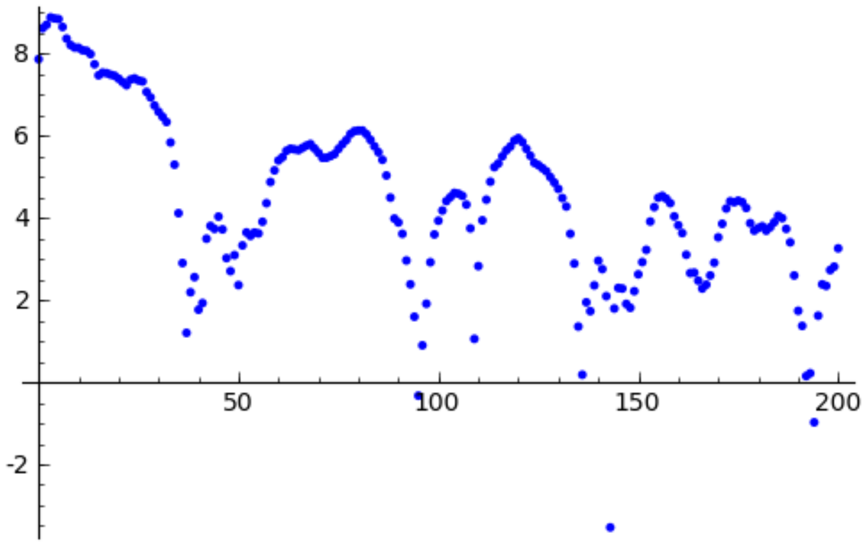
Figure 31:



Let's zoom in on figures 14 and 31.
Below is a plot of the absolute error of the Monte-Carlo approximation of $\pi(10^{11})$ as a function of the number of evaluations (separated by 10^6) of $Li_{10^4}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^6) required to calculate this approximation:
Figure 32:



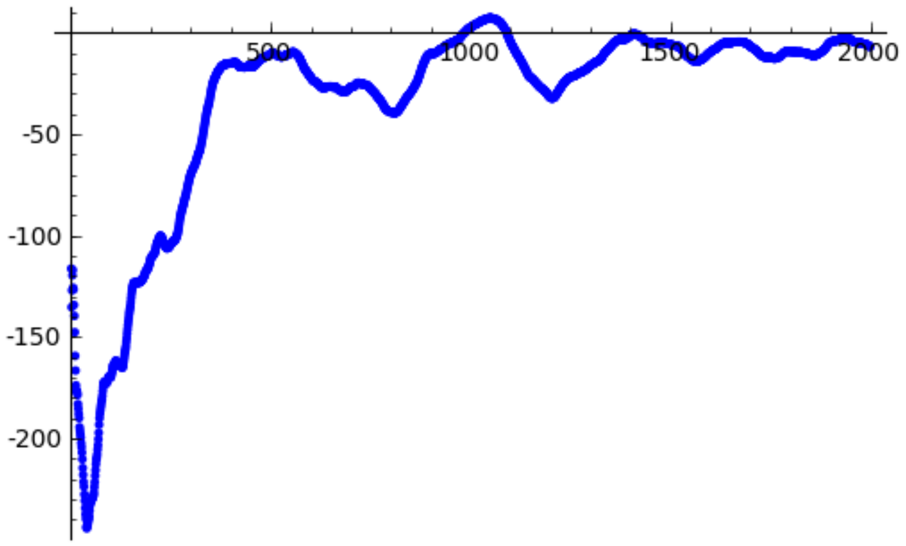
Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{11})$ as a function of the number of evaluations (separated by 10^6) of $Li_{10^4}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^6) required to calculate this approximation:
Figure 33:



Let's see how the last two plots change when the step size is changed from 10^6 to 10^5.

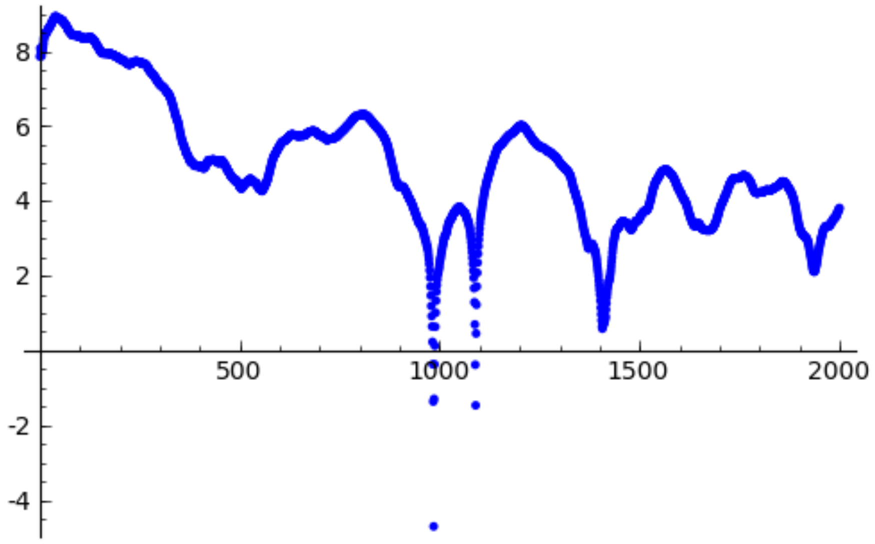
Below is a plot of the absolute error of the Monte-Carlo approximation of $\pi(10^{11})$ as a function of the number of evaluations (separated by 10^5) of $Li_{10^4}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^5) required to calculate this approximation:

Figure 34:



Below is a plot of the number of bits (i.e. the log base 2 plus 1) needed to store the error of the Monte-Carlo approximation of $\pi(10^{11})$ as a function

of the number of evaluations (separated by 10^5) of $Li_{10^4}(x_i)$–the x–axis also represents the amount of sieving (in units of 10^5) required to calculate this approximation:
Figure 35:

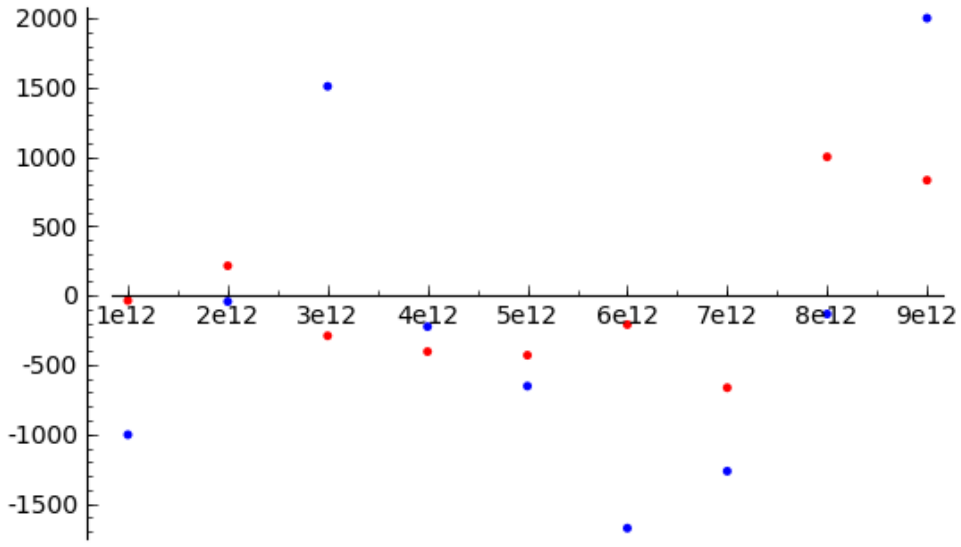


Notice that the error of the Monte-Carlo approximation of 10^{11} with a total sieving interval size of size $5 * 10^7$ (i.e. tick-mark 500 in figures 34/35 and tick-mark 50 in figures 32/33) is greater for a step size of 10^5 (figures 34/35) than for a step size of 10^6 (figures 32/33). More steps does not always mean less error.
Also notice that the last four figures show periodic behavior and are consistent with a wavelength somewhat less than that of the sinusoid for zero $10^4 + 1$, $6.358343021 * 10^{11}$ (i.e. tick-marks 60/600 in the last four figures). However this is less apparent than the corresponding periodic behavior in figure 23. The observed wavelength is also smaller for the last four figures than for figure 23.

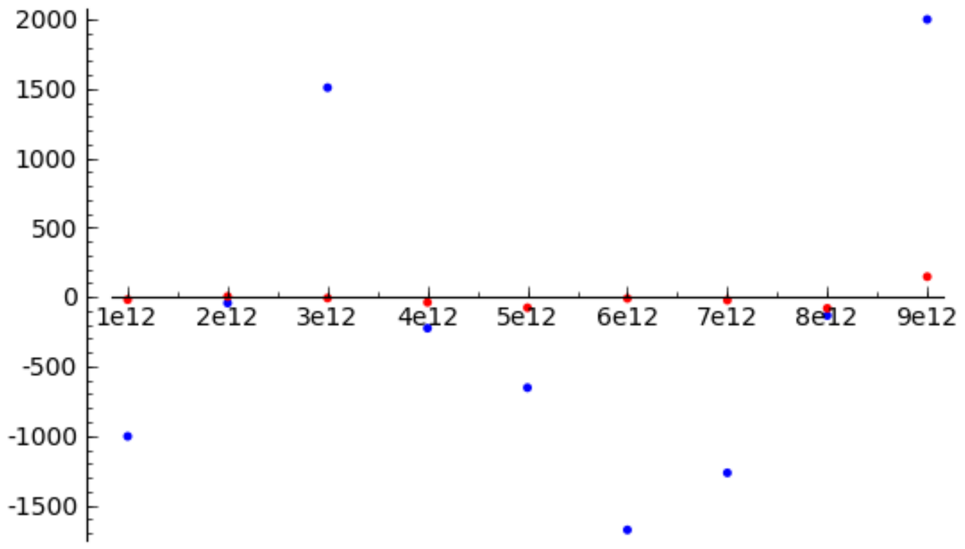
Comparing the Accuracy of the Monte–Carlo Method and the Explicit Formula

