A memory-free deterministic method for the generation of the n-th prime number: a Desmos implementation.

ABSTRACT:

An alternative method for computing deterministically the nth prime number is presented, based on counting composite numbers through a purely basic algebraic operations. The approach avoids lists, irrational constants, estimates, approximations, and/or precomputations—though these could optionally be incorporated to facilitate calculations. Despite lacking a formal theoretical proof, empirical results demonstrate its validity for every tested input beyond the 2nd prime number, as shown and verifiable in the provided Desmos implementation.

Some reference calculation times:

n.prime	time
10.000°	<1s
50.000°	~2s
100.000°	~5s
500.000°	~60s
1.000.000°	~180s

INTRODUCTION:

Calculating prime numbers efficiently is a classic problem in mathematics. Most methods rely on storing large lists of numbers or using approximations, which are impractical in environments like Desmos, where:

Memory is limited: No support for dynamic lists or large arrays.

Precision is constrained: Only 15-digit decimal precision, making irrational numbers (like π) unreliable for exact calculations.

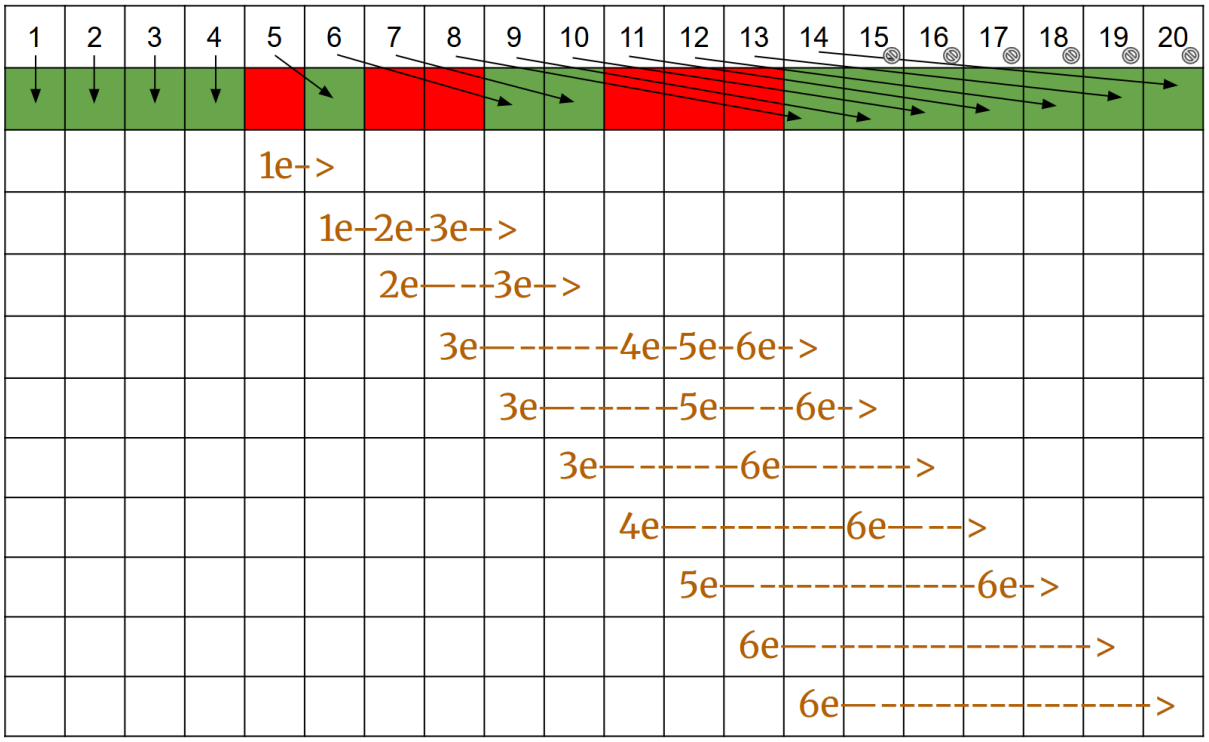
Performance is slow: Intensive calculations can freeze the interface.

METHOD:

1) MULTIPLES DISTRIBUTION AND THEIR COUNTING.

To determine the nth prime number, it is necessary to know the quantity of preceding prime or composite numbers.

Indeed, by using a specific recursive calculation I named the 'correction algorithm,' the image reveals green cells (primes) and red cells (composites). For any chosen number *n*, if you add to it the count of red cells (composites) up to *n*, you obtain a new number. By iteratively adding to the original number the count (e (errors)) of red cells preceding each newly found number—and repeating this process until the count of red cells stabilizes—you arrive at the *n*-th prime number.



To determine the error rate in prime numbers and—most importantly—to simplify calculations, I needed a formula that would return all prime numbers with the fewest errors. This led me to the 6x±1 formula, which I linearized using the function a(x).

•If x is even, the function can be written as 3x + 1.

•If x is odd, the function can be written as 3x + 2.

•Each x corresponds to the x-th number in the numerical sequence generated by 6x±1.

$$a(x) = 3x + \frac{3 - (-1)^x}{2}$$

The formula yields exactly four candidate numbers per decimal decade (10x, 10(x+1)) that can be prime, corresponding to those ending with 1, 3, 7, or 9 in their unit digit.

While studying the sequence generated by function a(x), I observed that multiples (i.e., 'non-prime numbers') followed a precise pattern, which I was able to translate into a mathematical formula (D(x)). However, although this formula work in practice, I still lack a formal proof explaining why this is the case.

$$D(x) = x + -\operatorname{round}\left(\frac{x}{6}\right)$$

Every element in a(x) possesses a dual periodicity governing the x value for a(x) of its multiples. The cases of 5, 7, and 11 exemplify this property:

a(0<x<33)

5 7 11 13 17 19 23 25 29 31 35 37 41 43 47 49 53
55 59 61 65 67 71 73 77 79 83 85 89 91 95 97

x-index for a(x) of multiples of a(1) = 5

D(5+) = 7
D(5-) = 3

1+7	8	a(8) = 25
+3	11	a(11) = 35
+7	18	a(18) = 55
+3	21	a(21) = 65
+7	28	a(28) = 85
+3	31	a(31) = 95
…	…	…

a(0<x<46)

5 7 11 13 17 19 23 25 29 31 35 37 41 43 47 49 53
55 59 61 65 67 71 73 77 79 83 85 89 91 95 97 101
103 107 109 113 115 119 121 125 127 131 133 137

x-index for a(x) of multiples of a(2) = 7

D(7+) = 9
D(7-) = 5

2+9	11	a(11) = 35
+5	16	a(16) = 49
+9	25	a(25) = 77
+5	30	a(30) = 91
+9	39	a(39) = 119
+5	44	a(44) = 133
…	…	…

a(0<x<70)

5 7 11 13 17 19 23 25 29 31 35 37 41 43 47 49 53 55
59 61 65 67 71 73 77 79 83 85 89 91 95 97 101 103
107 109 113 115 119 121 125 127 131 133 137 139
143 145 149 151 155 157 161 163 167 169 173 175
179 181 185 187 191 193 197 199 203 205 209 211

x-index for a(x) of multiples of a(3) = 11

D(11+) = 15
D(11-) = 7

3+15	18	a(18) = 55
+7	25	a(25) = 77
+15	40	a(40) = 121
+7	47	a(47) = 143
+15	62	a(62) = 187
+7	69	a(69) = 209
…	…	…

However, as can be observed, some numbers in a(x) possess more than one prime factor—for example, 35 (a multiple of both 5 and 7) or 77 (a multiple of 7 and 11). This prevented an absolute count of composites in the chosen interval, since simply summing the multiples of each number would lead to double-counting shared multiples. To resolve this, I attempted to reformulate these dual periodicities of each number using sinusoidal functions. By multiplying these sinusoids with those of all other prime factors, I derived a composite sinusoidal wave whose zeros corresponded to integer x-values that would generate composite numbers in a(x).

Naturally, since the numbers in a(x) are not all prime, the sinusoids aren't always useful for composite counting. For instance, 25—though not prime—will still generate its own sinusoid. This is redundant because its multiples are already accounted for by its prime factor 5.

But it's likely better to keep redundant sinusoids than to add overly complicated filtering formulas—at least in Desmos.

2) THE UNSUCCESSFUL SINUSOIDAL APPROACH.

By using formula D(x) to understand how to construct the general function that would allow me to derive the sinusoids for each number in a(x), I managed to express the final sinusoid (generated by multiplying all the sinusoids of the chosen numbers) with this formula:

$$\prod_{k=1}^{c}\left(\sin\left(\left(\frac{\pi}{2a(k)}\right)(x+1+k)\right)\right)\sin\left(\left(\frac{\pi}{2a(k)}\right)\cdot\left(x+1+a(k)+2\,\mathrm{round}\left(\frac{a(k)+1}{6}\right)+k\right)\right)$$

where the formula c(x) returns the index of the largest number in a(x) that is less than or equal to c's x-value.

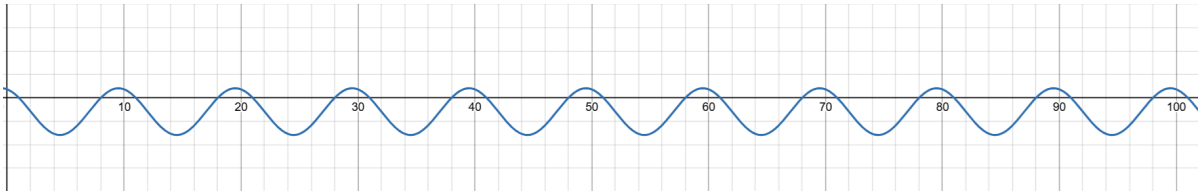
$$c(x)=\mathrm{round}\left(\frac{x}{6}\right)-\mathrm{round}\left(\frac{x}{6}-\mathrm{floor}\left(\frac{x}{6}\right)\right)+\mathrm{ceil}\left(\frac{x}{6}-\mathrm{floor}\left(\frac{x}{6}\right)\right)+\mathrm{ceil}\left(\frac{5x-21}{30}\right)-1$$

Indeed, to determine the minimum number of sinusoids required to detect all composite numbers, I

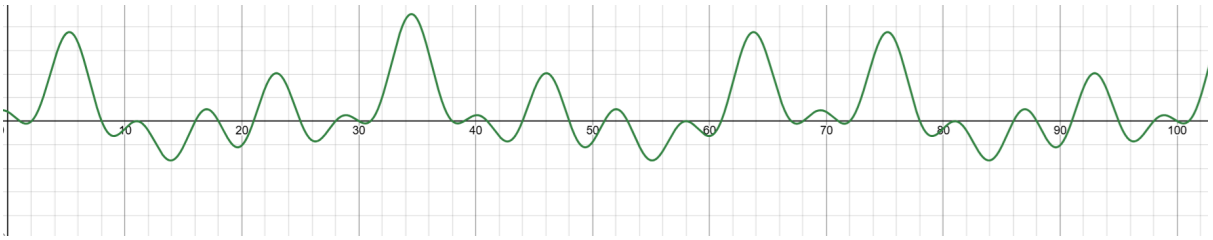
had to use the square root of a(x) through function b(x). Once b(x) was obtained, I then needed to convert this value (which belongs to natural numbers, not to the a(x) sequence) into the largest number less than or equal to b(x) within the numerical sequence of a(x), using formula c(x). Thus, the triad c(b(a(x))) (which I named C(x)) returns the index x in a(x) representing the upper bound of the interval containing all prime factors that compose a given a(x) inserted into c(b(x)).

$$b(x) = \text{floor}\left(\sqrt{x}\right)$$

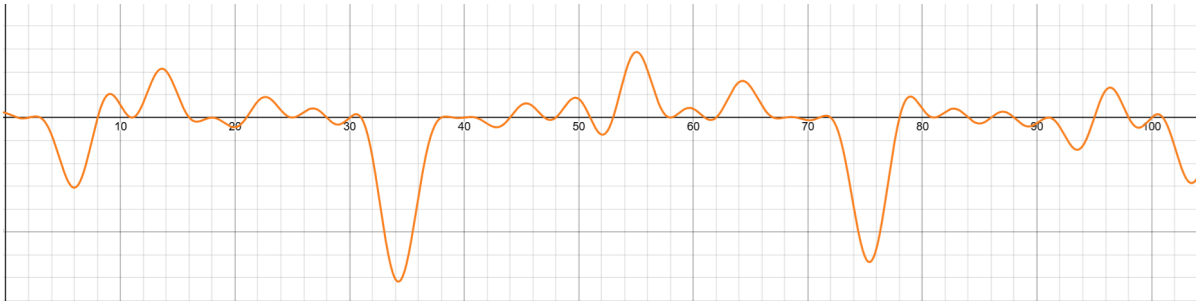
Here are the graphs of 4 distinct sinusoids, each defined by a different quantity of prime factors:
Sinusoid with a(1);



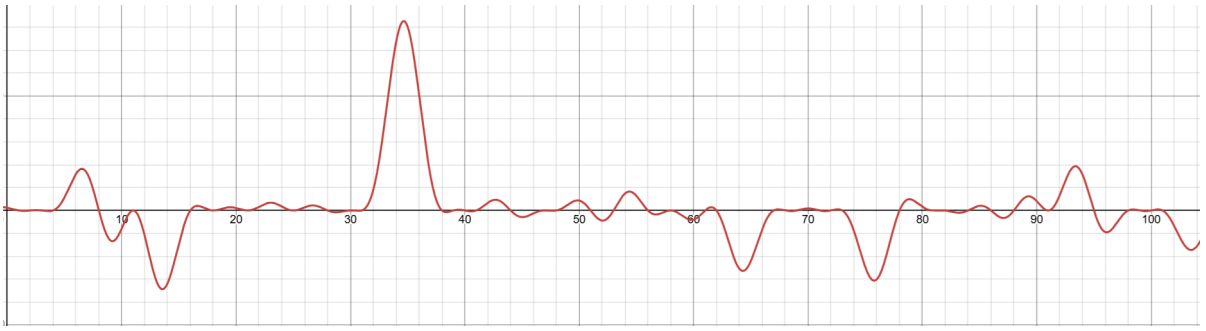
Sinusoid with a(1,2);



Sinusoid with a(1,2,3);



Sinusoid with a(1,2,3,4);



The solution seemed straightforward: simply count the zeros of the final sinusoid to determine the number of composites in a given interval. But reality proved otherwise. After painstakingly crafting

the zero-counting formula (E(n)), I noticed something was off—the method deterministically calculated all primes up to the 114th... but no further. I couldn't understand why.

$$E = n - \sum_{c=1}^{n} \left(\operatorname{ceil}\left(\left| \frac{|p(c)|}{|p(c)| + 1} \right| \right) \right)$$

Upon closer inspection of this critical interval, I realized the error wasn't in the formula (which was flawless) but in a limitation of Desmos: its precision for π. Desmos uses only the first 15 decimal digits of π and generally truncates calculations beyond the 15th decimal place. This introduced a minuscule margin of error (visible in the attached image) that became catastrophic for composite counting at larger numbers.

Reflecting on this, I recognized the formula's fundamental constraint—its dependence on π. As target numbers grew, more decimal digits of π would be required for accuracy. I concluded that irrational constants, at least within Desmos, were a dead end. This forced me to explore alternative approaches.

$$D(x) = 2\left(\sin\left(\frac{\pi}{11}(x+6) \right) + \cos\left(\frac{\pi 15}{22} \right) \right)$$

$$D(223) = 4.4408920985 \times 10^{-15}$$

3) SINUSOIDS SIMULATION VIA BINARY SEQUENCES.

After a couple of days of contemplation, one evening I realized it was possible to encode the information generated by the sinusoids using non-Diophantine functions. These functions would return:

•**0** if the input x in a(x) does *not* generate a composite (i.e., it's prime);

•**1** if x generates a composite.

This approach created binary sequences simulating the intersections (1) and non-intersections (0) of the sinusoids—where 0 represents a non-root (no composite detected) and 1 a root (composite detected). Each number had a unique binary sequence determined by the periodic distribution of its multiples. Here's the general formula I derived:

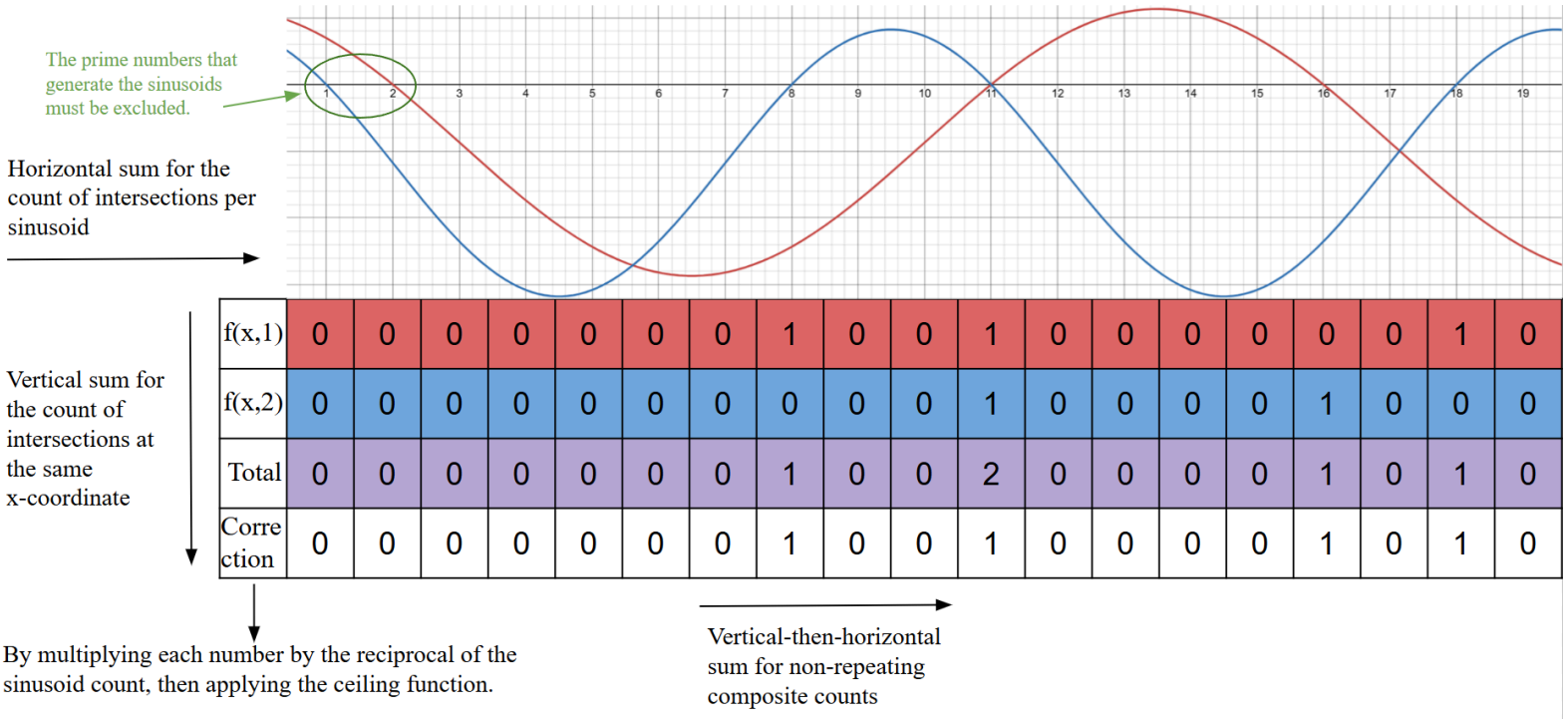
$$f(x,i) = \left(1 - \operatorname{sgn}\left(\operatorname{mod}\left(a(x), a(i)\right)\right)\right) \left| \operatorname{sgn}(i - x) \right|$$

Where "x" (1, 2, 3,...) is the variable and "i" is the number that determines the sequence (a(1), a(2),...).

Once the formula to generate each binary sequence (simulating a sinusoid's information) was created, two key properties emerged:

Single-Prime Composite Count: Summing the 1s and 0s of a *single* binary sequence yields the number of intersections (i.e., composites) for that specific prime. Summing these counts across all primes gives the *total composites with repetitions* (e.g., 35 is counted twice, as it's a multiple of both 5 and 7).

Per-x Intersection Density: By summing the outputs of all binary sequences sharing the same x but different i, we obtain a new sequence where each value represents the *number of sinusoid intersections* at that x. Multiplying each term in this sequence by the reciprocal of its sinusoid count and applying the ceiling function converts it back to a binary sequence. The sum of this final sequence counts *unique composites* (e.g., 35 is counted once).



f(x,1)	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0
f(x,2)	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
Total	0	0	0	0	0	0	0	1	0	0	2	0	0	0	0	1	0	1	0
Correction	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	1	0

All these operations are now performed by a new function E(x), which has replaced the previous one:

$$E(x) = \sum_{v=1}^{C(x)} \left(\sum_{k=p(v)+d(v)}^{Y(v,x)} \operatorname{ceil}\left(\frac{\sum_{i=1}^{v} \left(f(k,i) \right)}{C(x)} \right) \right)$$

Where Y(v,x), p(v), and d(v) serve to avoid unnecessary calculations. Without them, for any given number, the system would generate sequences for all x values starting from 1. However, these sequences only become effective for composite counting after a certain interval - there's no point

looking for multiples of large primes (like 29) before reaching the prime itself (i.e., before 29 in this case).
The functions p(v) and d(v) compute the lower bound of the interval, with the constraint of preventing overlap between endpoint and starting points of adjacent intervals, while Y(v,x) calculates the upper bound while ensuring it doesn't exceed the initially determined target number.

CONCLUSIONS

This document doesn't specify the implementation details of the correction algorithm. However, since Desmos doesn't support iterative calculations, I had to manually code a finite number of iterations (K) - enough to theoretically compute every prime up to the 1,000,000th prime. I acknowledge that some numbers within this range might require more iterations than provided. Currently, the largest tested number requiring maximal iterations is the 500,000th prime.
Furthermore, the method for using precomputation and estimation to significantly speed up calculations (reducing processing to just seconds depending on the target number and available precomputations) is not specified here. If this alternative method gains even minimal relevance, I will provide detailed explanations of the formulas (which I already possess and have implemented) that enable these optimizations.

The full implementation of this method is available in this Desmos workspace:
https://www.desmos.com/calculator/63j6rzlksh?lang=it

-Yūgen-