

Analysis of a Prime-Representing Constant

G. Nagy*

*Óbuda University, Alba Regia Technical Faculty, Institute of Geoinformatics

nagy.gabor@amk.uni-obuda.hu

ORCID ID: 0000-0001-9453-2291

Abstract—Fridman et al [1] have described an irrational constant, which can be used to generate a sequence of all of the prime numbers. This paper gives a program, which can calculate this constant in arbitrary precision, and presents some analysis with the calculated data.

Index Terms—prime numbers, prime-representing constant, Python

I. INTRODUCTION

Fridman et al describes a sequence in [1]. If the i th element of the sequence of prime numbers denoted by p_i , the described sequence is determined by these expressions:

$$f_1 = \sum_{k=1}^{\infty} \frac{p_k - 1}{\prod_{j=1}^k p_j} \simeq 2.92 \quad (1)$$

$$f_i = \lfloor f_{i-1} \rfloor (f_{i-1} - \lfloor f_{i-1} \rfloor + 1) \quad (2)$$

The described f sequence connects to the sequence of prime numbers, because:

$$p_i = \lfloor f_i \rfloor \quad (3)$$

The first element of the sequence is an irrational constant, which can generate the sequence of prime numbers by the equations (2) and (3). All $\lfloor f_i \rfloor$ is a prime number, and this method provides all of the prime numbers.

II. THE TEST PROGRAM

I have developed a Python [2], [3], [4] program for calculating the prime-representing constant, and doing some calculations with this number.

```
"""
A program for calculating the prime generator constant with
arbitrary precision, and check the sequence of generated numbers.
(See in DOI:10.1080/00029890.2019.1530554)
Author: Nagy Gabor
"""
```

```
#module imports

#the decimal module for calculating with arbitrary precision
import decimal

#the log function from the math module
from math import log

#some simple functions for prime numbers

def isprime(n):
    for i in range(2, int(n**0.5)+1):
        if n%i==0:
            return False
    return True

def nextprime(n):
    i=n
    while True:
        i+=1
        if isprime(i):
            return i
```

```
#Set the number of decimal digits
decdig=1000 # modify this number to change the precision
decimal.setcontext(decimal.Context(prec=decdig+2))

#Set the limits of the increase
qlim=decimal.Decimal(0.1)**(decdig+3)

#Calculate the prime generator constant
c=decimal.Decimal(0)
denominator=1
p=2
while True:
    q=(p-1)/decimal.Decimal(denominator)
    if q<qlim:
        break
    c+=q
    denominator*=p
    p=nextprime(p)
#The prime generator constant is rounded to the required precision
#(the calculation works with two more digits)
c=round(c, decdig)
#Print the value of the prime generator constant
print(f'The prime generator constant with {decdig} digits:')
print(c)

#Calculate the sequence until the first number which is not prime

#the initial list with the first element of the sequence
f=[c]
while True:
    #calculate the next element of the sequence
    f.append(int(f[-1])*(1+f[-1]%1))
    #end the iteration, if the last element is not correct
    if int(f[-1])!=nextprime(int(f[-2])):
        break
#calculated the list of generated primes
#(the last element of the f list is not correct)
pl=[int(x) for x in f[:-1]]
#print the result
print(f'\nThis number provides {len(pl)} prime numbers:')
print(' '.join([str(pj) for pj in pl]))
print(f'\nThe first wrong number: {int(f[-1])}')
print(f'\nThe count of prime's decimal digits:', sum([len(str(pj)) for pj in pl]) )
print(f'\nThe sum of primes's logarithms (base=10):', sum([log(pj, 10) for pj in pl]))
```

This sample program can be downloaded from GitHub¹.

The programs use the decimal Python module. This module provides decimal objects, which represents an number with arbitrary decimal digits. In the first step the program calculate the prime-representing constant (f_1) by the equation (1). This equation contains an infinite summation. The program calculate the elements of this summation until the increase will be less than a limit, which is less than the required precision.

