On Fast Search of First Confirmation Of Goldbach's Strong Conjecture

Marcin Barylski (marcin.a.barylski@gmail.com)

Published: May 8, 2016
The last update: December 29, 2019

Abstract

Goldbach strong conjecture states that all even integers $n>2$ can be expressed as the sum of two prime numbers (Goldbach partitions of n). Hypothesis still remains open and is confirmed experimentally for bigger and bigger n. This work studies different approaches to finding the first confirmation of this conjecture in order to select the most effective confirmation method.

1 Introduction

Goldbach strong (also called binary) conjecture asserts that all positive even integer $n \geq 4$ can be expressed as the sum of two prime numbers. This hypothesis, formulated by Goldbach in 1742 in letter to Euler [1] and then updated by Euler to the form above is one of the oldest and still unsolved problems in number theory. Empirical verification showed that it is true for all $n \leq 4$ x 10^{18} [2].

The expression of a given number n as a sum of two primes p_1 and p_2 is called a Goldbach Partition (GP) of n. Let's denote this relation as $GSC(n, p_1, p_2)$. Then Goldbach strong conjecture can be written as (1):

$$\forall_{x>1, x\in\mathbb{N}} \exists_{p_1,p_2\in\mathbb{P}} GSC(2x, p_1, p_2) \quad (1)$$

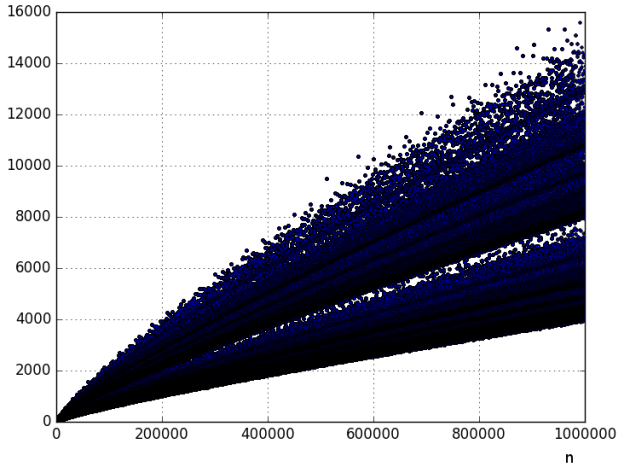

Figure 1: $r(n)$ ($2 < n < 10^6$, $n = 2k, k \in \mathbb{N}$)

Let $r(n)$ be the number of GPs of n and let $R(n)$ be a set of distincs GPs of n (uniqueness guaranteed through $p_1 \leq p_2$). Goldbach strong conjecture may be rewritten that $r(n) > 0$ for all positive even integers $n \geq 4$. Computational experiments show (for $n < 10^6$) that the hyphothesis may be reinforced: bottom estimation for $min(r(n))$ is increasing with n (Figure 1). [3] formulates conjecture that lower and upper bounds can be expressed as simple exponentials.

Empirical verification of Goldbach strong conjecture and search for GP requires fast and reliable primality test, repeated even twice for both components of a candidate for GP. The following paper is devoted to designing of the fastest algorithm for searching the first confirmation: a pair of primes: p_1 and p_2 for n, where $GSC(n, p_1, p_2)$. Design of algorithm is based on detailed observations for all GPs found for all even n ($4 \leq n < 10^6$), and then confirmed for bigger numbers. All listings are written in Python programming language and published at [4].

2 Fast primality test

For the sake of this work, to check if a given number is prime or not, algorithm skewed in Listing 1 was used. Presented approach is taking advantage of preloaded prime and composite sets (containing prime and composite numbers found earlier) which gives instant result. Then, algorithm is testing if a candidate for prime is even (divides it by 2) or is a multiple of 3 - in case of success the candidate is confirmed as a composite number. Eventually, it is taking advantage of Lemma 1.

Listing 1: Primality test

```
# input: integer n
# external dependencies:
#   prime_set - a set of primes
#   composite_set - a set of non-primes
# output: True if n is prime; False
    otherwise
def is_prime (n):
  if n<=1:
    return False
  elsif n<=3:
    return True
  elsif n in prime_set:
    return True
  elsif n in composite_set:
    return False
  elsif n%2 == 0 or n%3 == 0:
    return False
  i=5
  while i*i<=n:
    if n%i == 0 or n%(i+2) == 0:
      return False
    i+=6
  return True
```

Lemma 1. *Every prime $p>3$ can be written as $p = 6k \pm 1$ (where k is a positive integer).*

Proof. Every positive integer $n \geq 6$ can be expressed as $6k + m$, where m=0, 1, ..., 5, $k \geq$1. Numbers $6k$, $6k + 2$, $6k + 3$ and $6k + 4$ are always composite because they are divisible by either 2 or 3 or both ($6k$=$2 \times 3k$, $6k$+2=2×(3k+1), $6k$+3=3×(2k+1), $6k$+4=2×(3k+2)). $6k + 1$ and $6k + 5$ are either prime (ie. $6 \times 1 + 1 = 7$, $6 \times 2 - 1 = 11$) or composite (ie. $6k + 5$ is divisible by 5 if k is multiple of 5, $6k + 1$ is divisible by 3 if sum of decimal digits is divisible by 3). $6k+5$ can be rewritten as $6\times(k+1)-1$. All primes <5 (2 and 3) cannot be expressed as $p = 6k \pm 1$ where k is a positive integer. This means that every prime $p \geq 5$ can be expressed as $6k \pm 1$ (where $k \geq 1$). □

3 Characteristics of GPs

The first part of work is devoted to detailed examination of different characteristics of $R(n)$ (n <10^6) in order to find useful observations which are the foundation of further versions of GP fast confirmation algorithms.

3.1 Difference between primes in GP

First analysis concentrates on possible differences between primes in $R(n)$ (for n <10^6), including the smallest, average and the biggest difference.

Tests show that minimal difference may be low (comparing to n). In majority of examined cases it is below 1000, with just six recorded examples above 2000 (Figure 2). The bold observation may lead to a hypothesis that primes in at least one GP in $R(n)$ are not so far from each other (2):

$$\underset{k>1,k\in\mathbb{N}}{\forall} \underset{GSC(2k,p_1,p_2)}{\exists} \underset{C\in\mathbb{N}}{\exists} C = \mid p_1 - p_2 \mid \ll 2k \qquad (2)$$

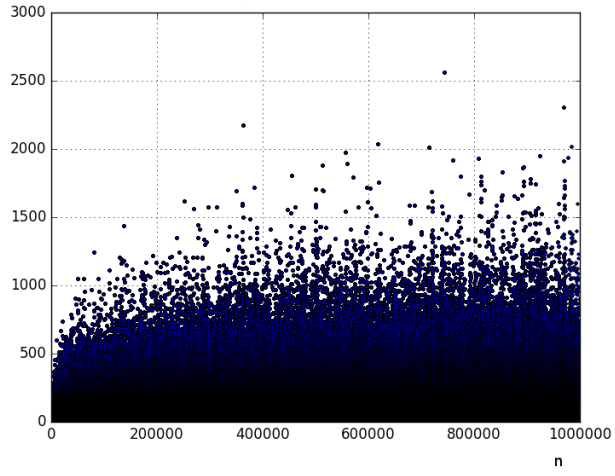

Figure 2: Minimal difference in $R(n)$ (2 <n <10^6)

Maximum difference in GP is close to n, indicating that p_2 in such extreme case ($p_1 \leq p_2$) is close to n (Figure 3). Average difference in GP is fluctuating slightly above $\frac{n}{2}$ (Figure 4). The bigger n, the more visible fluctuations.

Further observation of trends in change of minimum / maximum / average difference in GPs of n (+1 if difference between previous and current value is positive, -1 if negative, +0 if no change) shows that all those three

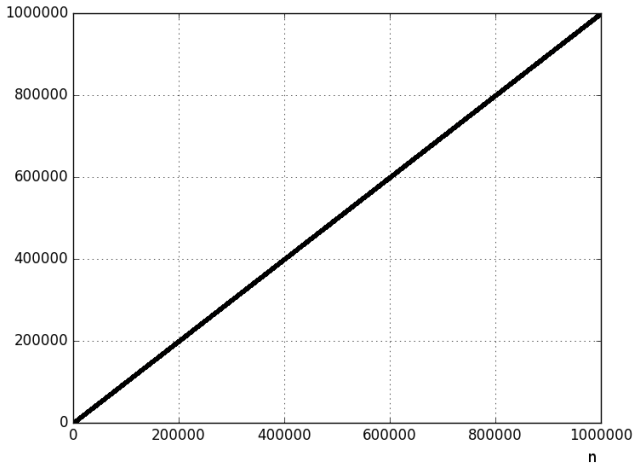

Figure 3: Maximum difference in $R(n)$ (2 <n <10^6)

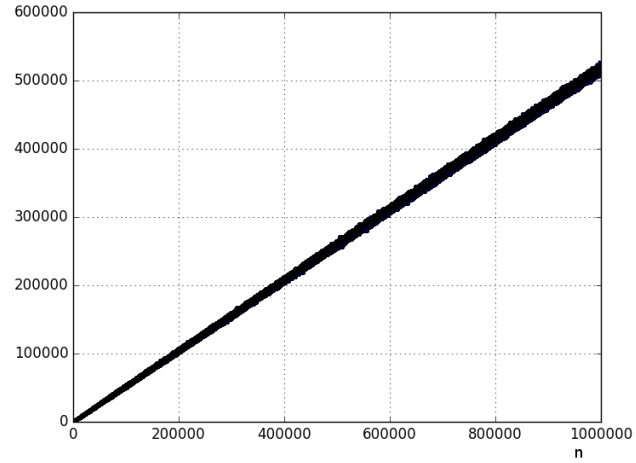

Figure 4: Average difference in $R(n)$ (2 <n <10^6)

trends are generally descending, with just few ascending episodes (Figure 5). Big picture (Figure 6) presents that both trends for minimum and average difference have very similar characteristics.

3.2 Minimal prime in GP

Detailed examination of $R(n)$ (n <10^6) shows that the minimal prime in at least one of GPs is usually low (Figure 7). For all n <10^6 it has been computionally verified that 523 is the biggest minimal prime (for n=503222) in all possible GPs. [2] verified that 3325581707333960528 is the smallest number that has no GP with a prime below 9781. Among the smallest primes in GBs (n <10^6) the most popular is 3 (Table 1).

3.3 Twin primes in GP

[5] shows that original Goldbach conjecture could be extended to a form that every even n>4 (this is (1) without case $n = 4 = 2+2$) can be expressed as a sum of twin prime and another prime (3):

$$\underset{x>2,x\in\mathbb{N}}{\forall} \underset{p_1\in\mathbb{P}_T}{\exists} \underset{p_2\in\mathbb{P}}{\exists} GSC(2x, p_1, p_2) \qquad (3)$$

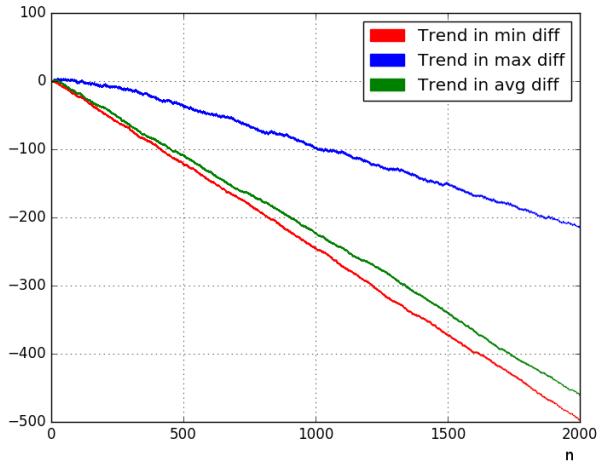

Figure 5: Trends in $R(n)$ for the smallest n ($2 < n < 2000$)

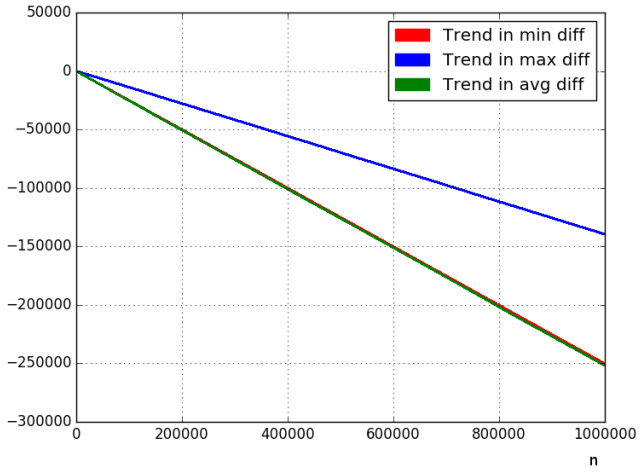

Figure 6: Trends in $R(n)$ ($2 < n < 10^6$)

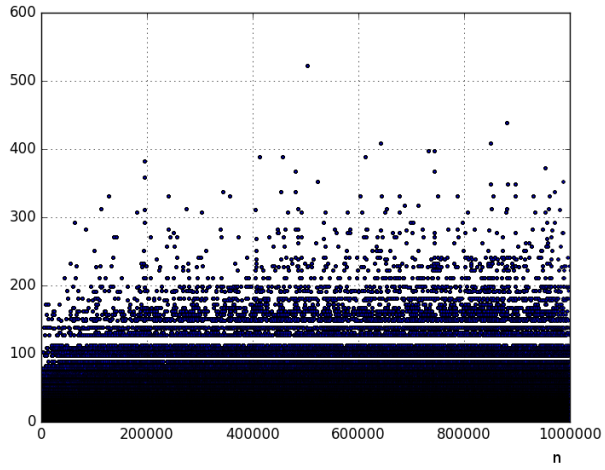

Figure 7: Minimal prime in $R(n)$ ($2 < n < 10^6$)

Figure 8 depicts that number of distint twin primes in $R(n)$ is increasing with n.

3.4 GP and symmetrical primes

We can rewrite Goldbach hypothesis to the following form: all positive integers >1 can be expressed as a half of sum of two primes. This means that given integer $n > 1$ is in fact a symmetry point for two primes p_1 and p_2. Let us call such p_1 and p_2 as symmetrical primes to n, denoting this symmetry as $psym(n, i) = (p_1, p_2)$, if there exists yet another integer i ($0 \leq i \leq n$ - 2) that $p_1 = n$ - i and $p_2 = n + i$. For example, $2 = (2 + 2)/2$ ($psym(2, 0) = (2, 2)$), $3 = (3 + 3)/2$ ($psym(3, 0) = (3, 3)$), $4 = (3 + 5)/2$ ($psym(4, 1) = (3, 5)$). If n is prime, then we always have $psym(n, 0) = (n, n)$.

Let $s(n)$ be a number of available symmetrical primes to n, and $S(n)$ a set of all symmetrical primes to n. Figure 9 which is depicting $s(n)$ has very similar shape to Figure 1 which is depicting $r(n)$. Further empirical examination suggests that $s(n)$ and $r(n)$ are correlated what is depicted by Figure 12. Each GP is built from two primes p_1 and p_2 which are symmetrical primes to $n = \frac{p_1+p_2}{2}$. n is always a positive integer because every GP is built from either both odd primes or both even primes, thus sum of such two primes is always even. Each GP is constructed from two primes which are fulfilling $psym(\frac{p_1+p_2}{2}, \frac{p_2-p_1}{2})$.

Table 1: Smallest primes in $R(n)$ ($2 < n < 10^6$)

Prime	Appearances	Prime	Appearances
2	1	79	101
3	78497	181	219
5	70328	191	76
7	62185	193	109
11	48582	197	49
13	40916	199	112
17	31091	211	97
19	29791	223	40
23	21422	227	37
29	16776	229	42
31	18119	233	32
37	13165	239	25
41	10001	241	41
43	9100	251	19
47	6625	257	12
53	5076	263	9
59	4012	269	3
61	6417	271	22
67	4839	277	15
71	2597	281	4
73	2801	283	17
79	3030	293	8
83	1753	307	14
89	1442	311	3
97	1763	313	7
101	988	317	2
103	1266	331	12
107	889	337	4
109	1245	349	3
113	507	353	2
127	730	359	1
131	356	367	2
137	358	373	1
139	602	383	1
149	279	389	3
151	522	397	2
157	253	409	2
163	258	439	1
167	168	523	1

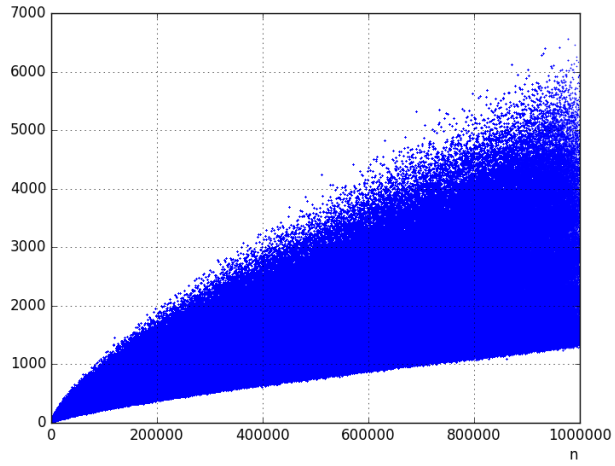

Figure 8: Number of distinct twin primes in $R(n)$ ($2 < n < 10^6$)

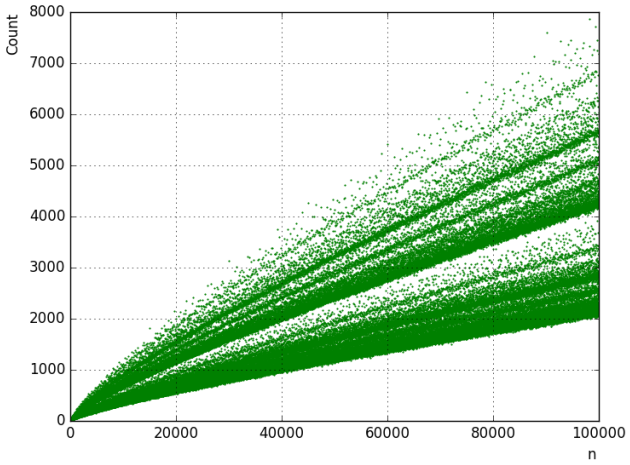

Figure 9: $s(n)$ ($2 < n < 10^5$)

Lemma 2. *Every prime $p > 3$ can be written as $p = 3x \pm 1$ (where x is a positive integer).*

Proof. All integers $n>3$ can be written as $3x + i$, where $i = 0, 1, 2$ and $x \geq 1$. If $n_1 = 3x>3$, then n_1 is always composite number, because $x>1$ and both 3 and x divides n_1. On the other hand, numbers of form $3x-1$ can be either prime (ie. $5 = 3 \times 2 - 1$) or composite (ie. $8 = 3 \times 3 - 1 = 2 \times 2 \times 2$) and numbers of form $3x+1$ can be either prime (ie. $7 = 3 \times 2 + 1$) or composite (ie. $10 = 3 \times 3 + 1 = 2 \times 5$). □

Figure 10 shows that minimal i in $psym(n, i)$ is relatively low (in comparison to n), similarly to smallest primes in $R(n)$, depicted by Figure 7. Maximum i is close to n (Figure 11).

