Kernel: SageMath 10.4

Counting Primes and the Riemann Zeta Function

Math 242 Modern Computational Math

Today we will continue our study of the *prime counting function* $\pi(x)$, which is defined to be the number of primes less than or equal to x.

Remember our big question from last time: ***What is the shape of the prime counting function?*** In other words, can we approximate the prime counting function with simpler functions?

Here is our sieve of Eratosthenes function from a previous class session:

In [3]:
```
# Sieve of Eratosthenes
def sieveEratos(nMax):
    # initialize a list of numbers
    nums = list(range(2, nMax+1))

    # initalize index
    i = 0

    # main loop
    while nums[i] <= sqrt(nMax):
        # check whether we have reached the next nonzero number
        if nums[i] > 0:
            # replace all multiples of p=nums[i] with zero
            j = i + nums[i] # first position to replace with zero
            while j < len(nums):
                nums[j] = 0
                j += nums[i]  # increment j by p=nums[i]
        # increment i
        i += 1

    return [n for n in nums if n != 0] # use a list comprehension to
select nonzero values
```

Here is our function from Friday that returns a list of values of the prime counting function $\pi(x)$.

In [4]:
```
# returns a list of values of the prime-counting function π(x), for
integers x from 1 to nMax
def computePiVals(nMax):
    # compute a list of primes up to nMax
    primeList = sieveEratos(nMax)

    # make a list of nMax+1 zeros
    piVals = [0]*(nMax+1)

    # track how many primes we've found so far
    count = 0
```

```
    # loop over integers i from 2 to nMax
    for i in range(2, nMax + 1):
        # if i is the next prime, then add 1 to our count
        if count < len(primeList) and i == primeList[count]:
            count += 1  # we found the next prime

        # store the current count in piVals[i]
        piVals[i] = count

    # return the list of piVals
    return piVals
```

Let's make a big list of $\pi(x)$ values for integers x from 0 up to some large M.

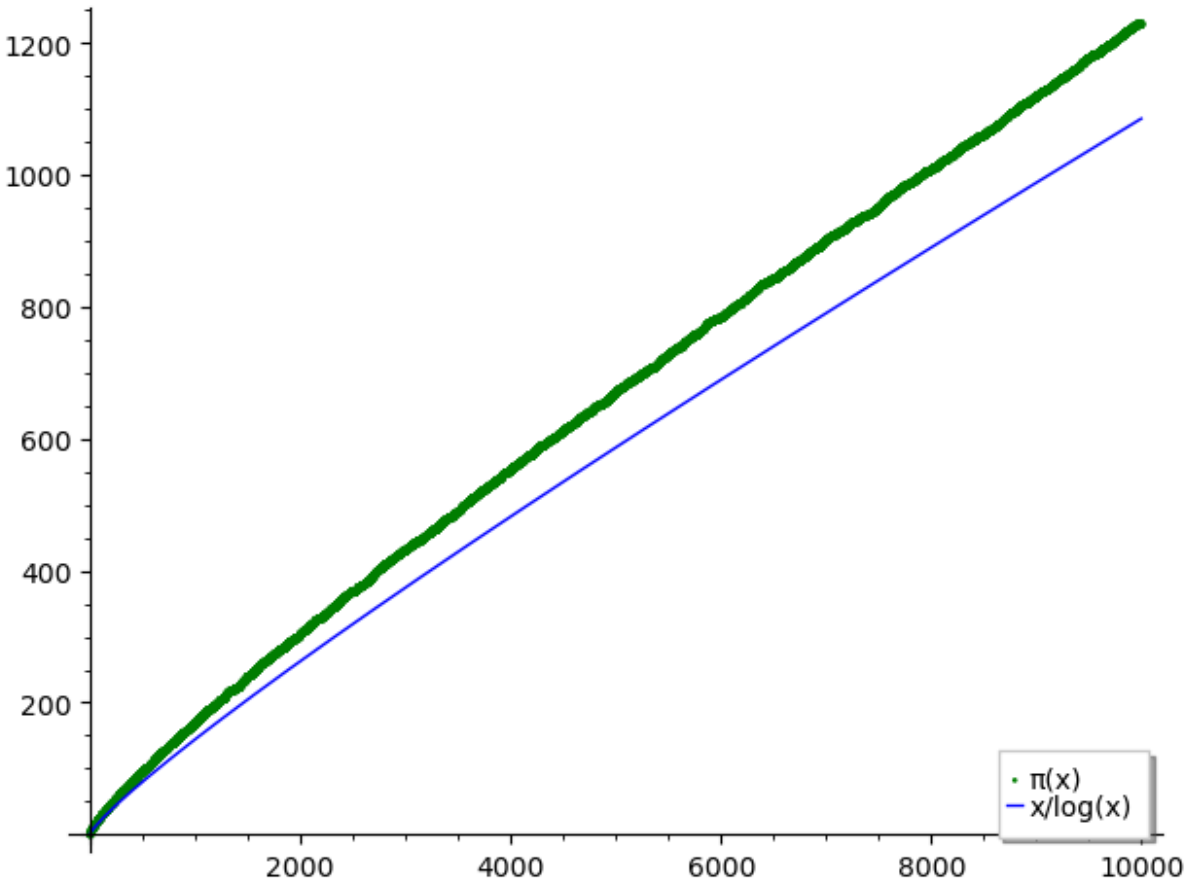
In [5]:
```
nMax = 1000000
piVals = computePiVals(nMax)
piVals[:20]
```

Out[5]: [0, 0, 1, 2, 2, 3, 3, 4, 4, 4, 4, 5, 5, 6, 6, 6, 6, 7, 7, 8]

In the homework for today, we considered $\frac{x}{\ln(x)}$ as a possible approximation of $\pi(x)$. Make a plot comparing $\frac{x}{\ln(x)}$ and $\pi(x)$.

In [6]:
```
xMin=2
xMax=10000
combinedPlot = list_plot(piVals[:xMax], color="green",
legend_label="π(x)") + plot(x/log(x), (x, xMin, xMax),
legend_label="x/log(x)")
combinedPlot.show(legend_loc="lower right")
```

Out[6]:



Density of Primes

Roughly speaking, the *density* of primes near x is the proportion of integers near x that are prime. We can approximate the density of primes near x by fixing some length ℓ and computing the proportion of integers in the interval $(x, x+\ell)$ that are prime. In other words, the density of primes near x is approximately

$$\frac{\pi(x+\ell) - \pi(x)}{\ell}$$

Complete the following function that approximates the density of primes near x:

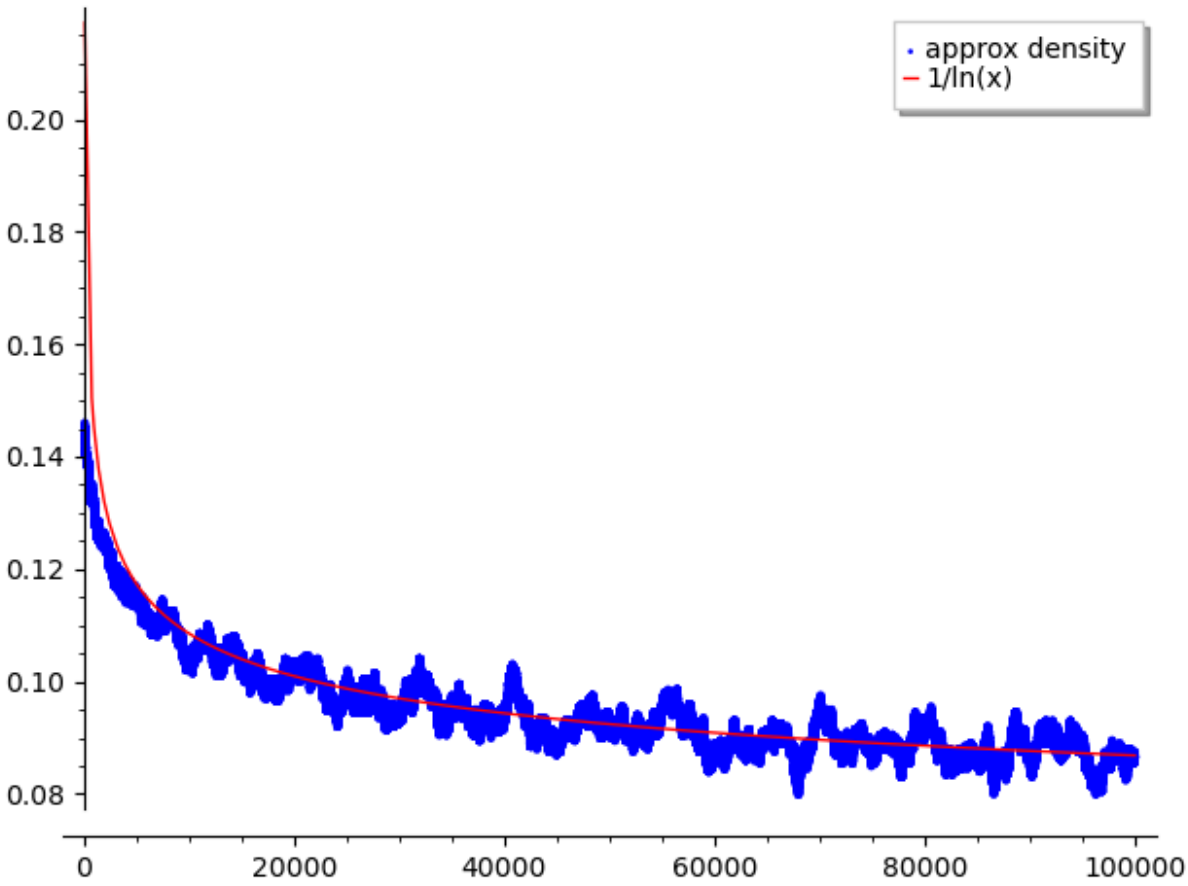
In [7]:
```
def primeDensity(x, length):
    return (piVals[x+length] - piVals[x])/length
```

Make some plots of your prime density function for different values of `x` and `length`.

In [8]:
```
xVals = range(100,100000)
denseVals = [primeDensity(x, 2000) for x in xVals]

list_plot( list(zip( xVals, denseVals)) , legend_label="approx
density") + plot(1/log(x), (x,100,100000), color="red",
legend_label="1/ln(x)")
```

Out[8]:



In [0]:

In the late eighteenth century, using printed tables of prime numbers, Gauss conjectured that the density of primes near x is approximately $\frac{1}{\ln(x)}$. Can you provide computational evidence for or against Gauss's conjecture?

In [0]:

In [0]:

The Logarithmic Integral

If the density of primes near x is approximately $\frac{1}{\ln(x)}$, then the *count* of primes up to x should be approximately the integral $\int_0^x \frac{1}{\ln(t)} dt$. This integral has a special name and notation:

Definition: The *logarithmic integral*, $\mathrm{li}(x)$ is defined

$$\mathrm{li}(x) = \int_0^x \frac{1}{\ln(t)} dt.$$

This integral is a bit tricky to compute, but fortunately it is already implemented in Sage as `li()` and also as `log_integral()`.