This program can calculate the value of the prime representing constant with arbitrary precision. The first 1000 digits are:

2.920050977316134712092562917112019468002727899
32142671977268253310773377212776612419017811231
75837422983388595531013766557907893508590053204
58879066450802184292252411229314474766432726781
35940161193093481787115189352526460357537952181
08130464201677079773895702501118038092826861786
58154888032203367829269125018738690914047486202
26129121622736838982992945558789156975535538705
18739251538181607957171434240446633933138827942

¹<https://github.com/ngabor/primegenconst>

63272129485703450548197999040464926096485197137
 63913013768679285321042395712078626139752461367
 42059302377616384076898261130632683169598660539
 41841083990772474014702698539309967440776095857
 59049691130821855859902170542405345175550582796
 53606079314120316578636254109249073897393328606
 72772227778457834272139260508254028067617639389
 06299645073523384311614730916065503649336668524
 71602529082686731529397629219137499668462799308
 53615755759049885203735106436548610543326144375
 30514732349448181398754438855296110963025921245
 99095044825003663535855619587169625921692456215
 563334172710729

III. ANALYSIS OF THE CONSTANT

The prime-representing constant is an irrational number, the program can calculate this number only an arbitrary but finite precision. (In the sample program this precision is determined as the decimal digits of the number.) When anybody calculates the sequence from an approximate value, only the first part of the calculated sequence equals to the sequence of prime numbers.

The question arises that what is the connection between the precision (the numbers of decimal digits) and the length of the list of the correct generated prime numbers.

The program calculates the sum of the decimal based logarithm of the correct generated prime numbers. The Table I contains the result of these calculation with some different decimal digits from 10 to 10000.

The results of $\sum_{i=1}^n \log_{10} p_i$ are very close to the numbers of decimal digits. The sums of the decimal digits of the correct generated primes are typically greater than the decimal digits (the precision) of the prime-representing constant.

IV. CONCLUSION

The sequence described [1] can generate the total sequence of prime numbers, but the practical application is limited by the precision of the calculations.

The size of information is very similar in the approximated prime-representing constant and in the sequence of primes, which can be generated by the approximated constant.

REFERENCES

- [1] D. Fridman, J. Garbulsky, B. Glicer, J. Grime, and M. T. Florentin, "A prime-representing constant," *The American Mathematical Monthly*, vol. 126, no. 1, pp. 70–73, 2019.
- [2] G. Van Rossum *et al.*, "Python Programming Language," in *USENIX Annual Technical Conference*, vol. 41, 2007.
- [3] T. E. Oliphant, "Python for scientific computing," *Computing in Science & Engineering*, vol. 9, no. 3, 2007.
- [4] K. J. Millman and M. Aivazis, "Python for scientists and engineers," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 9–12, 2011.

Table I

THE PRIME-REPRESENTING CONSTANT IS CALCULATED WITH DIFFERENT DECIMAL DIGITS (d), AND LOOK FOR THE LAST ELEMENT OF THE SEQUENCE (f_n), WHICH PROVIDES A CORRECT PRIME NUMBER (p_n). THE SUM OF THIS PRIME'S DECIMAL DIGITS AND 10-BASED LOGARITHM ARE ALSO CALCULATED. THIS TABLE PRESENTS SOME EXAMPLE BETWEEN 10 AND 10000.

d	p_n	n	sum of decimal digits	$\sum_{i=1}^n \log_{10} p_i$
10	31	11	18	11,3
20	53	16	28	19,51
30	83	23	42	32,43
40	103	27	52	40,38
50	131	32	67	50,72
60	157	37	82	61,55
70	181	42	97	72,73
80	197	45	106	79,59
90	227	49	118	88,92
100	257	55	136	103,22
200	487	93	250	200,71
300	727	129	358	300,86
400	967	163	460	400,33
500	1201	197	591	503,44
600	1447	229	719	603,31
700	1657	260	843	702,19
800	1889	290	963	799,68
900	2129	320	1083	898,81
1000	2371	351	1207	1002,75
1500	3533	494	1779	1498,44
2000	4679	633	2335	2000,42
2500	5849	768	2875	2502,86
3000	6983	898	3395	2997,81
3500	8191	1028	3915	3502,21
4000	9337	1155	4423	4002,91
4500	10457	1279	4969	4498,26
5000	11689	1403	5589	4999,68
5500	12799	1526	6204	5502,49
6000	13921	1646	6804	5997,62
6500	15131	1767	7409	6501,33
7000	16229	1886	8004	7000,52
7500	17431	2005	8599	7503,48
8000	18523	2122	9184	8001,32
8500	19777	2239	9769	8502,46
9000	20939	2355	10349	9002,25
9500	22051	2471	10929	9504,8
10000	23167	2585	11499	10001,2