AN ALGORITHMIC VERSION OF

THE SIEVE OF ERATOSTHENES

PLUS OTHER SOLUTIONS ASSOCIATED

WITH PRIME NUMBERS.

(Version 2).

Peter G.Bass.

www.relativitydomains.com

M19 Ver. 2.0.0
February 2020.

ABSTRACT.

This paper develops an algorithmic version of the Sieve of Eratosthenes to identify all the prime numbers up to and including any desired Natural number. The method also provides for the simple calculation of $\pi(N)$.
In addition, the primary term in the algorithm enables a new and very simple method for the determination of primality or compositeness of any number and, in the case of the latter, the factors involved, which enables these parameters to be easily determined for very large composite numbers.

In version 2.0.0 all of the above determinations have been implemented via two Python programs.

CONTENTS.

1.0 Introduction.

Some two hundred and fifty years BC, a Greek mathematician, Eratosthenes of Cyrene, devised a method of identifying all the prime numbers, by elimination of the composites from the group of Natural numbers. Today this method is known as the Sieve of Eratosthenes, and is still the most successful method of producing prime numbers. However, it suffers from the difficulty that, as the method progresses into extremely large numbers, it becomes more and more difficult to identify the composites.

The method presented here eliminates this difficulty by incorporating a simple algorithm for the generation of all the composites, so enabling the fast identification of the primes. This algorithm also provides the means to determine three other important parameters. They are (i) The instantanious determination of an exact value for $\pi(N)$. (ii) A very simple method for determining whether any odd number is prime or composite, and (iii) a very simple method for determining the prime factors for extremely large composite numbers. The method is implemented by two short Python programs.

The Python programming language was chosen for this implementation because it contains arbitrary precision integer arithmetic, a facility necessary for computation with extremely large numbers. However, to run the programs here it is also necessary to install Python's floating point equivalent.
Python can be downloaded from the web together with its arbitrary precision floating point module. Installation is very simple, but it is important to follow exactly the Python coding protocol when entering the programs here into it.

The program coding below should be entered into Python's IDLE facility and run from there or from the *.py executables.

2.0 An Algorithmic Version of the Sieve of Eratosthenes.

2.1 Development of the Algorithm.

It is well known that the Natural numbers are the combination of all the primes and all the composites, thus

$$[N] = [P] \wedge [M] \tag{2.1}$$

where

- $[N]$ is the group of Natural numbers.
- $[P]$ is the group of prime numbers
- $[M]$ is the group of composite numbers
- $\wedge$ represents a Boolean type operator "AND"

The group of composite numbers can be represented as

$$[M] = [E] \wedge [O] \tag{2.2}$$

where

- $[E]$ is the group of even composites = $[2(n + 1)]$.
- $[O]$ is the group of odd composites = $[Q(2n + Q - 2)]$.

where Q is the group of odd prime numbers, and where $n = 1 \rightarrow \infty$.

Substitution of (2.2) as defined into (2.1) then yields

$$[N] = [P] \wedge [2(n+1)] \wedge [Q(2n+Q-2)] \tag{2.3}$$

So that

$$[P] = [N] \vee [2(n+1)] \vee [Q(2n+Q-2)] \tag{2.4}$$

and where

$\vee$ represents a Boolean type operator "NOT".

To avoid the presence of [*Q*] on the right hand side of (2.4), it is replaced with the group of all odd numbers, i.e. [2*m* + 1] thus

$$[P] = [N] \vee [2(n+1)] \vee [(2m+1)\{2n+(2m+1)-2\}] \tag{2.5}$$

and this reduces to

$$[P] = [N] \vee [2(n+1)] \vee [4m^2+4mn+2n-1] \tag{2.6}$$

Where now *m* and $n = 1 \rightarrow \infty$ independently. Appendix A.1 shows that putting *n* = 0 in (2.6) is not applicable.

Eq(2.6) can now be used to identify primes as *m* and *n* are varied. The second term produces all the even composites, leaving just the number 2 as the only even prime. The third term produces all the odd composites, the remaining odd numbers thereby being prime.
There are two minor anomalies that result. The first is that replacing [*Q*] by [2*m* + 1] in (2.4) means that the third term in (2.6) will produce a number of duplicate composites. This however, is eliminated in program code. The second anomaly is that because the distribution of composites with regard to *m* and *n* is irregular, the composites, and therefore primes, will not be identified in the correct order. This is eliminated by sorting, again in the program code.

Note, in the remainder of this paper, only the odd composites/primes will be considered, so that only the third term in (2.6) is relevant. Consequently, that term is now separately identified thus

$$[O] = [4m^2+4mn+2n-1] \tag{2.7}$$

2.2 Identification of the All the Primes up to Some Number *N*, (Python Program #1, Implementation 1).

To effect this exercise it is first necessary to equate (2.7) to *N* and re-arrange as follows, working with individual terms inside the group.

$$n = \frac{N-4m^2+1}{2(2m+1)} \tag{2.8}$$

Eq.(2.8) determines the maximum value of *n* for all values of *m* from unity upwards until *n* falls below unity. Also, only the integer part of *n* so determined is taken. This ensures that the list of composites produced by (2.7), up to and including *N*, with these values of *m* and *n*, is complete, so also ensuring that the list of all remaining odd numbers, being prime, is also complete.

