Kernel: SageMath 10.7

Prime Explorations

MATH 242 Modern Computational Mathematics

Sieve of Eratosthenes

Here is an implementation of the sieve of Eratosthenes.

In [3]:
```
def sieveEratosthenes(nMax):
    # initialize a list
    # if you start the list from zero, then each number in the list is
EQUAL to its index in the list, which is very convenient
    nums = list(range(0, nMax + 1))

    # we will replace non-prime numbers in the list with zeros
    # start by replacing 1 with 0
    nums[1] = 0

    # loop over list items until we reach sqrt(nMax)
    i = 2
    while i <= sqrt(nMax):
        # if nums[i] is not zero, then it is prime
        if nums[i] > 0:
            # replace all multiples of nums[i] with zeros
            for j in range(2*i, nMax + 1, i):
                nums[j] = 0

        # testing: print the current state of nums
        #print(f"finished i={i}: nums={nums}\n")

        # increment i
        i += 1

    # return a list containing all nonzero elements of nums
    return [i for i in nums if i != 0]
```

In [4]:
```
sieveEratosthenes(20)
```

Out[4]:
```
[2, 3, 5, 7, 11, 13, 17, 19]
```

How efficient is the sieve of Eratosthenes?

In [8]:
```
%time primeList = sieveEratosthenes(10^6)
```

Out[8]:
```
CPU times: user 1.12 s, sys: 36.5 ms, total: 1.16 s
Wall time: 1.29 s
```

