Paper 3384-2015

Exploring Number Theory Using Base SAS®

Matthew Duchnowski, Educational Testing Service

ABSTRACT

Long before analysts began mining large datasets, mathematicians sought truths hidden within the set of natural numbers. This exploration was formalized into the mathematical subfield known as Number Theory. Though this discipline has proven useful for many applied fields, Number Theory delights in numerical truth for its own sake. The austere and abstract beauty of Number Theory prompted nineteenth century mathematician Carl Friedrich Gauss to dub it the "Queen of Mathematics." Here, the author will show that the analytical power of the SAS® engine is well-suited to the exploration of topics in Number Theory. While the concepts discussed here comprise only a small survey of topics, readers can imitate the general approach and craft their own investigations.

INTRODUCTION

Computers don't do mathematics. Computers compute.

This is a fact worth noting from the start, because although we seek here to *explore* and *illustrate* mathematical concepts, we should recognize our limitations: real mathematics involves reasoning and argumentation; it is a human endeavor.

Even so, great insights may be gained from "playing with numbers." It can be illuminating to verify existing numerical facts – and to discover new ones – that apply to an infinitude of integers, even when using a finite set of data to do so.

And with that prelude, we will begin by defining the set of positive integers up to a chosen limit. Here we will use one million.

```
%let N = 1000000;

data posints;
   do n= 1 to &N.;
      output;
   end;
run;
```

We will build on this data set throughout.

PRIME FACTORIZATION

The Fundamental Theorem of Arithmetic forms the bedrock of Number Theory and so provides a great place to start. Simply stated, the theorem tells us that every integer greater than 1 is represented uniquely as the product of primes. In other words, the following representation is unique for each sufficiently large integer n :

$$n = p_1^{\alpha_1} p_2^{\alpha_2} p_3^{\alpha_3} \dots p_r^{\alpha_r}$$

In the above, each prime p_s is unique and associated with a power $\alpha_s \geq 1$ where $1 \leq s \leq r$.

Factoring large numbers is a notoriously difficult problem. The code provided here strikes a balance between computational expense and needless complexity. It begins by factoring out all powers of 2 and then proceeds to do the same with odd numbers starting at 3. Of course, the only odd numbers that will succeed in dividing into the remainder are primes. Because factors exist in pairs, one member being less than or equal to $\sqrt{n}$ and the other being greater than or equal to $\sqrt{n}$, we can lower the search boundary. After this process concludes, any remaining number clearly has no factor (i.e. is prime) and so represents its own factorization.

The code below executes the algorithm and stores the factorization for later use:

```
data prmfctrs (drop= i quotient index);

  set posints;

  array prime{10} 8. prime1 - prime10;
  array power{10} 3. power1 - power10;

  index = 0 ;
  quotient = n/1;

  if quotient = 2 then do;
        prime[1] = 2;
        power[1] = 1;
  end;

  else if quotient  > 1 then do;

        /*Remove powers of 2*/
        if (MOD(quotient,2)=0) then do;
              index = index + 1;
              prime[index] = 2;
        end;
        do while (MOD(quotient,2)=0);
              power[index] = sum(power[index],1);
              quotient = quotient / 2;
        end;


        /*Remove powers of all odd numbers*/
        do i = 3 to FLOOR(SQRT(quotient)) by 2;
              if mod(quotient,i)=0 then do;
                    index = index + 1;
                    prime[index] = i;
              end;
              do while (mod(quotient,i)=0);
                    power[index] = sum(power[index],1);
                    quotient = quotient / i;
              end;
        end;

        /*Only odd primes remain*/
        if quotient ge 3 then do;
              index = index + 1;
              prime[index] = quotient;
              power[index] = 1;
        end;

  end;
run;
```

The new dataset contains the prime factorization of n, stored in two arrays, `prime[]` and `power[]`, which capture p_s and α_s, respectively. Each array is assigned here with length of 10. Of course, ten distinct primes are insufficient for representing all positive integers, but it should be noted that the product of the ten smallest prime numbers is astronomical in its own right, and so gives a reasonably large sample for our investigations.

