Unités sympathiques et conjecture de Goldbach : petite exploration numérique

Denise Vella-Chemla, pilotant l'IA claude

27 juin 2026

1 Définition

Soit n un entier pair. On dit que x, avec $1 < x < n$ et $\gcd(x, n) = 1$, est une *unité sympathique*[1] pour n si les quatre entiers

$$x, \qquad n - x, \qquad x^{-1} \bmod n, \qquad n - (x^{-1} \bmod n)$$

sont tous premiers. On a comme objectif d'étudier si tout entier pair possède au moins une unité sympathique.

Remarque : Si x est sympathique pour n, alors x est en particulier un décomposant de Goldbach de n (puisque x et $n - x$ sont premiers), et il en va de même pour $x^{-1} \bmod n$: c'est aussi un décomposant de Goldbach de n, distinct en général de x.

Exemple : Pour $n = 950$, $x = 13$ est une unité sympathique : $13^{-1} \equiv 877 \pmod{950}$, et les quatre nombres 13, $937 = 950 - 13$, 877, $73 = 950 - 877$ sont tous premiers.

2 Méthode de calcul et programme

Pour chaque n pair on parcourt les nombres premiers $x < n$ par ordre croissant (liste précalculée par crible d'Ératosthène) et on s'arrête à la première unité sympathique trouvée, ou on déclare n mauvais si aucun nombre premier $x < n$ ne convient. Le test $\gcd(x, n) = 1$ se réduit, x étant premier, à la condition $x \nmid n$. L'inverse modulaire $x^{-1} \bmod n$ est calculé directement (algorithme d'Euclide étendu).

Le programme en python est fourni en annexe.

3 Résultats numériques

Calcul exhaustif effectué jusqu'à $n = 10\,000\,000$:

1. L'idée des unités sympathiques m'est venue de la lecture des "éléments de l'orbite de ω" page 284 de cet article https://ems.press/content/serial-article-files/47669.

Borne	n pairs testés	sympathiques	pourcentage
1 000	499	356	71.34%
10 000	4 999	4 352	87.06%
100 000	49 999	49 045	98.09%
1 000 000	499 999	499 045	99.81%
2 000 000	999 999	999 045	99.90%
5 000 000	2 499 999	2 499 045	99.96%
10 000 000	4 999 999	4 999 045	99.98%

Fait surprenant constaté : Le nombre *absolu* de n mauvais cesse de croître au-delà de $n = 93\,824$: il vaut exactement 954 sur tout l'intervalle testé $[4, 10^7]$, et aucune nouvelle exception n'apparaît entre 10^5 et 10^7.

La densité relative des exceptions (proportion d'exceptions parmi les n pairs, par tranche) décroît nettement :

Tranche de n	exceptions	densité
$[1, 10)$	2	50.0%
$[10, 100)$	16	35.6%
$[100, 1\,000)$	125	27.8%
$[1\,000, 10\,000)$	504	11.2%
$[10\,000, 100\,000)$	307	0.68%
$[10^5, 10^7]$	0	0%

4 Étude des 954 exceptions

- Toutes les exceptions sont paires par construction ; seules 2 d'entre elles (6 et... aucune autre en fait, vérifié : 2/954, soit 0.2%) sont divisibles par 3. La quasi-totalité des exceptions a donc une partie impaire premières avec 3.
- 493 exceptions sur 954 (51.7%) ont $v_2(n) = 1$, c'est-à-dire sont de la forme $n = 2m$ avec m impair ; 321 d'entre elles sont de la forme $n = 2p$ avec p premier.
- Le plus grand facteur premier apparaissant dans la factorisation d'une exception est 36 313 (dans $n = 72\,626 = 2 \times 36\,313$), nettement inférieur à la plus grande exception elle-même (93 824).
- La liste complète des 954 exceptions est fournie en annexe et dans le fichier joint.

5 Discussion

Conjecture 1 (empirique, non démontrée) *L'ensemble des entiers pairs n ne possédant aucune unité sympathique est fini, et égal exactement à l'ensemble des* 954 *valeurs listées en annexe, la plus grande étant* 93 824*.*

Ce que les données suggèrent, sans le démontrer :

- La stabilité du compte (954 inchangé sur deux ordres de grandeur supplémentaires, jusqu'à 10^7) est un indice statistique raisonnablement fort en faveur de la finitude de la liste, mais ne constitue en aucun cas une preuve. De nombreux phénomènes en théorie des nombres semblent se stabiliser sur des plages numériques considérables avant de réapparaître à des échelles bien plus grandes (l'exemple historique le plus connu étant le nombre de Skewes, lié au signe de $\pi(x) - \mathrm{li}(x)$).
- Si la conjecture ci-dessus était vraie, elle constituerait un énoncé strictement plus fort que la conjecture de Goldbach elle-même : non seulement tout n pair suffisamment grand admettrait un décomposant de Goldbach, mais il en admettrait un dont l'inverse modulaire serait *lui-même* un second décomposant de Goldbach valide.
- La condition supplémentaire portant sur $x^{-1} \bmod n$ introduit une contrainte multiplicative non standard, tordue par l'inversion modulaire. Elle ne se prête pas plus que la conjecture de Goldbach classique aux méthodes de crible actuelles : la même *obstruction de parité* qui empêche une preuve de Goldbach par crible pur s'applique ici, sans qu'aucun argument supplémentaire ne semble en atténuer la portée.
- Une vérification plus poussée (par exemple jusqu'à $n = 10^9$, avec une implémentation en C ou utilisant des bibliothèques de tests de primalité optimisées) permettrait de renforcer, ou potentiellement d'infirmer, la conjecture empirique ci-dessus.

Programme python

```
#!/usr/bin/env python3
"""
Test des "unites sympathiques" de Goldbach
==========================================

Pour n pair, on cherche x avec gcd(x,n)=1 tel que les quatre nombres
    x, n-x, x^{-1} mod n, n - (x^{-1} mod n)
soient tous premiers.

n est dit "sympathique" s'il existe un tel x ; "mauvais" sinon.

Ce script :
1. recherche, pour chaque n pair jusqu'a une borne LIMIT, l'existence
   d'une telle unite (recherche exhaustive sur les nombres premiers
   x < n, dans l'ordre croissant, et arret au premier trouve) ;
2. calcule le pourcentage de n "sympathiques" ;
3. liste les n "mauvais" (exceptions).

Optimisations par rapport a une implementation naive :
- un seul crible d'Eratosthene (bytearray) precalcule une fois ;
- parcours de x uniquement parmi les nombres premiers (liste precalculee),
  au lieu de tester un par un tous les entiers ;
- inversion modulaire via pow(x, -1, n) (Python >= 3.8, calcul rapide
  par l'algorithme d'Euclide etendu en C) ;
- filtre gcd(x,n)=1 remplace par "n % x != 0" car x est premier
  (gcd(x,n) != 1 et x premier <=> x divise n).

Usage :
```

```
    python3 goldbach_units.py [LIMIT]

Exemple :
    python3 goldbach_units.py 1000000
"""

import sys
import time


def sieve(limit):
    """Crible d'Eratosthene : renvoie un bytearray is_prime[0..limit]."""
    is_p = bytearray([1]) * (limit + 1)
    is_p[0] = is_p[1] = 0
    for i in range(2, int(limit ** 0.5) + 1):
        if is_p[i]:
            is_p[i * i::i] = bytearray(len(is_p[i * i::i]))
    return is_p


def find_nice_unit(n, is_prime, primes_list):
    """
    Cherche le plus petit x premier, x < n, n % x != 0, tel que
    x, n-x, x^{-1} mod n, n - x^{-1} mod n soient tous premiers.

    Renvoie le tuple (x, n-x, inv, n-inv) si trouve, sinon None.
    """
    for x in primes_list:
        if x >= n:
            break
        if n % x == 0:          # equivalent a gcd(x,n) != 1
                                # puisque x est premier
            continue
        if not is_prime[n - x]:
            continue
        inv = pow(x, -1, n)
        if is_prime[inv] and is_prime[n - inv]:
            #print(n,' --> ',x,'  ',n-x,'  ',inv,'  ',n-inv)
            return (x, n - x, inv, n - inv)
    return None


def run(limit, is_prime, primes_list, start=4):
    """
    Parcourt tous les n pairs de start a limit inclus.
    Renvoie (total, nice, pct, liste_des_mauvais_n).
    """
    total = 0
    nice = 0
    bad = []
    for n in range(start, limit + 1, 2):
        total += 1
        if find_nice_unit(n, is_prime, primes_list) is not None:
            nice += 1
        else:
```

```
            bad.append(n)
    pct = 100 * nice / total if total else 0.0
    return total, nice, pct, bad

limit = int(sys.argv[1]) if len(sys.argv) > 1 else 1_000_000
t0 = time.time()
is_prime = sieve(limit + 10)
primes_list = [i for i in range(2, limit + 10) if is_prime[i]]
print(f"Crible construit en {time.time()-t0:.2f}s "
      f"({len(primes_list)} nombres premiers <= {limit+10})")
t0 = time.time()
total, nice, pct, bad = run(limit, is_prime, primes_list)
dt = time.time() - t0
print(f"\nn pairs de 4 a {limit} :")
print(f"  total       = {total}")
print(f"  sympathiques= {nice}  ({pct:.4f}%)")
print(f"  mauvais     = {len(bad)}")
print(f"  temps       = {dt:.2f}s")
if bad:
    print(f"\nListe des {len(bad)} exceptions :")
    print(bad)
    print(f"\nPlus grande exception trouvee : {max(bad)}")
```

Annexe : liste complète des 954 exceptions

```
4, 6, 14, 22, 26, 32, 38, 44, 46, 52, 62, 64, 68, 76, 82, 86, 88, 94, 104, 116,
122, 124, 128, 134, 136, 142, 146, 148, 152, 154, 158, 164, 166, 172, 182, 190,
194, 200, 202, 206, 218, 226, 238, 242, 244, 250, 254, 258, 262, 266, 268, 274,
278, 284, 286, 302, 316, 320, 332, 334, 344, 350, 356, 362, 368, 376, 394, 398,
404, 412, 434, 446, 458, 464, 466, 478, 488, 494, 496, 500, 502, 512, 514, 524,
530, 536, 538, 542, 544, 554, 556, 568, 592, 602, 608, 614, 620, 622, 632, 634,
638, 644, 650, 652, 656, 664, 668, 676, 692, 704, 712, 718, 724, 734, 736, 740,
742, 746, 766, 782, 788, 800, 808, 818, 820, 824, 832, 838, 842, 844, 862, 866,
884, 886, 916, 928, 934, 938, 944, 956, 958, 962, 992, 1004, 1006, 1010, 1012,
1018, 1022, 1030, 1036, 1048, 1052, 1064, 1096, 1112, 1124, 1126, 1136, 1138,
1148, 1154, 1156, 1174, 1178, 1208, 1214, 1228, 1234, 1238, 1256, 1262, 1264,
1280, 1286, 1298, 1304, 1306, 1310, 1312, 1322, 1324, 1348, 1352, 1376, 1384,
1408, 1412, 1414, 1418, 1426, 1436, 1438, 1444, 1456, 1472, 1478, 1498, 1508,
1514, 1516, 1532, 1538, 1544, 1556, 1558, 1580, 1634, 1642, 1646, 1664, 1666,
1676, 1678, 1690, 1712, 1718, 1726, 1732, 1742, 1754, 1756, 1762, 1768, 1774,
1778, 1786, 1790, 1804, 1814, 1816, 1822, 1826, 1828, 1840, 1846, 1850, 1852,
1858, 1862, 1868, 1874, 1876, 1888, 1898, 1904, 1916, 1942, 1954, 1958, 1972,
1976, 1990, 2008, 2012, 2026, 2032, 2036, 2038, 2054, 2078, 2084, 2086, 2114,
2120, 2126, 2132, 2138, 2144, 2156, 2158, 2164, 2188, 2198, 2204, 2212, 2216,
2222, 2242, 2252, 2276, 2290, 2300, 2302, 2306, 2326, 2348, 2354, 2390, 2414,
2416, 2422, 2444, 2452, 2456, 2462, 2482, 2488, 2506, 2512, 2524, 2528, 2534,
2546, 2558, 2572, 2594, 2608, 2612, 2624, 2644, 2654, 2672, 2684, 2696, 2732,
2738, 2746, 2756, 2758, 2762, 2774, 2776, 2788, 2804, 2812, 2834, 2846, 2854,
2882, 2896, 2908, 2918, 2924, 2942, 2948, 2956, 2966, 2978, 2992, 3022, 3028,
```

3032, 3038, 3056, 3064, 3088, 3092, 3094, 3100, 3118, 3122, 3128, 3152, 3158,
3166, 3176, 3202, 3214, 3218, 3244, 3272, 3296, 3302, 3316, 3328, 3334, 3356,
3362, 3364, 3368, 3382, 3392, 3398, 3406, 3412, 3424, 3436, 3466, 3476, 3506,
3518, 3542, 3554, 3572, 3578, 3602, 3622, 3626, 3638, 3662, 3664, 3694, 3698,
3704, 3742, 3764, 3766, 3812, 3814, 3844, 3848, 3862, 3868, 3874, 3884, 3898,
3916, 3934, 3964, 3974, 3988, 3994, 4006, 4022, 4028, 4048, 4052, 4054, 4066,
4078, 4084, 4096, 4108, 4112, 4118, 4132, 4142, 4166, 4168, 4174, 4178, 4226,
4232, 4244, 4246, 4256, 4274, 4306, 4316, 4334, 4336, 4346, 4352, 4378, 4432,
4436, 4454, 4484, 4492, 4504, 4538, 4562, 4568, 4604, 4624, 4682, 4688, 4706,
4754, 4762, 4768, 4772, 4826, 4852, 4870, 4874, 4882, 4886, 4894, 4954, 4960,
4984, 4990, 4996, 5008, 5038, 5042, 5074, 5084, 5086, 5114, 5126, 5156, 5182,
5186, 5228, 5266, 5272, 5284, 5312, 5344, 5354, 5366, 5372, 5378, 5398, 5422,
5428, 5486, 5492, 5494, 5524, 5528, 5548, 5552, 5578, 5608, 5620, 5624, 5638,
5644, 5648, 5774, 5792, 5804, 5884, 5998, 6008, 6014, 6016, 6044, 6074, 6086,
6092, 6206, 6214, 6286, 6388, 6394, 6406, 6434, 6506, 6512, 6548, 6608, 6644,
6646, 6704, 6832, 6844, 6854, 6884, 6896, 6898, 6928, 6934, 6976, 7058, 7066,
7118, 7126, 7132, 7156, 7162, 7214, 7226, 7244, 7246, 7252, 7274, 7288, 7294,
7372, 7384, 7414, 7432, 7502, 7556, 7562, 7576, 7604, 7634, 7664, 7682, 7712,
7718, 7808, 7814, 7838, 7864, 7892, 7906, 7912, 7928, 7966, 8032, 8036, 8038,
8056, 8068, 8072, 8116, 8168, 8186, 8206, 8308, 8314, 8392, 8432, 8444, 8458,
8488, 8494, 8632, 8654, 8672, 8674, 8726, 8732, 8798, 8818, 8828, 8836, 8842,
8938, 8992, 9034, 9046, 9056, 9082, 9218, 9224, 9232, 9286, 9316, 9322, 9346,
9376, 9416, 9476, 9482, 9488, 9502, 9572, 9602, 9608, 9682, 9686, 9722, 9806,
9862, 9874, 9886, 9902, 9922, 9952, 10006, 10082, 10084, 10094, 10148, 10162,
10292, 10298, 10312, 10358, 10364, 10366, 10442, 10474, 10546, 10592, 10634,
10636, 10748, 10756, 10834, 10844, 10886, 10898, 10916, 10936, 10964, 10966,
11026, 11084, 11096, 11138, 11174, 11188, 11278, 11288, 11294, 11306, 11318,
11384, 11476, 11488, 11506, 11638, 11656, 11672, 11714, 11824, 11918, 11926,
11996, 12014, 12074, 12182, 12202, 12328, 12412, 12452, 12482, 12538, 12548,
12550, 12556, 12602, 12608, 12626, 12634, 12638, 12722, 12772, 12812, 12854,
12856, 12938, 13016, 13088, 13118, 13142, 13198, 13232, 13304, 13372, 13382,
13408, 13466, 13508, 13526, 13646, 13874, 13894, 13918, 13924, 14038, 14048,
14066, 14132, 14236, 14282, 14302, 14338, 14356, 14404, 14528, 14624, 14638,
14666, 14902, 14918, 14948, 15038, 15058, 15122, 15148, 15164, 15272, 15304,
15352, 15446, 15518, 15524, 15574, 15578, 15596, 15634, 15692, 15758, 15814,
15842, 15952, 16022, 16024, 16052, 16076, 16096, 16118, 16234, 16322, 16594,
16624, 16714, 16778, 16838, 16844, 17048, 17056, 17072, 17216, 17308, 17414,
17428, 17462, 17672, 17704, 17708, 17726, 17792, 17918, 17938, 18034, 18136,
18148, 18266, 18428, 18554, 18724, 18842, 18914, 19066, 19102, 19286, 19612,
19616, 19678, 19762, 19774, 19814, 19862, 19868, 20122, 20186, 20198, 20278,
20438, 20576, 20702, 20812, 20894, 21152, 21254, 21314, 21466, 21562, 22024,
22076, 22252, 22436, 22618, 22712, 22886, 22934, 23228, 23438, 23536, 23678,
23848, 23948, 24074, 24532, 24604, 24664, 24722, 24854, 24856, 25244, 25492,
25642, 25822, 25876, 25946, 26014, 26084, 26288, 26492, 26594, 27184, 27424,
27556, 27584, 28346, 28558, 28618, 29168, 29272, 29584, 29594, 29696, 30014,
30278, 30346, 30506, 30682, 30784, 30922, 31244, 31258, 31714, 31774, 32056,
32204, 32434, 32612, 32678, 32948, 32954, 33272, 33658, 33692, 34492, 35914,
36118, 36544, 36572, 36742, 37826, 37838, 38354, 39028, 39202, 39862, 40492,

40504, 41632, 42688, 43138, 43612, 46496, 46532, 46748, 46894, 46936, 47666,
47986, 49628, 49762, 50366, 50486, 53294, 53626, 55082, 58028, 58774, 58886,
59264, 60122, 60538, 62948, 63268, 63464, 63586, 64822, 65972, 68072, 69406,
70306, 70718, 72626, 93824.

Annexe : Programme initial

Ci-dessous, le programme initial fourni à l'IA claude, lent car trop de tests.

```
import math
# -*- coding: utf-8 -*-

import math
from math import log
import numpy as np
import  matplotlib.pyplot as plt
import time

def prime_sieve(N):
    is_prime = np.full(N, True)
    is_prime[:2] = False
    for p in range(2, math.isqrt(N) + 1):
        if is_prime[p]:
            is_prime[p*p::p] = False
    return np.nonzero(is_prime)[0]

class Primes():
    def __init__(self, N):
        self.__primes = prime_sieve(N)
    def __str__(self):
        return str(self.__primes)
    def __iter__(self):
        return iter(self.__primes)
    def __len__(self):
        return self.__primes.size
    def __getitem__(self, k):
        return self.__primes[k]
    def __contains__(self, x):
        k = self.index(x)
        return k < len(self) and self.__primes[k] == x
    def index(self, x):
        return np.searchsorted(self.__primes, x, side='left')
    def count(self, x):
        return np.searchsorted(self.__primes, x, side='right')
    def range(self, start, stop, step=1):
        return self.__primes[self.index(start):self.index(stop):step]
    def factors(self, n):
        if n in self:
            return np.array([n])
        else:
            P = self.range(2, n//2 + 1)
            return P[n % P == 0]
```

```
def pgcd(m,n):
    while (m != 0):
        r = n % m
        n = m
        m = r
    return(n)

def premier(atester):
    k = 2
    if atester in [0,1]: return False
    if atester in [2,3,5,7]: return True
    while True:
        if k * k > atester: return True
        else:
            if atester % k == 0: return False
            else: k = k+1

nmax = 1002
P = Primes(nmax)
nbtestes = 0
nbyoupi = 0
for n in range(6,nmax,2):
    if not n//2 in P:
        nbtestes += 1
        for x in range(3,n-2,2):
            if pgcd(x,n) == 1:
                inversex = 2
                while ((inversex * x)%n != 1) and (inversex < n):
                    inversex += 1
                if x in P and n-x in P
                  and inversex in P and n-inversex in P:
                    print(n,':::: voici un couple de dg inverses : ',
                            x,'  ',inversex,'  ',n-x,'  ',n-inversex)
                    nbyoupi += 1
                    break

print('ratio = ',nbyoupi/nbtestes, nbtestes-nbyoupi)
```