Let's compare the error of the Monte–Carlo method and the logarithmic integral based analytic approximation over a range of input values. First let's compare the results of changing the size of the sieving interval.
Below is a plot of the absolute error of the Monte-Carlo approximation with

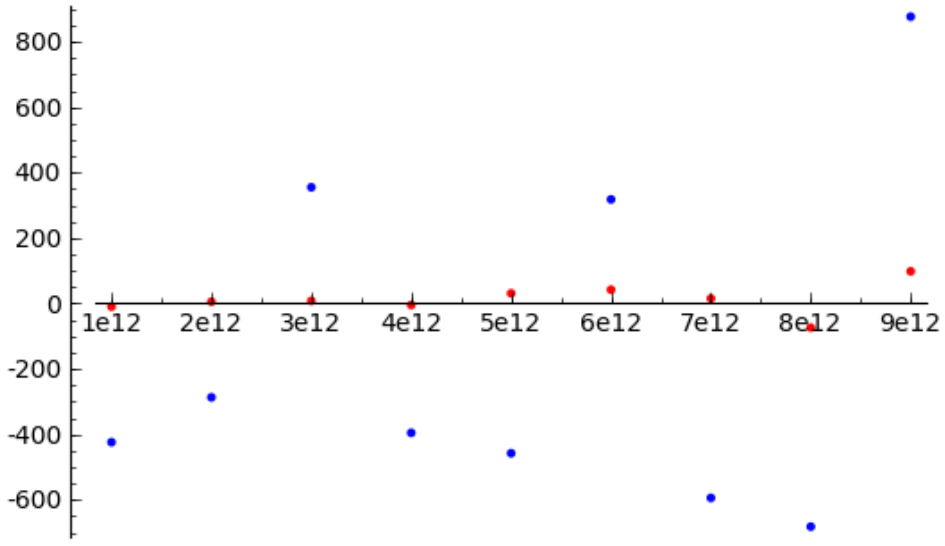
10^4 zeros and 100 sampling points separated by 10^7 (a total sieving interval of size 10^9) (red) and the error of the analytic approximation with 10^4 terms (blue).
Figure 36:



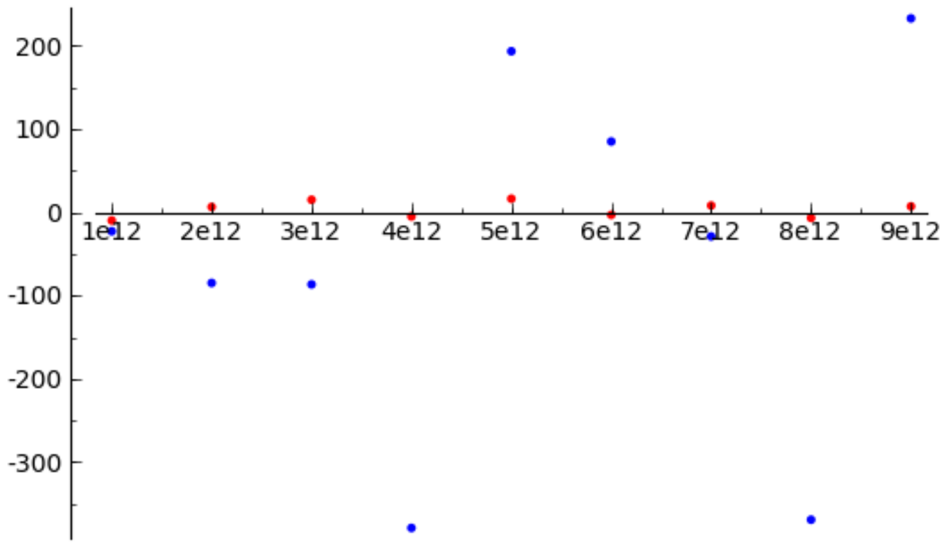
The error is significantly less, and closer to the factor of 10 improvement we would expect if we were sampling a random variable if we increase the size of the sieving interval to 10^{10}.
Below is a plot of the absolute error of the Monte-Carlo approximation with 10^4 zeros and 100 sampling points separated by 10^8 (a total sieving interval of size 10^{10}) (red) and the error of the analytic approximation with 10^4 terms (blue).
Figure 37:



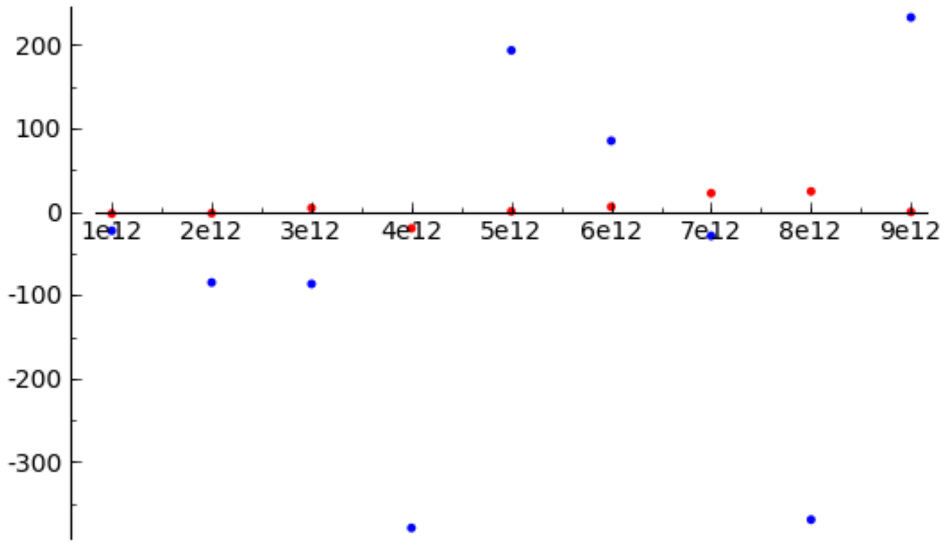
We obtain similar results by increasing the number of zeros.
Below is a plot of the absolute error of the Monte-Carlo approximation with 10^5 zeros and 100 sampling points separated by 10^7 (a total sieving interval of size 10^9) (red) and the error of the analytic approximation with 10^5 terms (blue).
Figure 38:



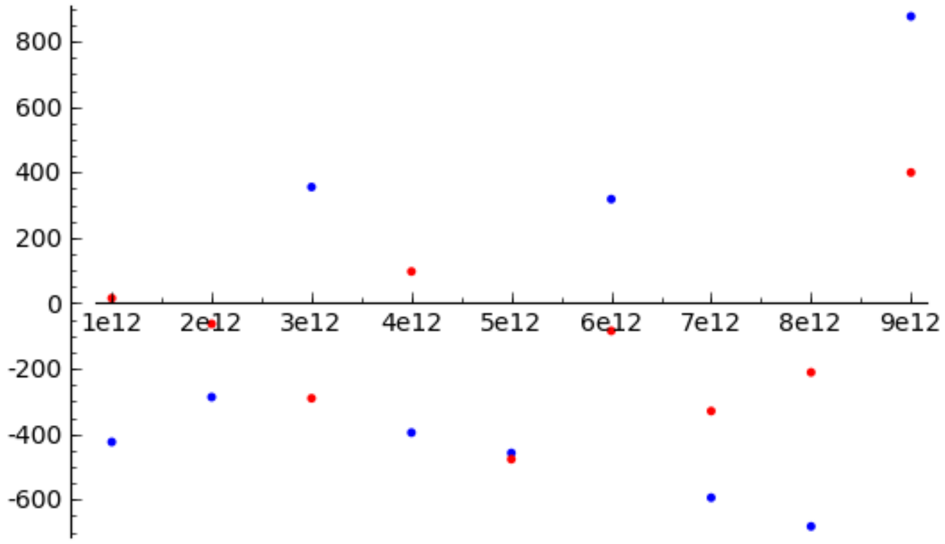
Below is a plot of the absolute error of the Monte-Carlo approximation with 10^6 zeros and 100 sampling points separated by 10^7 (a total sieving interval of size 10^9) (red) and the error of the analytic approximation with 10^6 terms

(blue).
Figure 39:


These results can be explained by considering the periods of the terms left out of the explicit formula in the Monte–Carlo method. The value of zero number $10^4 + 1$ is 9878.65477238, so the period (in the logarithmic domain) of its sinusoid is $\frac{2\pi}{9878.65477238} = 6.360365305 * 10^{-4}$. Taking the exponential of this number, we see that x must change by a factor of 1.000636239 for one complete period. This corresponds to an absolute change in x of approximately $6.4 * 10^8$ per period for an x near 10^{12}. For 10^5, 10^6, and 10^9 zeros this value becomes a change of $8.4 * 10^7$, $1.0 * 10^7$, and $1.7 * 10^4$ respectively. After a change in x on the order of this size, we might expect the arguments of the sinusoids for zeros after the n–th to become sufficiently shuffled that the error of the analytic approximation can be considered a random variable where each value is not correlated to the previous. The accuracy is not optimal in figure 36 because the sampling points are not sufficiently separated for the error of the analytic approximation to be considered a random variable. We can use a smaller sieving interval if we use more zeros.
Below is a plot of the absolute error of the Monte-Carlo approximation with 10^6 zeros and 100 sampling points separated by 10^6 (a total sieving interval of size 10^8) (red) and the error of the analytic approximation with 10^6 terms (blue).
Figure 40:



But we still cannot use too small a sieving interval for the number of zeros, or the error of the analytic approximation at successive sampling points will be too strongly correlated, and not act like a random variable.
Below is a plot of the absolute error of the Monte-Carlo approximation with 10^5 zeros and 100 sampling points separated by 10^6 (a total sieving interval of size 10^8) (red) and the error of the analytic approximation with 10^5 terms (blue).
Figure 41:



We now have another possible explanation for the periodic nature of the error of the Monte-Carlo method with respect to the size of the sieving interval seen in the previous section–after a certain interval the arguments of the sinusoids have been shuffled enough that the error of the analytic approximation is not correlated with the previous error and can be considered a random variable.