$$2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29 > 6.46 \times 10^9$$

Now that we have these factorizations stored, it might be of interest to report them in an easily readable form. To achieve this, we can write the entire expansion to a string using the following code.

```
data primefact (drop = i factor:) ;

   set prmfctrs;

   length expansion $200;

   array prime{10} 8. prime1 - prime10;
   array power{10} 3. power1 - power10;
   array factor{10} $20. factor1 - factor10;

   do i= 1 to 10;
      if prime[i] eq . then leave;
      else do;

      factor[i] =
         CATX(' ',%str(prime[i]),'~{super',' ',%str(power[i]),'}');

      end;
   end;

   expansion = CATX(' ', OF factor1-factor10);

run;
```

We now have a character variable `expansion`, which stores the factorization using a printable syntax. Using SAS output delivery system (ODS), we can assign an escape character and print a factorization table. The example below shows a factor table for integers having a specific property.

```
ods pdf file='FactorTable.pdf';
ods escapechar='^';

proc print data=primefact (obs = 5) noobs;
   title1 'Factorization Table';
   var n expansion;
   where 5 = mod(n,7);
run;
```

This produces a readable factorization table for the selected observations.

Factorization Table

n	expansion
5	5^{1}
12	$2^{2}\ 3^{1}$
19	19^{1}
26	$2^{1}\ 13^{1}$
33	$3^{1}\ 11^{1}$

PHI, SIGMA AND TAU

A number of arithmetic functions can be determined using an integer's prime factorization. The most famous of these are the multiplicative functions $\varphi(n)$, $\sigma(n)$ and $\tau(n)$. These three functions alone could provide a Number Theorist with enough material to last an entire career. The three functions can be described as follows:

> $\varphi(n)$: the number of integers less than and relatively prime to n.
>
> $\sigma(n)$: the sum of divisors of n (including 1 and n).
>
> $\tau(n)$: the number of divisors of n (including 1 and n).

Lucky for us, each of these functions can be evaluated using the prime factorization of n, which we have already captured. It is beyond the scope of this article to derive each formula but it is interesting to note that only one of the three formulas below requires information from both the `prime[]` and the `power[]` array populated above.

$$\varphi(n) = n\prod_{i=1}^{r}\left(1-\frac{1}{p_i}\right) \qquad \sigma(n) = \prod_{i=1}^{r}\frac{p_i^{(\alpha_i+1)}-1}{p_i-1} \qquad \tau(n) = \prod_{i=1}^{r}(\alpha_i+1)$$

where $n = p_1^{\alpha_1} p_2^{\alpha_2} p_3^{\alpha_3} \ldots p_r^{\alpha_r}$.

For more information, the reader is encouraged to consult Ore (1998) and/or Andrews (1994).

These functions may then be evaluated for each *n* by iterating over the prime factorization as shown below.

```
data numbers (drop=i);

   set primefact;

   array prime{*} prime:;
   array power{*} power:;

   phi = n;
   sigma = 1;
   tau = 1;

   do i =1 to dim(prime);
      if prime[i] eq . then leave;
      else do;

         /*Phi, Sigma and Tau functions*/
         phi   =   phi * (1-(1/prime[i]));
         sigma = sigma * ((prime[i]**(power[i]+1)-1)/(prime[i]-1));
         tau   =   tau * (power[i]+1);

      end;
   end;

   /*Additional properties*/

run;
```

Having these values stored allows the user to access them for a multitude of applications. For example, we can ask:

Are there any solutions to $\varphi\left(\sigma\left(n\right)\right) = n$?

The data set can be merged with itself to evaluate the compound function.

```
proc sql outobs=10;
   select x.n , x.expansion
   from
      numbers as x, numbers as y
   where
      x.n = y.phi and y.n = x.sigma
   ;
quit;
```

By executing this query, we see that there are a handful of solutions where n < 1,000,000. Not having known this in advance, have confined the number of observations using the `outobs` keyword. In doing so, we eliminate the danger of accidentally printing thousands of pages to the output window.

NUMERIC PROPERTIES

At this point, our dataset NUMBERS contains one million integers, each factored and evaluated for three arithmetic functions. We can assign new variables to store additional arithmetic properties; some of which can be obtained directly from the integer, others that can be deduced using the prime factorization, and some that require other previously-generated properties.

All of the code segments below – and others that the reader may wish to fashion – can be inserted into the data step above, in place of `/*Additional properties*/`. Examples of these include:

Basic properties, derived directly from integer:

```
isOdd = MOD(n,2);
isEven = MOD(n+1,2);
```

Properties derived from the prime factorization:

```
isPowerOf4 = 1;
do i =1 to dim(prime);
   if power[i] eq . then leave;
   else do;
      if MOD(power[i],4) > 0 then isPowerOf4 = 0;
   end;
end;
```

Properties derived from previously generated properties. For detailed information on *perfect*, *abundant* and *deficient* numbers, the reader is encouraged to consult Andrews (1994):

```
if tau = 2         then isPrime = 1;       else isPrime = 0;
if sigma = 2*n     then isPerfect = 1;     else isPerfect = 0;
if sigma > 2*n     then isAbundant = 1;    else isAbundant = 0;
if sigma < 2*n     then isDeficient = 1;   else isDeficient = 0;
if mod(tau,2) = 1 then isSquare = 1;      else isSquare = 0;
```

We now have several numeric properties stored in Boolean variables – or, more precisely, numeric variables containing 0s and 1s. This allows us to reference those properties using a logical operator.

For example, the code below draws a simple random sample of ten square numbers from our dataset. The WHERE clause is concise and descriptive.

```
proc surveyselect data=numbers out=tensquares method=srs n=10 noprint;
   where isSquare;
run;
```

The Boolean variables will also enable us to calculate a proportion by using an average of 0s and 1s. The code below does this by using a GROUP BY clause in PROC SQL. In doing so, we can report on the data using equal intervals.

```
%let interval = 100000;

title1 "Distribution of Select Properties";
title2 "by &interval.";
proc sql;
      select
            &interval.*(FLOOR((n-1)/&interval.))+ 1  as A LABEL = 'From' ,
            &interval.*(FLOOR((n-1)/&interval.) + 1) as B LABEL = 'To' ,

            AVG(isPrime)     as isPrime     FORMAT = PERCENT7.2,
            AVG(isAbundant)  as isAbundant  FORMAT = PERCENT7.2,
            AVG(isDeficient) as isDeficient FORMAT = PERCENT7.2

      from numbers
      group by A, B
      ;
quit;
```

Interesting observations arise from the resulting output. It is evident, for example, that abundant and deficient numbers are distributed in a fairly uniform manner. Primes, on the other hand, become sparse as n grows large.

Distribution of Select Properties
by 100000

From	To	isPrime	isAbundant	isDeficient
1	100000	9.59%	24.8%	75.2%
100001	200000	8.39%	24.7%	75.3%
200001	300000	8.01%	24.7%	75.3%
300001	400000	7.86%	24.8%	75.2%
400001	500000	7.68%	24.8%	75.2%
500001	600000	7.56%	24.8%	75.2%
600001	700000	7.45%	24.7%	75.3%
700001	800000	7.41%	24.8%	75.2%
800001	900000	7.32%	24.7%	75.3%
900001	1000000	7.22%	24.8%	75.2%

GAUSS' CIRCLE PROBLEM

Now that we have developed a robust dataset, let us use it to explore a classic problem in Number Theory involving Diophantine equations.

What are the solutions to $x^2 + y^2 = n$ *where* x, y *and* n *are integers?*

The following theorem tells us that the existence of integer solutions (x, y) are entirely dependent on the value of n.

A positive integer n can be represented as the sum of two squares if and only if
the prime factorization of n contains no odd powers of primes congruent to 3 modulo 4.

Using this theorem as a guide, we can compile all candidate values for the right side of the equation using our stored prime decomposition:

```
data rightside;
   set numbers;

   array prime{*} prime:;
   array power{*} power:;

   do i =1 to dim(prime);
      if prime[i] eq . then leave;
         else do;
            if mod(power[i],2)=1 then do;
               if mod(prime[i],4)=3 then delete;
            end;
         end;
   end;

run;
```

For the left side of the equation, we need only to compile all square integers:

```
data leftside;
   set numbers;
   where isSquare;
run;
```

Instead of finding specific solutions for our equation, we will focus on finding the *number of solutions* for a given value of *n*. The number of distinct ways that an integer n can be expressed as the sum of two squares is denoted as $r_2(n)$ and a table of such values for n can be generated by the code below. Please note that merging large datasets is computationally expensive and queries such as these could run for an extended time.

```
proc sql;
      create table gauss as
      select z.n as n, 4*count(z.n) as r2
      from
            leftside as x, leftside as y,
            rightside as z
      where
            (x.n + y.n  = z.n)
            or
            (x.n = y.n and y.n = z.n)
      group by z.n;
quit;
```

We should note that our data set only contains one of the four solution pairs, $(+x,+y)$, and not $(+x,-y)$, $(-x,+y)$ or $(-x,-y)$. Consequently, some adjustments in logic were needed to allow for solutions containing negative integers and zero. The new dataset GAUSS contains *n* as well as $r_2(n)$.

The problem of estimating $r_2(n)$ for large n is known as *Gauss' Circle Problem* – "circle" because the resulting solution has implications for counting points with integer coordinates contained within a circle centered at the origin. As Gauss noted, the value of $r_2(n)$ has a curious property:

$$\lim_{N\to\infty} \frac{\sum_{n=0}^{N} r_2(n)}{N} = \pi$$

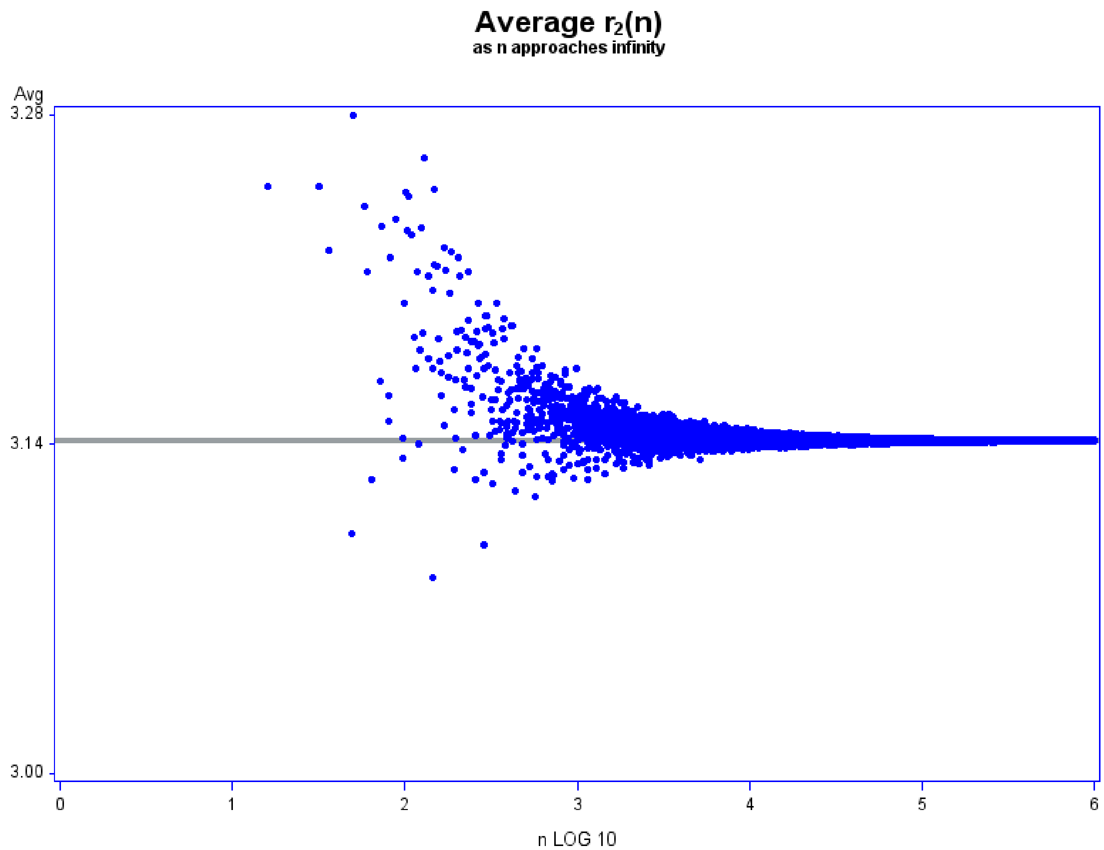
The statement claims that while the value of $r_2(n)$ is certainly integral, the *average* value of $r_2(n)$ approaches π for increasingly large *n*. The calculation can be made for all values in our data by using a RETAIN statement as shown here:

```
data circle;
      set gauss;
      retain summation;
      if _N_ = 1 then summation = r2;
      summation = summation  + r2;
      Avg = summation / n;
run;
```

The limit can now be visualized with a scatterplot.

```
symbol1 color=blue v=dot interpol=none height=1.5;
axis1 logbase=10 logstyle=power;
axis2 order=(3.00 to 3.28 by .14) minor=none;
title1 h=3.5 'Average r' move=(-0,-.3) h=2 '2' move=(+0,+.3) h=3.5 '(n)';
title2 h=2 'as n approaches infinity';

proc gplot data=circle;
   plot  Avg*n/
   haxis=axis1 vaxis=axis2 caxis=blue hminor=0 wvref=3 vref=3.14159265;
   run;
quit;
```



What we see in the above plot is a series of averages converging to the vertical reference line that has been drawn approximately at π.

GOLDBACH'S CONJECTURE

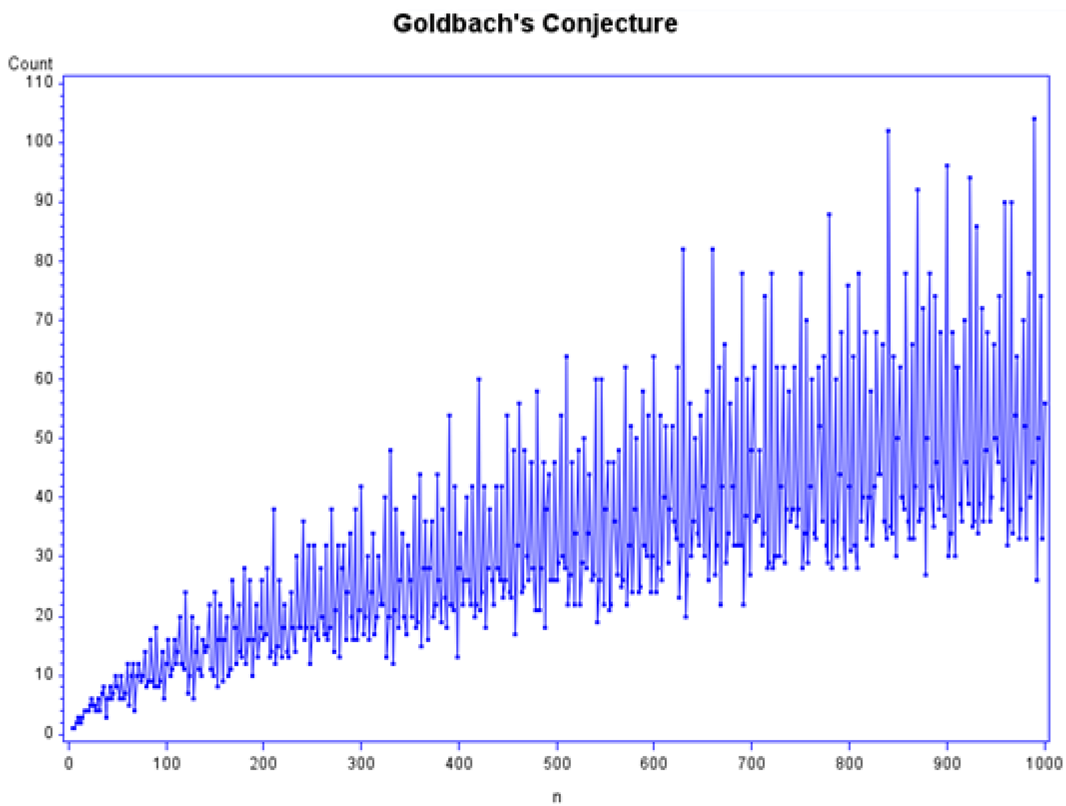
In a similar fashion to our approach to *Gauss' Circle Problem*, we can also visualize one of the most famous unproven conjectures in all of Number Theory, *Golbach's Conjecture*. This conjecture, put forth by Christian Goldbach in the 17th century, states:

Every even integer larger than two is expressible as the sum of two primes.

Following the approach from the previous section, we can develop a solution set. Merging large datasets is computationally expensive, and so the reader is encouraged to develop queries such as these on smaller datasets. Below, we use only a subset of the first 1000 integers from `numbers`.

```
proc sql;
      create table goldbach as
      select z.n as n, count(z.n) as Count
      from numbers as x, numbers as y, numbers as z
      where x.isPrime and y.isPrime and z.isEven and (x.n + y.n  = z.n)
      group by z.n
;
quit;
```

We then plot the resulting data:



In this plot we see evidence in support of *Goldbach's Conjecture* as the number of solutions trend away from zero for large values of *n*.

PRIME NUMBER THEOREM

A final visualization that is worth our attention concerns the famous *Prime Number Theorem*. The theorem concerns the distribution of prime numbers as expressed by the prime counting function:

$\pi(n)$: the number of prime numbers less than or equal to n.

The theorem states that $\pi(n)$ is approximated by $\dfrac{n}{\log n}$ as n approaches infinity.

Stated differently,

$$\lim_{n\to\infty} \frac{\pi(n)}{n/\log n} = 1$$

In a letter to a friend, Gauss (1863) recalled developing the above formula around the year 1792. Nearly 5 years later, another mathematician, Adrien-Marie Legendre, independently developed an estimate that is similar but contains an important difference:

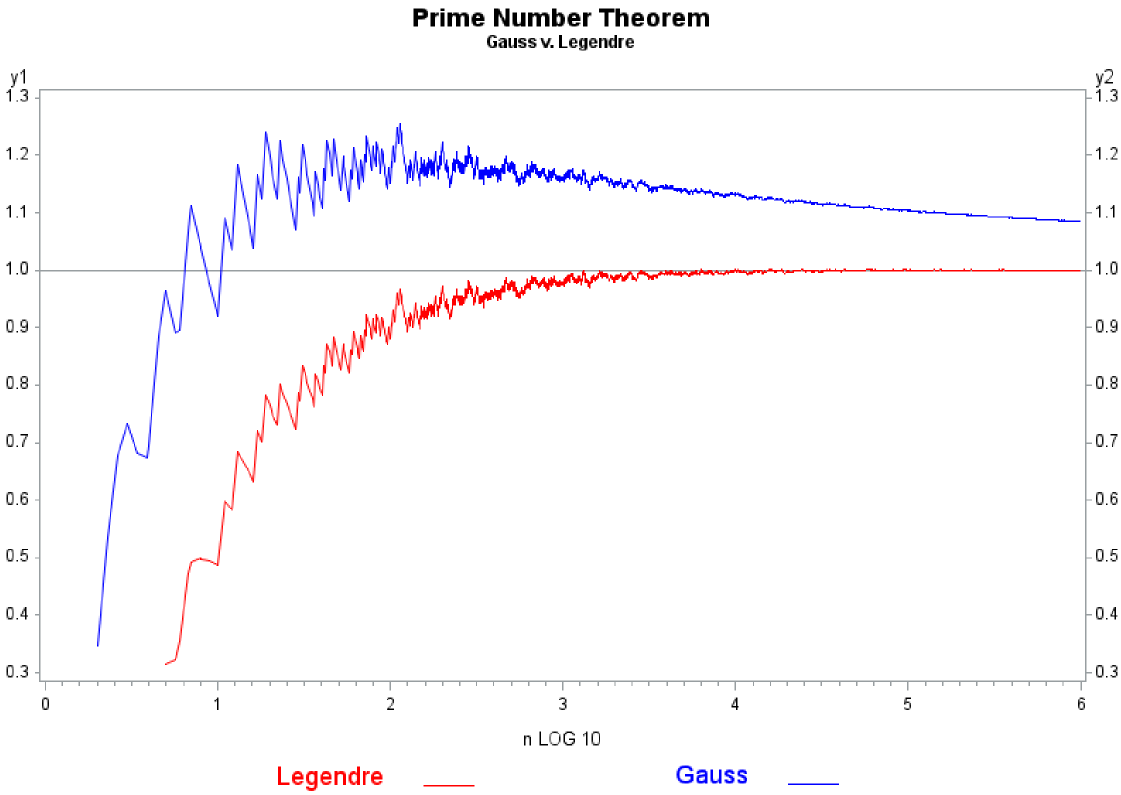
$$\lim_{n\to\infty} \frac{\pi(n)}{n/\log n - B(n)} = 1$$