Lemma 3. *If integer n is a half of the sum of two primes then n can be expressed as either $3x - 1$ or $3x$ or $3x + 1$, where x is integer ≥ 1.*

Proof. Based on Lemma 1, every prime $p > 3$ is a form of $6k \pm 1$ (where $k \geq 1$). Sum of two primes s_i is then in 3 variants (k_i is integer ≥ 1, $k_3 = k_1 + k_2$): $s_1 = 6k_1 - 1 + 6k_2 - 1 = 6k_3 - 2$; $s_2 = 6k_1 - 1 + 6k_2 + 1 = 6k_3$; $s_3 = 6k_1 + 1 + 6k_2 + 1 = 6k_3 + 2$. We will have then: $\frac{s_1}{2} = 3k_3 - 1$; $\frac{s_2}{2} = 3k_3$; $\frac{s_3}{2} = 3k_3 + 1$. Lemma is then true if both primes are >3. Let us then analyze all missing cases with primes ≤ 3: 2 and 3. If one of the primes is 2, then sum of the primes can only be even (so that divided by 2 gives integer) if second prime is also 2. This gives us: $\frac{2+2}{2} = 2 = 3 \times 1 - 1$, which fulfills the lemma. If one of the primes is 3, then the

Table 2: Symmetrical primes to $n>1$ ($a, x, k_1, k_2 > 0$)

n	$\frac{p_1+p_2}{2}$	(p_1, p_2)
2	$3 \times 1 - 1$	$(2, 2)$
3	3×1	$(3, 3)$
4	$3 \times 1 + 1$	$(3, 5)$
5	$3 \times 2 - 1$	$(5, 5)$, $(3, 7)$
6a	3x	$(3k_1 - 1, 3k_2 + 1), (3k_1 + 1, 3k_2 - 1)$ 3x is not prime, $2 \mid k_1 + k_2$
6a + 1	3x + 1	$(3k_1 + 1, 3k_2 + 1), (3, 2n - 3)$ $2 \mid k_1 + k_2$
6a + 2	3x - 1	$(3k_1 - 1, 3k_2 - 1), (3, 2n - 3)$ $2 \mid k_1 + k_2$
6a + 3	3x	$(3k_1 - 1, 3k_2 + 1), (3k_1 + 1, 3k_2 - 1)$ 3x is not prime, $2 \mid k_1 + k_2$
6a + 4	3x + 1	$(3k_1 + 1, 3k_2 + 1), (3, 2n - 3)$ $2 \mid k_1 + k_2$
6a + 5	3x - 1	$(3k_1 - 1, 3k_2 - 1), (3, 2n - 3)$ $2 \mid k_1 + k_2$

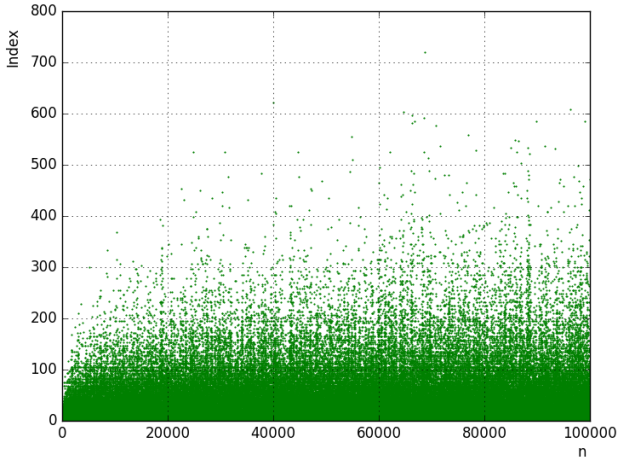

Figure 10: Minimal i in $psym(n, i)$ ($n < 10^5$)

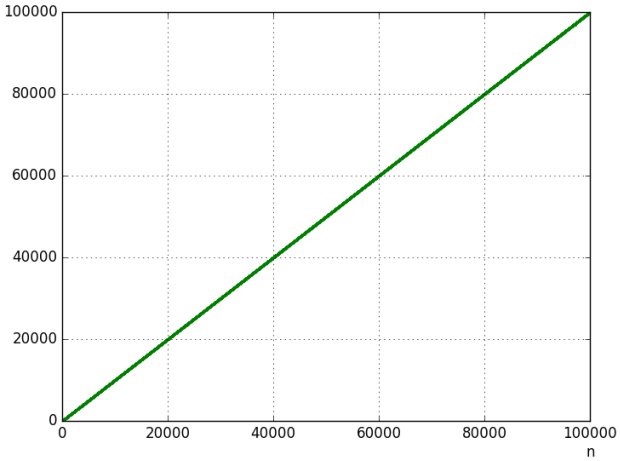

Figure 11: Maximum i in $psym(n, i)$ ($n < 10^5$)

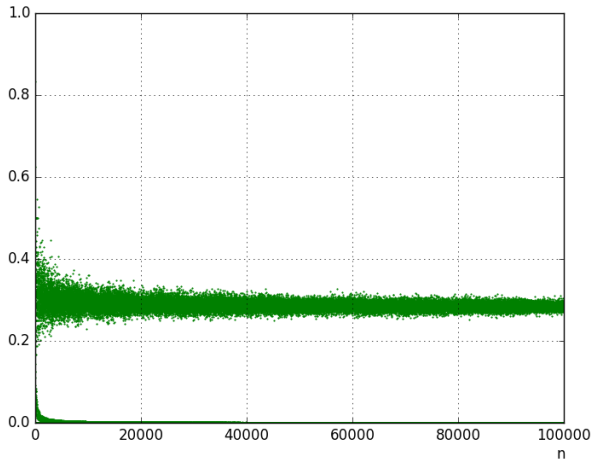

Figure 12: $r(n)$ / $s(n)$ (n <10^5)

second prime cannot be 2 and can be either 3 or prime >3. If second prime is 3, then we have $\frac{3+3}{2} = 3 = 3 \times 1$, which fullfills the lemma. If second prime is >3, then we have the following variants: $s_4 = 3 + 6k_2 - 1 = 6k_2 + 2$; $s_5 = 3 + 6k_2 + 1 = 6k_2 + 4 = 6(k_2 + 1) - 2$. We have then: $\frac{s_4}{2} = 3k_2 + 1$; $\frac{s_5}{2} = 3(k_2 + 1) - 1$, which also fullfills the lemma. □