Optimizing the Monte–Carlo Method

We derived:

$$\pi(x) \approx Li_n(x) + Avg_i[\pi(x) - \pi(x_i) - Li_n(x) + Li_n(x_i))]$$

where

$$Li_n(x) = li(x) - log(2) - \sum_{k=2}^{\infty} \pi(x^{1/k}) + \int_x^{\infty} \frac{dt}{t(t^2-1)log(t)} - \sum_{\rho}^{nzeros} li(x^{\rho})$$

In his source code, Tomás Oliveira e Silva says of the integral term, "this term is always smaller than $1/(2*x^2*log(x))$, and so can usually be ignored". We will take his advice and remove this term from our approximation.
We now have:

$$\pi(x) \approx li(x) - log(2) - \sum_{k=2}^{\infty} \frac{\pi(x^{1/k})}{k} - \sum_{\rho}^{nzeros} li(x^{\rho})$$

$$+Avg_i[\pi(x) - \pi(x_i) - (li(x) - log(2) - \sum_{k=2}^{\infty} \frac{\pi(x^{1/k})}{k} - \sum_{\rho}^{nzeros} li(x^{\rho}))$$

$$+(li(x_i) - log(2) - \sum_{k=2}^{\infty} \frac{\pi(x_i^{1/k})}{k} - \sum_{\rho}^{nzeros} li(x_i^{\rho}))]$$

$$= -log(2) - \sum_{k=2}^{\infty} \frac{\pi(x^{1/k})}{k} + Avg_i[\pi(x) - \pi(x_i) + \sum_{k=2}^{\infty} \frac{\pi(x^{1/k})}{k}$$

$$+(li(x_i) - \sum_{k=2}^{\infty} \frac{\pi(x_i^{1/k})}{k} - \sum_{\rho}^{nzeros} li(x_i^{\rho}))]$$

$$= -log(2) - \sum_{k=2}^{\infty} \frac{\pi(x^{1/k})}{k} + Avg_i[\sum_{k=1}^{\infty} \frac{\pi(x^{1/k}) - \pi(x_i^{1/k})}{k} + li(x_i) - \sum_{\rho}^{nzeros} li(x_i^{\rho})]$$

$$= Avg_i[li(x_i)] - log(2) - \sum_{k=2}^{\infty} \frac{\pi(x^{1/k})}{k} + Avg_i[\sum_{k=1}^{\infty} \frac{\pi(x^{1/k}) - \pi(x_i^{1/k})}{k}] - Avg_i[\sum_{\rho}^{nzeros} li(x_i^{\rho})]$$

$Avg_i[li(x_i)]$ can be approximated very well as $\frac{1}{b-a}\int_a^b li(x)dx$, where [a, b] is the total sieving interval. This integral can be calculated by integrating the exponential integral based series for the logarithmic integral.
$\sum_{k=2}^{\infty}\frac{\pi(x^{1/k})}{k}$ can be calculated in essentially $O(x^{\frac{1}{2}})$ time with sieving, $O(x^{\frac{1}{3}})$ time with the combinatorial method, or $O(x^{\frac{1}{4}})$ time with the analytic or table-based method.
$Avg_i[\sum_{k=1}^{\infty}\frac{\pi(x^{1/k})-\pi(x_i^{1/k})}{k}]$ can be calculated in time essentially equal to $O(b-a+\sqrt{b})$ with sieving or time essentially equal to $O(b-a)$ with a primality test. Most likely, we will have $b-a>\sqrt{b}$, producing essentially $O(b-a)$ time with sieving. For $k>1$, it is likely that the sieving interval will be small enough for a primality test to be the best option for counting primes.
We are left with computing the $Avg_i[\sum_{\rho}^{nzeros} li(x_i^{\rho})]$ term. Define m to be the number of Monte–Carlo sample points. We now have:

$$Avg_i[\sum_{\rho}^{nzeros} li(x_i^{\rho})] = \frac{1}{m}\sum_{i}^{msamples}[\sum_{\rho}^{nzeros} li(x_i^{\rho})] = \frac{1}{m}\sum_{\rho}^{\text{n zeros}}[\sum_{i}^{msamples} li(x_i^{\rho})]$$

$$\approx \frac{1}{m}\sum_{\rho}^{\text{n zeros}}[\sum_{i}^{msamples} 2\frac{\sqrt{x_i}}{\log x_i}\frac{0.5\cos(t\log x_i)+t\sin(t\log x_i)}{0.25+t^2}\sum_{k=0}^{\infty}\frac{k!}{(\log x_i^{0.5+It})^k}$$