where the value $B(n)$ approaches an irrational value known as Legendre's constant (1.08366…).

A comparison of the two formulas can be made by calculating $\pi(n)$ directly and also calculating the two estimates.

```
data primes;
      set numbers;
      retain pi;
      if _N_ = 1 then do; pi=0; end;
      else do;
            if isPrime then pi = pi + 1;         /* prime counting function */
            gauss = pi/(n/(log(n)));             /* Gauss' original limit*/
            legendre = pi/(n/(log(n)-1.08366)); /* Legendre's improvement */
      end;
run;
```

We can overlay the two plots with PROC GPLOT and observe the differences in the two formulas.



The two plots are displayed on the same vertical scale with a reference line that allows us to observe the asymptotic behavior of each formula with the true value of $\pi(n)$. While each formula is known to converge, it is clear to see here that Legendre's estimate does so more quickly. It is also interesting note that Gauss' (1863) story, if true, would mean that he developed his formula at the age of about 15.

CONCLUSION

The reader has now been given code segments relating to a few core topics in Number Theory. While the discussions contained in this paper barely scratch the surface of what can be done, the general approaches can be extended to address a variety of other topics. These techniques, paired with the power of Base SAS, students and educators will find a mighty arsenal for exploring, illustrating and visualizing an endless supply of concepts.

REFERENCES

Andrews, G. (1994). *Number theory*. New York, NY: Dover Publications.

Gauss, C. F. (1863). Letter to Eucke. In *Werke* (Bd. 2), Göttingen, Germany: Author.

Guy, R. (2004a). Behavior of $\phi(\sigma(n))$ and $\sigma(\phi(n))$. In *Unsolved problems in number theory* (3rd ed., pp. 138–151). New York, NY: Springer-Verlag.

Guy, R. (2004b). *Unsolved problems in number theory* (3^{rd} ed.). New York, NY: Springer-Verlag.

LeVeque, W. (1990). *Elementary theory of numbers*. New York, NY: Dover Publications.

Ore, Ø. (1988). *Number theory and its history*. New York, NY: Dover.

Weisstein, E. W. (n.d.). *Gauss's circle problem*. Retrieved from the MathWorld—A Wolfram Web Resource site: http://mathworld.wolfram.com/GausssCircleProblem.html

ACKNOWLEDGMENTS

I would like to thank all friends and colleagues at ETS who are endlessly willing to discuss topics such as those discussed here. Special thanks to Amanda Cirillo and Shari Rutherford for their feedback on this paper. Lastly, I would like to thank my wife Robin, who provides me with unending support in all that I do.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Matthew Duchnowski
Educational Testing Service
Rosedale Rd, MS 20T Princeton, NJ 08541
(609) 683-2939
mduchnowski@ets.org