The program to effect this implementation is shown below. Note that printing of composites has been # out because these are at least 5+ times the number of primes, and should only be printed if specifically desired. To print composites, remove the # before running the program.

```
print("\nThis program will determine all the composite numbers between any two ")
print("designated Natural numbers and thereby all the Primes Within the Same Range. ")

import time
import math
from mpmath import mpf
from mpmath import mp

mp.dps=4

Nh=int(input("\nEnter the High Value of N, (Must be Even):- "))
Nl=int(input("\nEnter the Low Value of N, (Must be Even and lower than Nh):- "))

m=1
i=1
c=1
n1=1
n2=1
q=1
inventory1=[]
inventory0=[]

start=time.clock()

while m>0:
    n1=(Nh-4*m**2+1)//(4*m+2)
    n2=(Nl-4*m**2+1)//(4*m+2)
    if n2<=0:
        n2=0
    if n1<=0:
        break
    while n1>n2:
        c=4*m**2+4*m*n1+2*n1-1
        if c not in inventory1:
            inventory1+=[c]
            n1 -=1
        else:
            n1 -=1
    m=m+1

inventory1.sort()

while q<Nh:
    q=Nl+i
    if q>Nh:
        break
    if q not in inventory1:
```

```
        inventory0 +=[q]
        i +=2
    else:
        i +=2

#print("\nComposites are:-")
#print(inventory1)
print("\nPrimes are:-")
print(inventory0)
pi=len(inventory0)
print("\nPi(N) is: "),
print(pi)

finish=time.clock()
run=finish-start
hours=int(run/3600)
run=run-hours*3600
minutes=int(run/60)
seconds=run-minutes*60

print("\n\nRun Time is:- ")
print(hours, minutes, seconds)

print("\nPress the ENTER Key to Exit. ")
```

Table 2.1 – Program #1.

This program will print (on screen), when actioned, all the composite odd Natural numbers from 0 to *N* followed by all the primes. Duplicates have been eliminated and sorting effected in the code. An exact value of $\pi(N)$ has also been determined.

2.3 A Simple Test for Primality or Compositeness, (Python Program #1, Implemetation 2).

To effect this test for any particular Natural number *N*, simply enter into the above program #1 a value of Nh = *N* + 3, and Nl = *N* – 3. The program will then show whether *N* is prime or composite. Note that other outcomes may result, (i) Nh-1 is prime with *N* and Nl+1 composite, (ii) Nl+1 is prime with *N* and Nh-1 composite, or (iii) N may be one of a twin prime with either Nh-1 or Nl+1.

2.4 Determination of the Factors of Very Large Composite Numbers, (Python Program #2).

The program to effect this implementation is shown below.

```
print("\nThis Program will Factorize any odd Natural composite number.\n ")

import time
import math
from mpmath import mp
from mpmath import mpf

N=int(input("Enter the Value of the Number, N, to be Factorized, (Must be Odd):- "))

mp.dps=20
```

```
inventory=[]
f=3
fmax=int(N**0.5)
print("\n")
start=time.clock()

while f<fmax:
    if N= =1 or N<f:
        break
    n=mpf(N-f**2)/(2*f)-1
    if n= =int(n):
        inventory +={f}
        N=mpf(N)/f
        print(f),
        fmax=int(N**0.5)
        f +=2
    else:
        f +=2
if inventory= =[]:
    print("No Factors, N is Prime")
elif N>1:
    N=int(N)
    inventory +=[N]
    print("\n\nThe Factors are:- ")
    print(inventory)

finish=time.clock()
run=finish-start
hours=int(run/3600)
run=run-hours*3600
minutes=int(run/60)
seconds=run-minutes*60

print("\n\nRun Time is:- ")
print(hours, minutes, seconds)

input("\n\nPress the enter key to exit. ")
```

Table 2.1 – Program #2.

This program will print, (on screen), all the factors of the number inputted, or, if there are none, that the number is prime.

3.0 Examples of Application.

3.1 Program #1, Implementation 1.

The following is a printout of all the composites and primes between $N = 0$ and 1,000.

This program will determine all the composite numbers between any two
designated Natural numbers and thereby all the Primes Within the Same Range.