The previous step approximates the logarithmic integral with the asymptotic series. The next step is to change

$$\sum_{\rho}^{\text{n zeros}}\sum_{i}^{\text{m samples}}$$

to

$$\sum_{\rho}^{\text{(n zeros)}}\sum_{j}^{\text{(r groups of i's)}}\sum_{i}^{\text{(m/r samples)}}$$

$$=\sum_{j}^{\text{(r groups of i's)}}\sum_{\rho}^{\text{(n zeros)}}\sum_{i}^{\text{(m/r samples)}}$$

where in each group j of i's we make the approximation of considering

$$2\frac{\sqrt{x_i}}{\log x_i}[\sum_{k=0}^{\infty}\frac{k!}{(\log x_i^{0.5+It})^k}]$$

to be constant. Now we have:

$$\sum_{i}^{\text{(m/r samples)}} 2\frac{\sqrt{x_j}}{\log x_j}\frac{0.5\cos(t\log x_i)+t\sin(t\log x_i)}{0.25+t^2}\sum_{k=0}^{\infty}\frac{k!}{(\log x_j^{0.5+It})^k}$$

$$= 2\frac{\sqrt{x_j}}{\log x_j}[\sum_{k=0}^{\infty}\frac{k!}{(\log x_j^{0.5+It})^k}]\sum_{i}^{\text{(m/r samples)}}\frac{0.5\cos(t\log x_i)+t\sin(t\log x_i)}{0.25+t^2}$$

We choose our x_i's so that they are in a geometric progression and apply the formulas:

$$\sin\varphi+\sin(\varphi+\alpha)+\sin(\varphi+2\alpha)+\cdots+\sin(\varphi+s\alpha)=\frac{\sin\left(\frac{(s+1)\alpha}{2}\right)\cdot\sin(\varphi+\frac{s\alpha}{2})}{\sin\frac{\alpha}{2}}$$

and

$$\cos\varphi+\cos(\varphi+\alpha)+\cos(\varphi+2\alpha)+\cdots+\cos(\varphi+s\alpha)=\frac{\sin\left(\frac{(s+1)\alpha}{2}\right)\cdot\cos(\varphi+\frac{s\alpha}{2})}{\sin\frac{\alpha}{2}}$$

where $s = m/r$. The number of calls to sine and cosine will be no more than $6 * n * r$.

The Complexity of the Monte–Carlo Method

For one period of one of the sinusoids to occur, x must change by a factor of $e^{\frac{2\pi}{t}}$. Because $\frac{2\pi}{t}$ is small, considering the Taylor series for the exponential function provides $e^{\frac{2\pi}{t}} \approx 1+\frac{2\pi}{t}$. This means that the size of the interval needed for the arguments of the sines in the error of the analytic approximation of x (i.e. those sinusoids not included in the analytic approximation) to become shuffled is about $\frac{2\pi x}{t}$, where t is the height of the first zero not included in the analytic approximation. The Riemann-von Mangoldt formula says that n is essentially $O(t)$, so the size of one of the sieving steps, (i.e. the separation between Monte–Carlo sampling points, at one point anyway) is essentially $O(\frac{x}{n})$. We know t is $o(n)$, so the separation between Monte–Carlo sampling points is $\Omega(\frac{x}{n})$. Let's choose an m small enough that $2\frac{\sqrt{x_i}}{\log x_i}[\sum_{k=0}^{\infty}\frac{k!}{(\log x_i^{0.5+It})^k}]$ changes by less than about 1 and r can be one. We are considering essentially $2\sqrt{x_i}$, which has a derivative of $\frac{1}{\sqrt{x_i}}$. The function $2\sqrt{x}$ is concave down, and will remain constant for an interval of size about $\sqrt{x_i}$. Hence we choose m and n such that $\frac{x*m}{n}$ is essentially $O(\sqrt{x})$. Note that we are using the approximation $(e^{\frac{2\pi}{t}})^m \approx 1+\frac{2\pi*m}{t}$ which is valid for $t >> m$.

Let's assume that increasing the number of zeros by a factor of 10 decreases the error by a factor of about 2.3 and that increasing the number of sample points by a factor of 10 decreases the error by a factor of about 3.16. Let's let the number of zeros grow like essentially $O(x^{0.875})$. Then the separation between Monte–Carlo sampling points is essentially $O(x^{0.125})$. We can use essentially $O(x^{0.375})$ sampling points without exceeding the $\sqrt{x}$ barrier.

$0.875 * log_{10}(2.3) + .375 * log_{10}(3.16) > 0.5$, so we have decreased the error from essentially $\sqrt{x}$ to about 0.5. We used essentially $O(x^{\frac{7}{8}})$ terms from the nontrivial zeros and $O(x^{1/2})$ time sieving to calculate an approximation to $\pi(x)$ accurate to within about 0.5.
Under the more conservative assumption that increasing either the number of zeros or the number of sample points by a factor of 10 decreases the error by a factor of about 2.3, we can let the number of zeros grow like essentially $O(x^{0.942})$ and the number of sampling points grow like essentially $O(x^{0.442})$ for similar results.

Conclusions

The Monte–Carlo method highlights the relationship between order and randomness in the primes. We have given a heuristic argument that the Monte–Carlo method reduces the time complexity of the explicit formula from essentially $O(x^{1.38})$ to essentially $O(x^{\frac{7}{8}})$, which is $o(x)$. Although much faster methods of computing $\pi(x)$ exist, the Monte–Carlo method is interesting in that it uses direct evaluations of the explicit formula and heuristically uses $o(x)$ time. The Monte–Carlo method is also interesting in its unique way of understanding the prime counting function. The Achilles' heel of the Monte–Carlo method is the validity of the assumptions on which it is based: that the mean error approaches 0, and that the error can be considered to be a random variable. If the interval sieved is too large and x varies too greatly, the mean error may not approach 0 (see figures 15 and 16, e.g.). For an interval of size essentially $\sqrt{x}$, it seems reasonable that the variation of x will have negligible effect on the mean error. If the sampling spacing is not large enough, the arguments of the sinusoids will not be shuffled enough and the errors will be too strongly correlated. We hope that this paper will inspire others to explore new ways of understanding the prime counting function and that it will lead to more use of tables of the nontrivial zeros. Hopefully someone will continue this work by doing a more thorough analysis of the Monte–Carlo method.

Appendix A–Calculating the Periodic Terms

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
//This is some rough code to calculate li(x^(0.5+it))+li(x^(0.5-it))
//Authors:
//Tomas Oliveira e Silva: Original Version January 2010
```

```
//Kevin Stueve: Rewrote and optimized for speed in C March 2010
//Leif Leonhardy: Moved zeros to separate file March 2010
const double zeros[]={
#include "zeros6.c" // the first 1000000

};
double li(double x, double t);
int main(int argc,char **argv){
  //printf("zero%f",zeros[50000]);
  double x=atof(argv[1]);
  long int num = atol(argv[2]);
  //printf("%f\n",x);
  //printf("%ld\n",num);
  double result=0;
  long int i=0;
  for (i=0;i<num;i++){
    //printf("%f\n",zeros[i]);
    result+=li(x,zeros[i]);
  }
  printf("%f",result);
  return 0;
}
double li(double x, double t) {
    double logx = log(x);

    double u_real = 0.5 * logx;
    double u_imag = t * logx;

    double u_norm = u_real * u_real + u_imag * u_imag;

    double so_real_num = 2 * sqrt(x) * cos(t * logx);
    double so_imag_num = 2 * sqrt(x) * sin(t * logx);

    double so_real = (so_real_num * u_real + so_imag_num * u_imag) / u_norm;
    double so_imag = (so_imag_num * u_real - so_real_num * u_imag) / u_norm;

    double tol=pow(10,-9)/(sqrt(so_real * so_real + so_imag * so_imag));
    double tol_squared=tol*tol;

    double s1_real = 1;
    double s1_imag = 0;
    double s2_real = 1;
    double s2_imag = 0;

    double s2_real_temp;
    double s2_imag_temp;

    int k = 1;
    double norm,lastnorm;
    for(;;) {

        if (k>100) {
          printf("error%f\n",t);
          return;
          //break;
        }

        s2_real_temp = k * (s2_real * u_real + s2_imag * u_imag) / u_norm;
        s2_imag_temp = k * (s2_imag * u_real - s2_real * u_imag) / u_norm;

        s2_real = s2_real_temp;
        s2_imag = s2_imag_temp;

        s1_real += s2_real;
        s1_imag += s2_imag;
        norm=s2_real * s2_real + s2_imag * s2_imag;
        if (norm < tol_squared) {
            break;
        }

        k++;
    }
    return so_real * s1_real - so_imag * s1_imag;
}
```

Apendix B–Fredrik Johansson's Comparison of the Exact and Asymptotic Li Series

```
from math import exp, log
```

```
gamma = 0.57721566490153287
eps = 1e-17
asymp_cutoff = 50
li2 = 1.04516378011749278

def ei(x, _xexp=None):
   if abs(x) < asymp_cutoff:
       s = gamma + log(abs(x))
       t = 1.0
       k = 1
       while abs(t) > eps:
           t *= x
           t /= k
           s += t / k
           k += 1
   else:
       s = t = 1.0
       k = 1
       r = 1. / x
       while abs(t) > eps:
           t *= k
           t *= r
           s += t
           k += 1
       if _xexp is None:
           _xexp = exp(x)
       s = s * r * _xexp
   return s

def li(x):
   return ei(log(x), x)