Every positive integer n>1 can be expressed as $6a + m$, where $m = 0, 1, \ldots, 5$ and $a \geq 0$. Let us analyze all possible solutions in integer numbers if we confront this with Lemma 3 - we will have six (from a) to f)) cases then:

a)
$$6a = \begin{cases} 3x - 1 & \text{if } x = 2a + \frac{1}{3} \\ 3x & \text{if } x = 2a \\ 3x + 1 & \text{if } x = 2a - \frac{1}{3} \end{cases}$$

b)
$$6a + 1 = \begin{cases} 3x - 1 & \text{if } x = 2a + \frac{2}{3} \\ 3x & \text{if } x = 2a + \frac{1}{3} \\ 3x + 1 & \text{if } x = 2a \end{cases}$$

c)
$$6a + 2 = \begin{cases} 3x - 1 & \text{if } x = 2a + 1 \\ 3x & \text{if } x = 2a + \frac{2}{3} \\ 3x + 1 & \text{if } x = 2a + \frac{1}{3} \end{cases}$$

d)
$$6a + 3 = \begin{cases} 3x - 1 & \text{if } x = 2a + \frac{4}{3} \\ 3x & \text{if } x = 2a + 1 \\ 3x + 1 & \text{if } x = 2a + \frac{2}{3} \end{cases}$$

e)
$$6a + 4 = \begin{cases} 3x - 1 & \text{if } x = 2a + \frac{5}{3} \\ 3x & \text{if } x = 2a + \frac{4}{3} \\ 3x + 1 & \text{if } x = 2a + 1 \end{cases}$$

f)
$$6a + 5 = \begin{cases} 3x - 1 & \text{if } x = 2a + 2 \\ 3x & \text{if } x = 2a + \frac{5}{3} \\ 3x + 1 & \text{if } x = 2a + \frac{4}{3} \end{cases}$$

In each case just one subcase has solution where both x and a are integers. Table 2 presents these solutions, suplemented by candidates for prime pairs which can produce a given integer symmetry point. Table 2 is taking advantage of Lemma 2 for numbers ≥ 6 and contains manual calculations for exact symmetrical primes if $2 \leq n \leq 5$. Lemma 2 is also useful when symmetry point n is of form $3x$ and n>3 - in such case $psym(n, 0)$ does not exist because n cannot be prime. Additionaly, $2 \mid k_1 + k_2$ in each row of Table 2, otherwise $\frac{p_1+p_2}{2}$ would not be integer. If $n = 6a$ or $n = 6a + 3$ ($a \geq 1$) (both numbers are of form $3x$), then $psym(n, n - 3)$ does not exist because of Lemma 4.

Lemma 4. *If $n = 3a$ (a is integer >1), then 3 cannot be a symmetric prime to n.*

Proof. If $p_1 = 3$ is going to be symmetric prime to n, then the second prime in symmetry pair is $p_2 = 2n - 3$. If $n = 3a$, then $p_2 = 2 \times 3a - 3 = 3 \times (2a - 1)$. If $2a - 1$>1, then p_2 is a composite number (divisors: 3 and $2a - 1$). $2a - 1$ is always >1 if a is integer >1. □

4 GP confirmation algorithm

The gist of this work studies various approaches to GP confirmation for consecutive even numbers. First class of methods (Class A) is representing top-down approach which is expressing a given even number n >2 as a possible sum of two components p_1 and p_2 first, and then checks their primality. Second method class (Class B), a member of bottom-up solutions, is iterating over possible sums built from two numbers p_1 and p_2, both confirmed as primes in advance. Third method (Class C) is looking for the symmetrical primes p_1 and p_2 around n >1.

4.1 Class A: primality test of possible components

Listing 4 depicts algorithm base A used to find the first GP confirmation using primality test of one or two components, p_1 and p_2, which sum is producing odd number n subjected to GP check.

There are two input parameters in the presented approach: starting point for the first round of GP check (initial values of p_1 and p_2) and next step details to calculate new values of p_1 and p_2 if the previous interation failed. Starting point could be also expressed as p_1/p_2 ratio. Next step value could be either constant or variable. As a result algorithm returns GP details (values of p_1 and p_2, both primes). In addition, in order to compare different versions of algorithms, it returns both number of iterations (denoted as $I(A)$) and total duration of internal calculations. Algorithm throws exception if no GP is found and there is no further iteration possible (next candidate for either p_1 or p_2 is smaller than the smallest prime 2).

It is reasonable to set initial p_1 and p_2 values as odd numbers (there is just one GP where any of GP factors could be even: $4 = 2 + 2$), otherwise first iteration (excluding GP for 4) would always fail.

If p_1 and p_2 are odd, then the next step value (calculated in case of fail) should be a number that will not produce even number as the next candidate for neither p_1 nor p_2 - it should be a mutliple of 2. There is also a risk that in case of too big next step value algorithm loop would go to the end without finding a candidate in between. Although we are still empirically confirming GP correctness (in other words: there might be even n for which GP is not possible) but if we miss any possible candidate in between going back could be a troublesome. Having that in mind, next step value = 2 looks reasonable - we are moving slowly, one by

Table 3: Summary of algorithm A variants

Variant	Initial p_1 and p_2	Delta
A_1	$p_1 = n/2$ $p_2 = n$ - p_1 if p_1 % 2 == 0: p_1 -= 1 p_2 += 1	-2
A_2	$p_1 = 3$ $p_2 = n$ - p_1	+2
A_3	$p_1 = \text{int}(n/3)$ $p_2 = n$ - p_1 if p_1 % 2 == 0: p_1 -= 1 p_2 += 1	-2
A_4	$p_1 = 5$ $p_2 = n$ - p_1	+2
A_5	$p_1 = 5$ $p_2 = n$ - p_1	variable = 0 for iter 0 = -2 for iter 1 = +4 for iter 2 = +2 for iter >2
A_6	$p_1 = 3$ $p_2 = n$ - p_1	variable p_1 = next prime
A_7	$p_1 = 3$ $p_2 = n$ - p_1	variable p_1 = next twin prime

one through odd numbers - we will not miss any possible prime number.