In [20]:
```
li?
```

```
Out[20]: Type:           Function_log_integral
         String form:    log_integral
         File:           /ext/sage/10.4/src/sage/functions/exp_integral.py
         Docstring:
            The logarithmic integral \operatorname{li}(z) defined by

               \operatorname{li}(x) = \int_0^z \frac{dt}{\ln(t)} =
               \operatorname{Ei}(\ln(x))

            for x > 1 and by analytic continuation for complex arguments z (see
            [AS1964] 5.1.3).

            EXAMPLES:

            Numerical evaluation for real and complex arguments is handled
            using mpmath:

               sage: N(log_integral(3))
               2.16358859466719
               sage: N(log_integral(3), digits=30)
               2.16358859466719197287692236735
               sage: log_integral(ComplexField(100)(3+I))
               2.2879892769816826157078450911 + 0.87232935488528370139883806779*I
               sage: log_integral(0)
               0

            Symbolic derivatives and integrals are handled by Sage and Maxima:

               sage: x = var('x')
               sage: f = log_integral(x)
               sage: f.diff(x)
               1/log(x)
               sage: f.integrate(x)
               x*log_integral(x) - Ei(2*log(x))

            Here is a test from the mpmath documentation. There are
            1,925,320,391,606,803,968,923 many prime numbers less than 1e23.
            The value of "log_integral(1e23)" is very close to this:

               sage: log_integral(1e23)
               1.92532039161405e21

            ALGORITHM:

            Numerical evaluation is handled using mpmath, but symbolics are
            handled by Sage and Maxima.

            REFERENCES:

            * https://en.wikipedia.org/wiki/Logarithmic_integral_function

            * mpmath documentation: logarithmic-integral
         Init docstring:
            See the docstring for "Function_log_integral".

            EXAMPLES:

               sage: log_integral(3)
               log_integral(3)
               sage: log_integral(x)._sympy_()
               li(x)
```

```
      sage: log_integral(x)._fricas_init_()
      'li(x)'
Call docstring:
   Evaluate this function on the given arguments and return the
   result.

   EXAMPLES:

      sage: exp(5)
      e^5
      sage: gamma(15)
      87178291200

   Python float, Python complex, mpmath mpf and mpc as well as numpy
   inputs are sent to the relevant "math", "cmath", "mpmath" or
   "numpy" function:

      sage: cos(1.r)
      0.5403023058681398
      sage: assert type(_) is float
      sage: gamma(4.r)
      6.0
```

```
sage: assert type(_) is float

sage: cos(1jr)  # abstol 1e-15
(1.5430806348152437-0j)
sage: assert type(_) is complex

sage: import mpmath
sage: cos(mpmath.mpf('1.321412'))
mpf('0.24680737898640387')
sage: cos(mpmath.mpc(1,1))
mpc(real='0.83373002513114902', imag='-0.98889770576286506')

sage: import numpy
sage: sin(numpy.int32(0))
0.0
sage: type(_)
<class 'numpy.float64'>
```

In [9]:
```
n(li(1000))
```

Out[9]: 177.609657990152

In [23]:
```
piVals[1000]
```

Out[23]: 168

Compute some values of $\mathrm{li}(x)$. How do they compare to $\pi(x)$ and $\frac{x}{\ln(x)}$?

In [24]:
```
n(li(100000))
```

Out[24]: 9629.80900105080

In [26]:
```
piVals[100000]
```

Out[26]: 9592

In [11]:
```
n(100000/ln(100000))
```

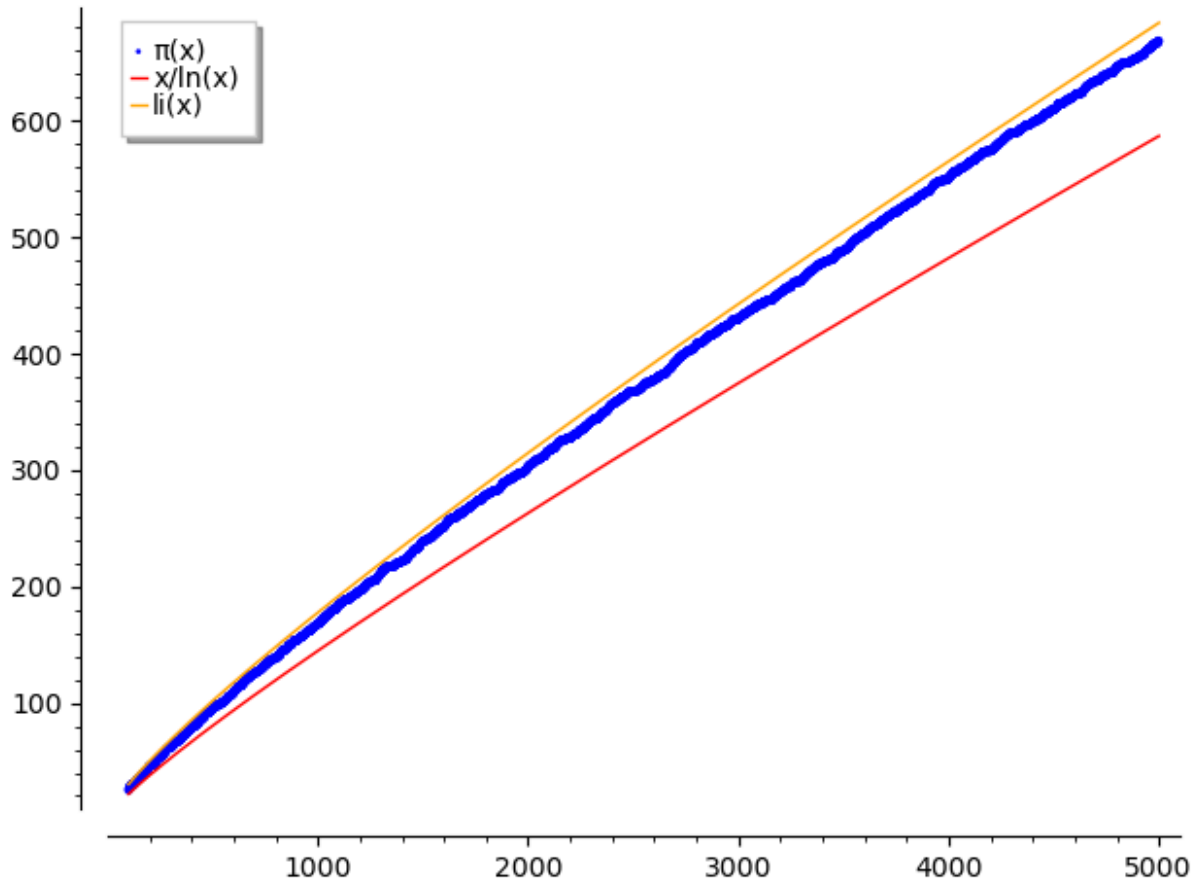
Out[11]: 8685.88963806503

Make a plot showing the values of $\pi(x)$, $\frac{x}{\ln(x)}$, and $\mathrm{li}(x)$. What do you observe?

In [12]:
```
xMin = 100
xMax = 5000
xVals = range(xMin,xMax)

list_plot( list(zip( xVals,piVals[xMin:xMax])) , legend_label="π(x)")
+ plot(x/log(x), (x,xMin,xMax), color="red", legend_label="x/ln(x)") +
plot(li(x), (x, xMin,xMax), color="orange", legend_label="li(x)")
```

Out[12]:



In [0]:

The Riemann Zeta Function

To find an even better approximation to the prime counting function, we must learn about a function called the *Riemann zeta function*. This function leads to one of the most important open questions in mathematics, the *Riemann Hypothsis*, which is a statement about the zeros of the Riemann zeta function.

The Riemann zeta function, $\zeta(s)$, is defined

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s} = \frac{1}{1^s} + \frac{1}{2^s} + \frac{1}{3^s} + \frac{1}{4^s} + \frac{1}{5^s} + \cdots$$

The Riemann zeta function converges for all $s > 1$. Moreover, it converges for all *complex* numbers s with real part greater than 1. However, for $s = 1$ the sum diverges, so $\zeta(1)$ is undefined.

Write a Python function below that approximates $\zeta(s)$ by computing a partial sum of `numterms` terms.

In [38]:

```
# compute a partial sum consisting of numTerms terms of ζ(s)
def zeta(s, numTerms):
    total = 0
```

```
    for m in range(1, numTerms):
        total += 1/(m^s)
    return total
```

Use your `zeta` function to compute decimal approximations of $\zeta(2), \zeta(3), \zeta(4), \ldots$. What do you notice? Can you find closed-form expressions for any of these values?

In [42]:
```
n(zeta(2, 10000))
```

Out[42]: 1.64483406184806

In [0]:

In [0]:

One connection between the Riemann zeta function and the prime numbers has to do with the following product:

$$\prod_{p \text{ prime}} \frac{1}{1 - p^{-s}}$$

This is a product over all prime numbers p.

Write a Python function below that computes partial products of this infinite product. Take the primes in order, starting with the smallest prime.

In [0]:
```
def primeProduct(s, numFactors):

  # YOUR CODE HERE
```

In [0]:

In [0]:

What is the connection between $\zeta(s)$ and $\prod_{p \text{ prime}} \frac{1}{1-p^{-s}}$?