# "Offset" logarithmic integral
def Li2(x):
   return ei(log(x), x) - li2

def li_approx(y):
   x = log(y)
   s = t = y
   r = 1. / x
   k = 1
   tprev = t
   while t <= tprev:
       tprev = t
       t *= k
       t *= r
       s += t
       k += 1
   return s*r
li2 = 1.04516378011749278
def Li_approx(y):
   return li_approx(y)-li2
for n in range(1,12):
    y = 10**n
    print "%14.3f %14.3f" % (li_approx(y)-li(2), li(y)-li(2))
```

Appendix C–Linear Regression Code

```
 #######################
#Linear Regression Code
#######################
import numpy

# Polynomial Regression
#From
http://stackoverflow.com/questions/893657/
how-do-i-calculate-r-squared-using-python-and-numpy
def polyfit(data, degree):
    results = {}
    x = list(point[0] for point in data)
    y = list(point[1] for point in data)
    coeffs = numpy.polyfit(x, y, degree)

     # Polynomial Coefficients
    results['polynomial'] = coeffs.tolist()

    # r-squared
    p = numpy.poly1d(coeffs)
    # fit values, and mean
    yhat = [p(z) for z in x]
    ybar = sum(y)/len(y)
    ssreg = sum([ (yihat - ybar)**2 for yihat in yhat])
    sstot = sum([ (yi - ybar)**2 for yi in y])
    results['determination'] = ssreg / sstot

    return results
```

Bibliography

Terence Tao, UCLA Deans Seminar, University of Sydney, 8 February 2008

http://www.ieeta.pt/~tos/primes.html#e

The Prime Counting Function and Related Subjects
Patrick Demichel (2005)
http://web.archive.org/web/20060908033007/http://demichel.net/patrick/li_crossover_pi.pdf

Meta Math! The Quest for Omega
Gregory Chaitin 2005
Vintage

http://wstein.org/rh/rh/code/code.sage

http://www.sagemath.org/doc/reference/sage
/databases/odlyzko.html

http://en.wikipedia.org/wiki/List_of_trigonometric_identities
#Other_sums_of_trigonometric_functions

http://mathworld.wolfram.com/RiemannPrimeCountingFunction.html

http://en.wikipedia.org/wiki/Explicit_formula#Riemann.27s_explicit_formula

http://primes.utm.edu/notes/faq/one.html

Don Zagier The First 50 Million Prime Numbers 1975

http://en.wikipedia.org/wiki/Logarithmic_integral_function

http://trac.sagemath.org/sage_trac/ticket/8135 http://mathworld.wolfram.com/Riemann-vonMangoldtFormula.html

http://listserv.nodak.edu/cgi-bin/wa.exe
?A2=ind0811&L=NMBRTHRY&P=R401&I=-3

The Nth Prime Page, A prime page by Andrew Booker
Copyright 1999-2009 Chris Caldwell
http://primes.utm.edu/nthprime/index.php

The Fluctuations of the Prime-Counting Function
Compiled by Andrey V. Kulsha
Data from: Xavier Gourdon, Thomas R. Nicely,
Anotoly F. Selvich, Tomas Oliveira e Silva 2010-03-03
http://www.primefan.ru/stuff/primes/table.html

Computing pi(x): An Analytic Method, Lagarias, Odlyzko
1987, http://www.dtc.umn.edu/~odlyzko/doc/arch/analytic.pi.of.x.pdf

Computing pi(x): The Meissel-Lehmer Method
Jeffrey Lagarias, Victor Miller, Andrew Odlyzko 1985
http://www.dtc.umn.edu/~odlyzko/doc/arch/meissel.lehmer.pdf

Computing pi(x): The Meissel, Lehmer, Lagarias, Miller, Odlyzko Method
Deleglise, Rivat 1996
http://cr.yp.to/bib/1996/deleglise.pdf

Computation of pi(x) : Improvements to the Meissel, Lehmer, Lagarias,
Miller, Odlyzko, Deleglise and Rivat method
Xavier Gourdon 2001
http://numbers.computation.free.fr/Constants/Primes/Pix/piNalgorithm.ps

Computing pi(x): The combinatorial method
Tomas Oliveira e Silva 2006
http://www.ieeta.pt/~tos/bib/5.4.pdf

What is Riemann's Hypothesis? (Draft)
Barry Mazur and William Stein 2009
http://wstein.org/rh/

Mathematical Mysteries, the Beauty and Magic of Numbers
Calvin C. Clawson 1996

Perseus Books

http://en.wikipedia.org/wiki/Standard_error_%28statistics%29

http://nt.sagenb.org/home/pub/2/

http://en.wikipedia.org/wiki/M%C3%B6bius_mu_function

http://en.wikipedia.org/wiki/Riemann_hypothesis

http://en.wikipedia.org/wiki/Monte_Carlo_method