It is also important to mention that all parallel runs take advantage of the same prime and composite number sets. If a given number p exists in either prime set or composite set, primality test gives instant result. This means that only the first run for number p which does not exist in neither prime set nor composite set pays price for full primality test, influencing significantly on time of the processing.

Seven variants of algorithm A are being considered, all based on sieve which is testing primality of each element of a candidate for GP (Table 4). Variants differ with initial values of $p_1 + p_2$ and delta used to calculate next candidates. All variants assume that $p_1 \leq p_2$ and both numbers are always odd (n >4 to exclude 4 = 2+2 case). In case of primality test failure for a given pair p_1 and p_2, the next pair of candidates is changed in regular (A_1, A_2, A_3, A_4 - p_1 is increased by delta and p_2 is decreased by delta) or variable (A_5, A_6, A_7 - delta is a function of iteration) way (Listing 2, Listing 3). Programatically it was possible to keep the same source code for all variants, including *delta*() (which is a function of iteration) passed to the function as a lambda expression.

Listing 2: Constant delta

```
def delta_constant_plus (i):
  return 2

def delta_constant_minus (i):
  return -2
```

The first variant, A_1, based on observation depicted in Figure 2, is assuming that difference between primes in GP is small (in comparison to n). Initial values of p_1 and p_2 will be the first matching odd numbers around $\frac{n}{2}$.

The second variant, A_2, based on observation depicted in Figure 7, is assuming that one of the primes in GP is small (in comparison to n). p_1 starts from 3 but not 2 because n - 2 would never be prime for n >4.

The third variant, A_3, is a proposal in between A_1 and A_2, with starting point about one third of n.

The fourth variant, A_4, is identical to A_2 except starting point: $p_1 = 5$, p_2 = n - 5.

The fifth variant, A_5, is more flexible than A_4. p_1 also starts from 5 but it checks $p_1 = 3$, and then $p_1 = 7$ and next odd numbers.

The sixth variant, A_6, is identical to A_2 except that next step lenght is variable. p_1 is always prime (for i^{th} iteration it is i^{th} prime number). [2] calculated that 9781 (1206^{th} prime) is the biggest known prime (so far) required in such approach. *delta_prime*() starts from 3 because all numbers n subjected to Goldbach partitioning are greater than 4, so they will not have 2 in their $R(n)$.

The seventh variant, A_7, is a mutation of A_6 - p_1 is always a twin prime. *delta_twinprime*() starts from 3 because this is the first twin prime.

Listing 3: Variable delta

```
# input: iteration
# output: delta for next iteration
def delta_variable (i):
  if i == 0:
    delta=0
  elif i == 1:
    delta=2
  elif i == 2:
    delta=-4
  elif i > 2:
    delta=2
  return delta

# input: iteration
# output: delta for next iteration
def delta_prime (i):
  if i == 0:
    delta=0
  else:
    delta=get_ith_prime(i+1)-
        get_ith_prime(i)
  return delta

# input: iteration
# output: delta for next iteration
def delta_twinprime (i):
  if i == 0:
    delta=0
  else:
    delta=get_ith_twinprime(i)-
        get_ith_twinprime(i-1)
  return delta

# primes[] = [2, 3, 5, 7 ...]
# twin_primes[] = [3, 5, 7, 11 ...]

# input: index of prime
# output: prime
function get_ith_prime (i):
   return primes[i]

# input: index of twin prime
```

```
# output: twin prime
function get_ith_twinprime (i):
    return twin_primes[i]
```

Listing 4: Fast search algorithm scheme A

```
# input: initial p1 and p2 candidates
#     (p1+p2=n)
#     delta - function to calculate
#         next p1 and p2 candidates
# output: final p1 and p2 pair
#     (both primes, p1+p2=n),
#     time elapsed to find this pair
#     number of iterations required
# exception:
#   when no partition is found
def search_for_partition (p1, p2, delta):
  found=False
  iteration=0

  startTime=time.time()
  while not found:
    iteration+=1
    if is_prime(p1) and is_prime(p2):
      found=True
    if not found:
      p1-=delta(iteration)
      p2+=delta(iteration)
    if p1 < 2 or p2 < 2:
      raise ("GPnotFound")
  duration=time.time()-startTime

  return p1, p2, duration, iteration
```

4.2 Class B: sum built from primes

Listing 5 depicts algorithm base B used to find GP confirmation using two primes sum building approach.

Listing 5: Fast search algorithm scheme B

```
# input: min and max indices of first
    prime
# external dependencies:
#   S1 - a set of numbers to be verified
#   S2 - a set of numbers already
    verified
#   N - number below which all even
    numbers were already verified
#   add_nums() - updates S1
#   remove_nums() - updates S1, S2, N
# output: S1, S2, N
function check_possible_sums (min_ip1,
    max_ip1):
  for ip1 in range (min_ip1, max_ip1):
    p1 = get_ith_prime(ip1)
    add_nums (2*p1)

    for ip2 in range (1, ip1+1):
      p2 = get_ith_prime(ip2)
      num = p1 + p2
      remove_nums  (num, 2*p1)
```

In contradiction to A in approach B primality test in direct loop is not required because both components, p_1 and p_2, are already prime numbers. B is iterating over possible pairs (starting from prime 3, prime number 2 is excluded from calculations because there is only one even number 4 with 2 in its GP) and collects information about possible sums (which are always even because both primes are odd). Let us define an even number N_{conf} below which all even numbers were already confirmed from GP standpoint. It is highly probable that N_{conf} would grow up along with progress of the algorithm because $r(n)$ grows with bigger n (Figure 1). There is no point in doing any calculations for any $p_1 + p_2 \leq N_{conf}$ because this part was already confirmed. Assuming that for a given p_1 we iterate down over p_2 from p_1 to p_m (where $p_1 + p_m > N_{conf}$) the most favourable situation would be if after completing all checks for p_1 and p_2 we have $N = 2 \times p_1$ - next iteration would not inherit any backlog.

4.3 Class C: looking for symmetric primes

Listing 6 presents algorithm base C used to find GP confirmation for even n by finding symmetrical primes to $\frac{n}{2}$. Like in approach A, C requires primality test inside the algorithm loop. C has two input parameters: symmetry point and function *delta*() used to calculate next pair of prime candidates (p_1, p_2) if both current values are not primes yet.

Table 4: Summary of algorithm C variants

Variant	Initial value of i	Delta i
C_1	0	1
C_2	0	1 if n is even 2 if n is odd

Listing 6: Fast search algorithm scheme C

```
# input: n
#     n = (p1 + p2) / 2
#     delta - function to calculate
#         next p1 and p2 candidates
# output: final p1 and p2 pair
#     (both primes, p1+p2=n),
#     time elapsed to find this pair
#     number of iterations required
# exception:
#   when no symmetrical primes were found
def search_for_sym_primes (n, delta):
  found=False
  iteration=0

  startTime=time.time()
  p1=n
  p2=n
  while not found:
    iteration+=1
    if is_prime(p1) and is_prime(p2):
      found=True
    if not found:
      p1-=delta(iteration)
      p2+=delta(iteration)
    if p2 < 2 or p1 < 2:
      raise Exception ("SymPrNotFound")
```

```
    duration=time.time()-startTime

    return p1, p2, duration, iteration
```

5 Results of experiments

5.1 Experiments against class A

All seven variants of class A of GP search algorithm were subjected to programmatic verification against all even numbers $14 < n < 4 \times 10^8$. Calculations show that A_2 required 5 times less iterations than both A_1 and A_3 (Figure 13). Results for A_2 and A_5 are comparable - number of iterations look almost identical (Figure 14 - line for A_2 covers line for A_5) while A_4 is worse than A_2 and A_5. The best results achieved A_6. In comparison to A_2 (Figure 15) this approach is using preloaded list of first primes and thanks to that is able to avoid primality checks for p_1 (p_1 is always prime), algorithm variant does not have to check a case when both candidates are not primes - only primality test of p_2 matters. A_7 was slightly worse than A_6.

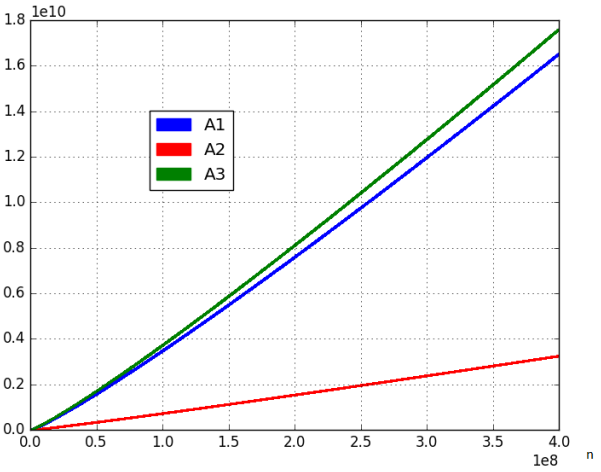

Figure 13: Iterations - A_1 vs A_2 vs A_3 ($n < 4 \times 10^8$), A_2 is the best one: I(A_3) >I(A_1) >I(A_2)

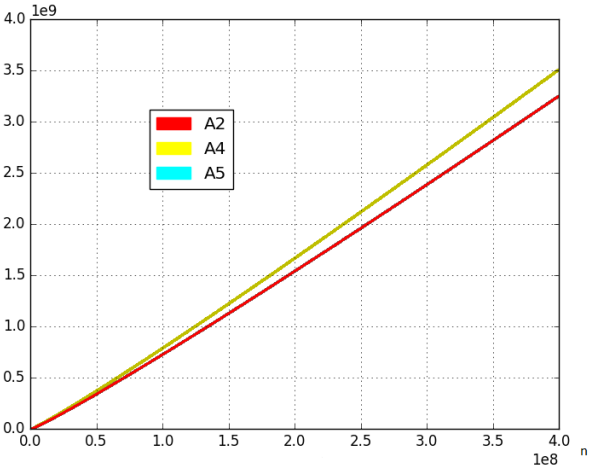

Figure 14: Iterations - A_2 vs A_4 vs A_5 ($n < 4 \times 10^8$), A_2 is the best one: I(A_4) >I(A_5) $\geq$ I(A_2)

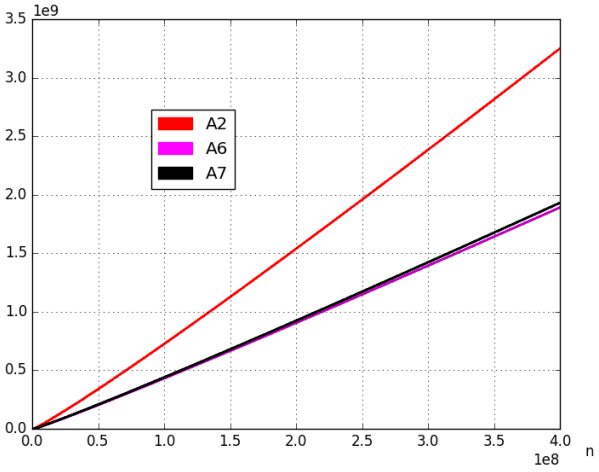

Figure 15: Iterations - A_2 vs A_6 vs A_7 ($n < 4 \times 10^8$), A_6 is the best one: I(A_2) >I(A_7) >I(A_6)

Results achieved by A_6 suggest that properties depicted by Figure 3 and Figure 7 are strong and still preserved for $n > 10^6$. Assuming that efficiency of primality test can be improved, A_6 looks like the best choice amongst all examined As. A_6 requires prework - a list of first primes for p_1 - but such preloaded set is not a big issue for modern computers and may be reused in primality test.

5.2 Experiments against class B

Effectiveness of sum building from prime numbers is very high. Each round of algorithm B has a theoretical maximum even number N_{max} below which all numbers are already verified: $N_{max} = max(p_1, p_2) \times 2$. Detailed examination of difference between N_{max} and N_{conf} show that this difference is low: both lines in Figure 16 almost match (although $N_{max} = N_{conf}$ for iterations 0, 1, 2, 4, 6 and 27 only, for other cases calculated so far $N_{max} > N_{conf}$), red values in Figure 17 are relatively low. Cardinalities of two supporting sets, a set of number to be verified $> N_{conf}$ and a set of spare verified numbers $> N_{conf}$, are also low (Figure 18).

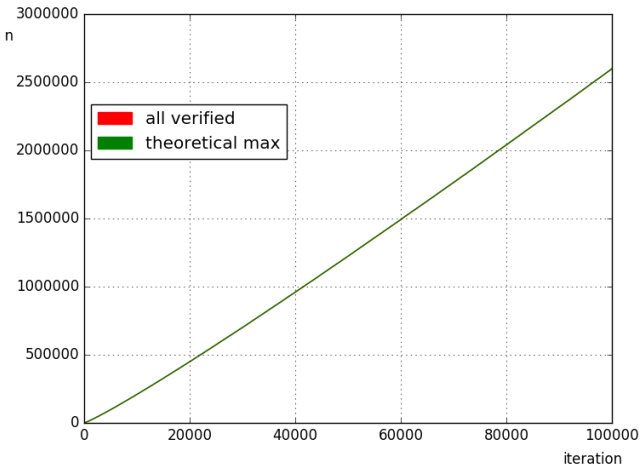

Figure 16: Effectiveness of sum building in B

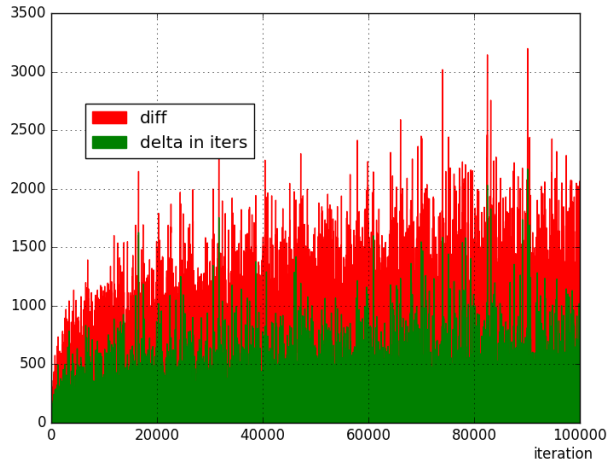

Figure 17: Differences in B: difference between actual N_{conf} and theoretical maximum N_{max} (in red); absolute difference of red values between individual iterations (in green)

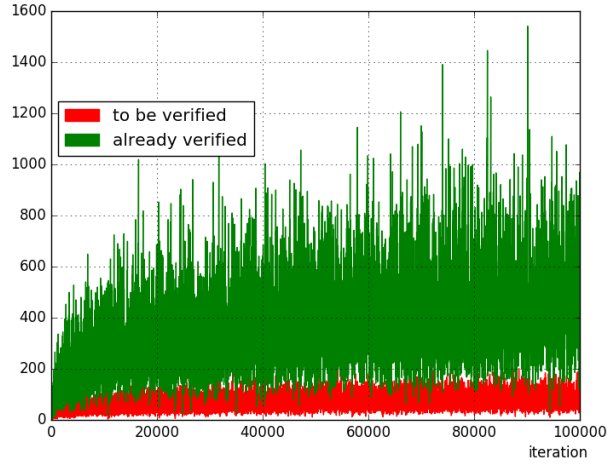

Figure 18: Cardinality of two supporting sets in B: numbers to be verified $>N_{conf}$ (in red) and spare numbers already verified $>N_{conf}$ (in green)

5.3 Experiments against class C

Executed experiments show that the most effective algorithm in class C is C_2.

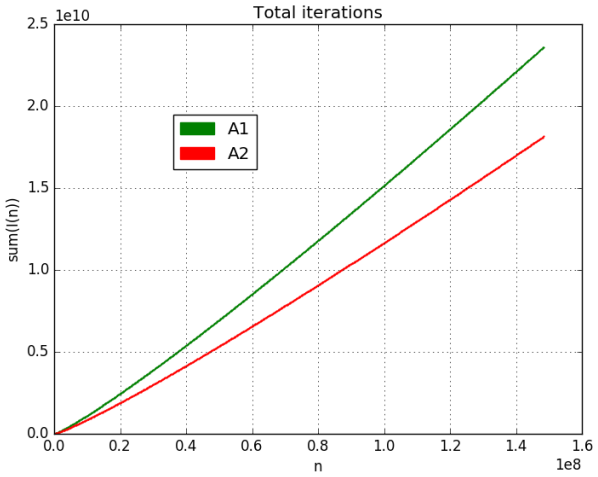

Figure 19: Iterations - C_1 vs C_2 ($n < 4 \times 10^8$), C_2 is the best one: I(C_1) >I(C_2)

6 Summary and future work

From all examined algorithm variants of search for the first pair of primes (p_1, p_2) for even n that $GSC(n, p_1, p_2)$ the most promissing results were achieved by A_6. When taking into account finding such partition for all consecutive even numbers $n_1, n_2, \ldots n_m$ A_6 allowed to find this partition with the least number of iterations. Listing 7 presents skew of distributed version of A_6-like approach - initial data range is divided into subintervals and each subinterval is verified independently on a separate compute node.

Listing 7: Distributed fast search

```
def verify (chunk):
    for n in chunk:
        p1 = 3
        p2 = n - 3
        (p1, p2, d, i) =
            search_for_partition (p1, p2,
            lambda i: delta_prime(i))

# main
chunks = divide (max, max_chunk_size)
for chunk in chunks:
    verify (chunk)
```

Furthermore, studies on GP fast confirmation methods resulted in additional interesting research areas and questions which could be a foundation of further research work.

First area relates to frequency of primes in GPs. 2 is the prime number that exists in one GP only ($4 = 2 + 2$) - let's call such prime a selfish prime - and no other prime of such property exists. But how about other most and least frequent primes? Is there any pattern?

Second research field may concentrate on finding further mathematical patterns of regular properties observed during above studies. Examples are: bottom and upper estimations of maximum and average difference between primes in GP.

References

[1] Christian Goldbach, *On the margin of a letter to Leonard Euler*, 1742.

[2] Tomás Oliveira e Silva, *Goldbach conjecture verification.* http://sweet.ua.pt/tos/goldbach.html, 2012.

[3] Max S.C. Woon, *On Partitions of Goldbach's Conjecture*, arXiv:math/0010027 [math.GM], 2000.

[4] Marcin Barylski, *Goldbach conjecture verification framework.* https://github.com/mbarylsk/goldbach-partition, 2016.

[5] Marcin Barylski, *Studies on Twin Primes in Goldbach Partitions of Even Numbers*, 2018.

[6] Marcin Barylski, *Goldbach Strong Conjecture Verification Using Prime Numbers*, 2018.

[7] Tomás Oliveira e Silva, *Fast implementation of the segmented sieve of Eratosthenes.* http://sweet.ua.pt/tos/software/prime_sieve.html, 2002.