Enter the High Value of N, (Must be Even):- 1000

Enter the Low Value of N, (Must be Even and Lower than Nh):- 0

Composites are:-
[9, 15, 21, 25, 27, 33, 35, 39, 45, 49, 51, 55, 57, 63, 65, 69, 75, 77, 81, 85, 87, 91, 93, 95,
99, 105, 111, 115, 117, 119, 121, 123, 125, 129, 133, 135, 141, 143, 145, 147, 153, 155,
159, 161, 165, 169, 171, 175, 177, 183, 185, 187, 189, 195, 201, 203, 205, 207, 209, 213,
215, 217, 219, 221, 225, 231, 235, 237, 243, 245, 247, 249, 253, 255, 259, 261, 265, 267,
273, 275, 279, 285, 287, 289, 291, 295, 297, 299, 301, 303, 305, 309, 315, 319, 321, 323,
325, 327, 329, 333, 335, 339, 341, 343, 345, 351, 355, 357, 361, 363, 365, 369, 371, 375,
377, 381, 385, 387, 391, 393, 395, 399, 403, 405, 407, 411, 413, 415, 417, 423, 425, 427,
429, 435, 437, 441, 445, 447, 451, 453, 455, 459, 465, 469, 471, 473, 475, 477, 481, 483,
485, 489, 493, 495, 497, 501, 505, 507, 511, 513, 515, 517, 519, 525, 527, 529, 531, 533,
535, 537, 539, 543, 545, 549, 551, 553, 555, 559, 561, 565, 567, 573, 575, 579, 581, 583,
585, 589, 591, 595, 597, 603, 605, 609, 611, 615, 621, 623, 625, 627, 629, 633, 635, 637,
639, 645, 649, 651, 655, 657, 663, 665, 667, 669, 671, 675, 679, 681, 685, 687, 689, 693,
695, 697, 699, 703, 705, 707, 711, 713, 715, 717, 721, 723, 725, 729, 731, 735, 737, 741,
745, 747, 749, 753, 755, 759, 763, 765, 767, 771, 775, 777, 779, 781, 783, 785, 789, 791,
793, 795, 799, 801, 803, 805, 807, 813, 815, 817, 819, 825, 831, 833, 835, 837, 841, 843,
845, 847, 849, 851, 855, 861, 865, 867, 869, 871, 873, 875, 879, 885, 889, 891, 893, 895,
897, 899, 901, 903, 905, 909, 913, 915, 917, 921, 923, 925, 927, 931, 933, 935, 939, 943,
945, 949, 951, 955, 957, 959, 961, 963, 965, 969, 973, 975, 979, 981, 985, 987, 989, 993,
995, 999]

Primes are:-
[1, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97,
101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191,
193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283,
293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401,
409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509,
521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631,
641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751,
757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877,
881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]

pi(*N*) is: 168

Run Time is:-
(0, 0, 0.08369567942659675)

Press the ENTER Key to Exit.

Table 3.1 – Composites and Primes Between 0 and 1,000.

Here the composites have been printed for interest and comparison with the primes. Number of primes = 168, number of composites = 832. Note that unity is considered prime in this paper.

3.2 Program #1, Implementation 2.

The following is a printout of a test of 536,870,911 for primality. For this test the # next to composites print have been removed.

```
This program will determine all the composite numbers between any two
designated Natural numbers and thereby all the Primes Within the Same Range.

Enter the High Value of N, (Must be Even):- 536870914

Enter the Low Value of N, (Must be Even and Lower than Nh):- 536870908

Composites are:-
[536870911, 536870913]

Primes are:-
[536870909]

Pi(N) is:  1


Run Time is:-
(0, 0, 0.06400554830014563)

Press the ENTER Key to Exit.
```

Table 3.2 – Primality Test of 536,870,911.

Note that this is a case where *N* and Nh-1 are composite but Nl+1 is prime, {Section 2.3 case(ii)}.

3.3 Program #2.

The following is a printout of factoring 98,041,789,572,153

```
This Program will factorize any odd Natural composite number.

Enter the Value of the Number, N, to be Factorized, (Must be Odd):- 98041789572153


3 7 29 139

The Factors are:-
[3, 7, 29, 139, 1158188203]


Run Time is:-
(0, 0, 2.956909870544304)

Press the enter key to exit.
```

Table 3.3 – Factorization of 98,041,789,572,153

The initial print of factors is as they are found as an indication of computation progress.

3.0 Conclusions.

The system developed here, together with the two short Python programs, have provided all the means necessary to determine (i) all the primes within any range of Natural numbers, together with an exact count of $\pi(N)$, (ii) a primality test of any Natural number, and (iii) all the factors of any odd composite Natural number.
When using Program#1, implementation 1 to determine all the primes within a specific range, if that range is excessively large, it may be preferable to split it into a series of sub-ranges of 20,000 or so. This would reduce computation time and allow the complete range to be examined over an extended period of time.

Python is quite adequate for relatively small numbers of tens and low hundreds of digits, but would incur excessively long run times for numbers possessing thousands/millions of digits. To cater for such numbers, the programs here would need to be coded in a low level language such as Assembler.[1]

APPENDIX A.

A.1 Proof that $n = 0$ is not Applicable in (2.6) and (2.7).

In $[E] = [2(n + 1)]$, if $n = 0$ then $E = 2$ which is not composite.

In $[O] = [4m^2 + 4mn + 2n - 1]$, if $n = 0$ then $O = 4m^2 - 1$ and if $m = 1$, then $O = 3$ which is not composite. For other values of $m > 1$, composites are produced, but they are all double duplicates and therefore in the determination of the composites/primes for any N, and/or $\pi(N)$, they do not contribute.

[1] A capability not currently possessed by the author.