In [10]:
```
%timeit primeList = sieveEratosthenes(10^6)
```

Out[10]:
```
1.19 s ± 36.8 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [13]:
```
import time
startTime = time.time()
primeList = sieveEratosthenes(10^6)
endTime = time.time()
timeElapsed = endTime - startTime
timeElapsed
```

Out[13]:
```
1.2760725021362305
```

In [14]:
```
def measureRuntime(nMax):
    startTime = time.time()
    primeList = sieveEratosthenes(nMax)
    endTime = time.time()
    return endTime - startTime
```

In [15]:
```
measureRuntime(10^6)
```

Out[15]:
```
1.5440020561218262
```

In [18]:
```
nMaxVals = range(10^5, 10^6 + 1, 10^5)
runTimes = [measureRuntime(m) for m in nMaxVals]
runTimes
```

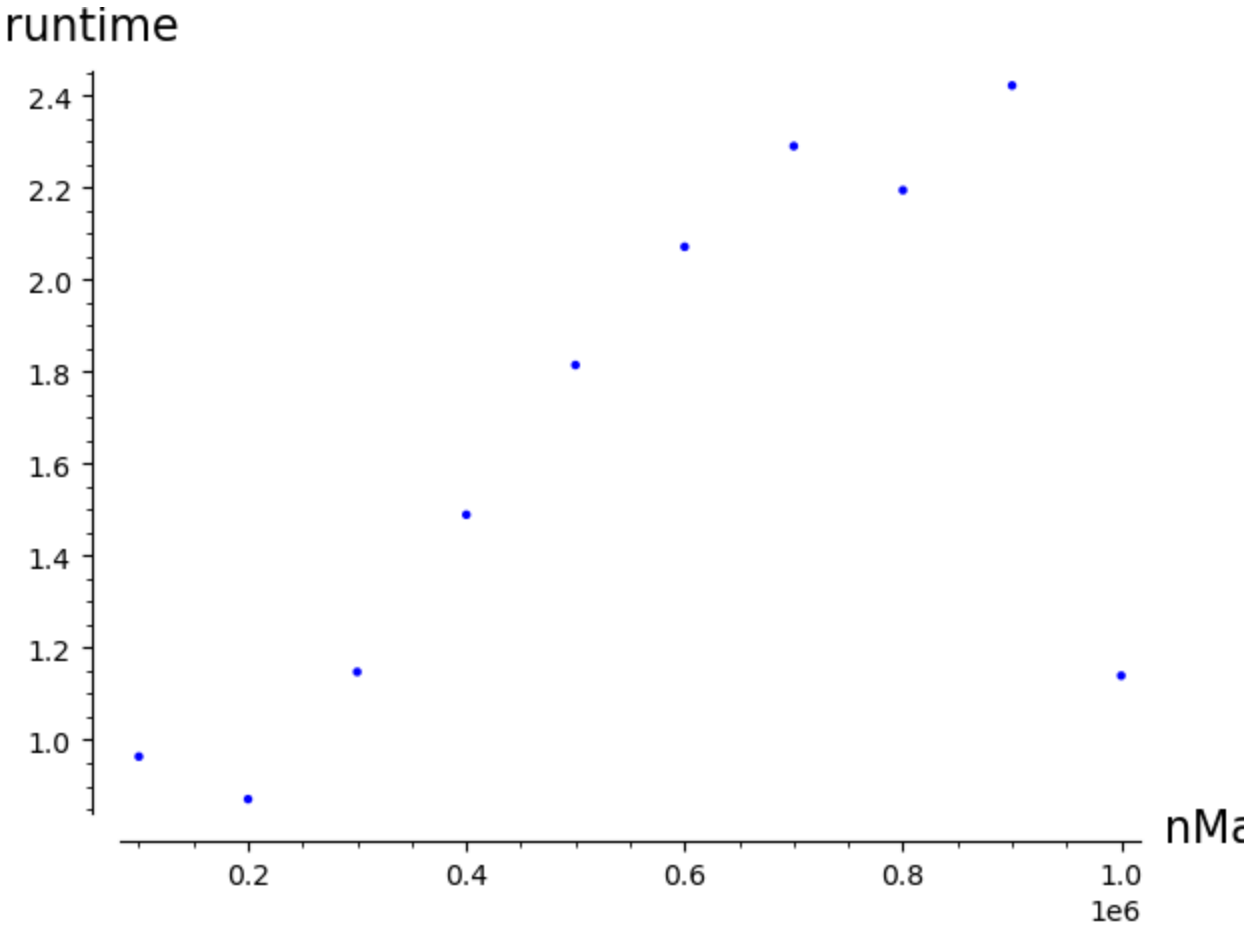
Out[18]:
```
[0.9639081954956055,
 0.8714134693145752,
 1.1476995944976807,
 1.4889991283416748,
 1.813976526260376,
 2.0706636905670166,
 2.2892189025878906,
 2.1935744285583496,
 2.4208765029907227,
 1.1395301818847656]
```

In [22]:
```
list(zip(nMaxVals, runTimes))
```

Out[22]:
```
[(100000, 0.9639081954956055),
 (200000, 0.8714134693145752),
 (300000, 1.1476995944976807),
 (400000, 1.4889991283416748),
 (500000, 1.813976526260376),
 (600000, 2.0706636905670166),
 (700000, 2.2892189025878906),
 (800000, 2.1935744285583496),
 (900000, 2.4208765029907227),
 (1000000, 1.1395301818847656)]
```

In [23]:
```
list_plot( list(zip(nMaxVals, runTimes)), axes_labels=["nMax","runtime"])
```

Out[23]:



Units Digits of Primes

The only primes with a units digit of 2 or 5 are the primes 2 and 5. All other primes have a units digit of 1, 3, 7, or 9. How often do these digits occur?

Warm-Up

Consider the primes less than 100. Of these primes, count how many have each units digit 1, 3, 7, and 9.

In [0]:

In [0]:

Exploration

Now replace 100 by some other integer M. How many primes less than or equal to M have each units digit 1, 3, 7, and 9? Consider various values of M.

- What patterns do you observe in your counts?
- What questions arise during your exploration?
- What conjectures can you make?

In [4]:
```
def countByUnit(primeList):
    counts = [0]*4
    for p in primeList:
        u = p % 10
        if u == 1:
            counts[0] += 1
        elif u == 3:
            counts[1] += 1
        elif u == 7:
            counts[2] += 1
        elif u == 9:
            counts[3] += 1
    return counts
```

In [37]:
```
primeList = sieveEratosthenes(100)
countByUnit(primeList)
```

Out[37]: [5, 7, 6, 5]

In [5]:
```
primeList = sieveEratosthenes(500)
countByUnit(primeList)
```

Out[5]: [22, 24, 24, 23]

In [6]:
```
primeList = sieveEratosthenes(1000)
countByUnit(primeList)
```

Out[6]: [40, 42, 46, 38]

In [7]:
```
primeList = sieveEratosthenes(10000)
countByUnit(primeList)
```

Out[7]: [306, 310, 308, 303]

It looks like the counts of primes ending with 3 and 7 are slightly higher than the counts of primes ending with 1 and 9. Is this always the case?

Extension

Instead of counting primes by their units digits, what if you count the primes by their remainders after division by some other number? For example:

- Are there more primes less than or equal to M that are 1 more than a multiple of 3 or 2 more than a multiple of 3? How does this depend on M?
- How many primes less than or equal to M are 1 more than a multiple of 8? ...3 more than a multiple of 8? ...5 more than a multiple of 8? ...7 more than a multiple of 8? How do these counts depend on M?
- What questions arise during your exploration?
- What conjectures can you make?

In [0]:

In [0]: