Algorithme d'Euclide étendu étendu pour décomposants de Goldbach

Denise Vella-Chemla

18.12.2021

Dans de précédentes notes (voir [1], [2], [3]), on a démontré qu'un nombre premier supérieur à $\sqrt{n}$ et inférieur ou égal à $n/2$, et qui n'est jamais congru à n modulo tout nombre premier inférieur à $\sqrt{n}$ est un décomposant de Goldbach de n.

Ici, pour résoudre un système de congruences, on utilise l'algorithme d'Euclide étendu (voir [4], [5]) : soit à résoudre le système de congruences avec les m_i des nombres tous premiers :

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \dots \\ x \equiv a_i \pmod{m_i} \end{cases}$$

On pose $M = \prod_{1 \leqslant k \leqslant i} m_i$.

Le théorème des restes chinois (voir une vidéo pédagogique [6]) permet d'obtenir la solution générale

$$x \equiv \left[\sum_{1 \leqslant k \leqslant i} a_i u_i \frac{M}{m_i} \right] \pmod{M}$$

L'algorithme d'Euclide étendu permet d'obtenir les différents u_i associés aux $\frac{M}{m_i}$ par résolution de l'équation :

$$u.\frac{M}{m_i} + v.m_i = 1.$$

Pour résoudre plusieurs systèmes de congruences, résultant de la combinatoire applicable à tous les restes possibles selon les différents modules, on étend l'algorithme d'Euclide étendu, d'où l'expression "algorithme d'Euclide étendu étendu" : on considère qu'à la recherche des décomposants de Goldbach d'un nombre pair n, selon tout module m_i, tous les restes de 1 à $m_i - 1$ sont possibles, sauf le reste de n.

Ci-dessous, le programme qui code cet algorithme d'Euclide étendu étendu :

```
import math, itertools

def prime(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
```

```
def listerestespossibles(n, k):
    l = [ ]
    for x in range(1,k):
        if n%k != x:
            l.append(x)
    return l

def EuclideEtendu(a, b):
    r0, u0, v0, r1, u1, v1 = a, 1, 0, b, 0, 1
    while r0 % r1 != 0:
        q = r0 // r1
        r0, u0, v0, r1, u1, v1 = r1, u1, v1, r0%r1, u0-q*u1, v0-q*v1
    return u1

def chinois(A, M):
    P = math.prod(M)
    x = 0
    for i in range(len(M)):
        mi = M[i]
        Mi = P//mi
        ui = EuclideEtendu(Mi, mi)
        x += A[i]*Mi*ui
    return x%P, P

for n in range(22,102,2):
    print(n,' —> ( 0, ',end='')
    modules = [2]
    racine = int(math.sqrt(n))
    for p in range(3,racine+1,2):
        if prime(p):
            modules.append(p)
            print(n%p,', ',end='')
    print(')')
    print('modules = ',modules)
    restes=[0 for k in modules]
    indice = 0
    for k in modules:
        restes[indice] = listerestespossibles(n, k)
        indice = indice+1
    print('restes = ',restes)
    for resti in itertools.product(*restes):
        x, P = chinois(resti, modules)
        if x > 1 and x <= n/2:
            print(f'x = x [P], resti')
    print('')
```

Ci-dessous le résultat du programme

22 —> (0, 1 ,)
x = 5 [6], (1, 2)

24 —> (0, 0 ,)
x = 5 [6], (1, 2)

26 —> (0, 2 , 1 ,)
x = 7 [30], (1, 1, 2)
x = 13 [30], (1, 1, 3)

28 —> (0, 1 , 3 ,)
x = 11 [30], (1, 2, 1)

30 —> (0, 0 , 0 ,)
x = 7 [30], (1, 1, 2)
x = 13 [30], (1, 1, 3)
x = 11 [30], (1, 2, 1)

32 —> (0, 2 , 2 ,)
x = 13 [30], (1, 1, 3)

34 —> (0, 1 , 4 ,)
x = 11 [30], (1, 2, 1)
x = 17 [30], (1, 2, 2)

36 —> (0, 0 , 1 ,)
x = 7 [30], (1, 1, 2)
x = 13 [30], (1, 1, 3)
x = 17 [30], (1, 2, 2)

38 —> (0, 2 , 3 ,)
x = 7 [30], (1, 1, 2)
x = 19 [30], (1, 1, 4)

40 —> (0, 1 , 0 ,)
x = 11 [30], (1, 2, 1)
x = 17 [30], (1, 2, 2)

42 —> (0, 0 , 2 ,)
x = 13 [30], (1, 1, 3)
x = 19 [30], (1, 1, 4)
x = 11 [30], (1, 2, 1)

44 —> (0, 2 , 4 ,)
x = 7 [30], (1, 1, 2)
x = 13 [30], (1, 1, 3)

46 —> (0, 1 , 1 ,)
x = 17 [30], (1, 2, 2)
x = 23 [30], (1, 2, 3)

48 —> (0, 0 , 3 ,)
x = 7 [30], (1, 1, 2)
x = 19 [30], (1, 1, 4)
x = 11 [30], (1, 2, 1)
x = 17 [30], (1, 2, 2)

50 —> (0, 2 , 0 , 1 ,)
x = 13 [210], (1, 1, 3, 6)
x = 19 [210], (1, 1, 4, 5)
52 —> (0, 1 , 2 , 3 ,)
x = 11 [210], (1, 2, 1, 4)
x = 23 [210], (1, 2, 3, 2)

54 —> (0, 0 , 4 , 5 ,)
x = 13 [210], (1, 1, 3, 6)
x = 11 [210], (1, 2, 1, 4)
x = 17 [210], (1, 2, 2, 3)
x = 23 [210], (1, 2, 3, 2)

56 —> (0, 2 , 1 , 0 ,)
x = 13 [210], (1, 1, 3, 6)
x = 19 [210], (1, 1, 4, 5)

58 —> (0, 1 , 3 , 2 ,)
x = 11 [210], (1, 2, 1, 4)
x = 17 [210], (1, 2, 2, 3)
x = 29 [210], (1, 2, 4, 1)

60 —> (0, 0 , 0 , 4 ,)
x = 13 [210], (1, 1, 3, 6)
x = 19 [210], (1, 1, 4, 5)
x = 17 [210], (1, 2, 2, 3)
x = 23 [210], (1, 2, 3, 2)
x = 29 [210], (1, 2, 4, 1)

62 —> (0, 2 , 2 , 6 ,)
x = 31 [210], (1, 1, 1, 3)
x = 19 [210], (1, 1, 4, 5)

64 —> (0, 1 , 4 , 1 ,)
x = 11 [210], (1, 2, 1, 4)
x = 17 [210], (1, 2, 2, 3)
x = 23 [210], (1, 2, 3, 2)

66 —> (0, 0 , 1 , 3 ,)
x = 13 [210], (1, 1, 3, 6)
x = 19 [210], (1, 1, 4, 5)
x = 23 [210], (1, 2, 3, 2)
x = 29 [210], (1, 2, 4, 1)

68 —> (0, 2 , 3 , 5 ,)
x = 31 [210], (1, 1, 1, 3)

70 —> (0, 1 , 0 , 0 ,)
x = 11 [210], (1, 2, 1, 4)
x = 17 [210], (1, 2, 2, 3)
x = 23 [210], (1, 2, 3, 2)
x = 29 [210], (1, 2, 4, 1)

72 —> (0, 0 , 2 , 2 ,)
x = 31 [210], (1, 1, 1, 3)
x = 13 [210], (1, 1, 3, 6)
x = 19 [210], (1, 1, 4, 5)
x = 11 [210], (1, 2, 1, 4)
x = 29 [210], (1, 2, 4, 1)

74 —> (0, 2 , 4 , 4 ,)
x = 31 [210], (1, 1, 1, 3)
x = 37 [210], (1, 1, 2, 2)
x = 13 [210], (1, 1, 3, 6)

76 —> (0, 1 , 1 , 6 ,)
x = 17 [210], (1, 2, 2, 3)
x = 23 [210], (1, 2, 3, 2)
x = 29 [210], (1, 2, 4, 1)

78 —> (0, 0 , 3 , 1 ,)
x = 31 [210], (1, 1, 1, 3)
x = 37 [210], (1, 1, 2, 2)
x = 19 [210], (1, 1, 4, 5)
x = 11 [210], (1, 2, 1, 4)
x = 17 [210], (1, 2, 2, 3)

80 —> (0, 2 , 0 , 3 ,)
x = 37 [210], (1, 1, 2, 2)
x = 13 [210], (1, 1, 3, 6)
x = 19 [210], (1, 1, 4, 5)

82 —> (0, 1 , 2 , 5 ,)
x = 11 [210], (1, 2, 1, 4)
x = 41 [210], (1, 2, 1, 6)
x = 23 [210], (1, 2, 3, 2)
x = 29 [210], (1, 2, 4, 1)

84 —> (0, 0 , 4 , 0 ,)
x = 31 [210], (1, 1, 1, 3)
x = 37 [210], (1, 1, 2, 2)
x = 13 [210], (1, 1, 3, 6)
x = 11 [210], (1, 2, 1, 4)
x = 41 [210], (1, 2, 1, 6)
x = 17 [210], (1, 2, 2, 3)
x = 23 [210], (1, 2, 3, 2)

86 —> (0, 2 , 1 , 2 ,)
x = 43 [210], (1, 1, 3, 1)
x = 13 [210], (1, 1, 3, 6)
x = 19 [210], (1, 1, 4, 5)

88 —> (0, 1 , 3 , 4 ,)
x = 41 [210], (1, 2, 1, 6)
x = 17 [210], (1, 2, 2, 3)
x = 29 [210], (1, 2, 4, 1)

90 —> (0, 0 , 0 , 6 ,)
x = 31 [210], (1, 1, 1, 3)
x = 37 [210], (1, 1, 2, 2)
x = 43 [210], (1, 1, 3, 1)
x = 19 [210], (1, 1, 4, 5)
x = 11 [210], (1, 2, 1, 4)
x = 17 [210], (1, 2, 2, 3)
x = 23 [210], (1, 2, 3, 2)
x = 29 [210], (1, 2, 4, 1)

92 —> (0, 2 , 2 , 1 ,)
x = 31 [210], (1, 1, 1, 3)
x = 13 [210], (1, 1, 3, 6)
x = 19 [210], (1, 1, 4, 5)

94 —> (0, 1 , 4 , 3 ,)
x = 11 [210], (1, 2, 1, 4)
x = 41 [210], (1, 2, 1, 6)
x = 47 [210], (1, 2, 2, 5)
x = 23 [210], (1, 2, 3, 2)

96 —> (0, 0 , 1 , 5 ,)
x = 37 [210], (1, 1, 2, 2)
x = 43 [210], (1, 1, 3, 1)
x = 13 [210], (1, 1, 3, 6)
x = 17 [210], (1, 2, 2, 3)
x = 23 [210], (1, 2, 3, 2)
x = 29 [210], (1, 2, 4, 1)

98 —> (0, 2 , 3 , 0 ,)
x = 31 [210], (1, 1, 1, 3)
x = 37 [210], (1, 1, 2, 2)
x = 19 [210], (1, 1, 4, 5)

100 —> (0, 1 , 0 , 2 ,)
x = 11 [210], (1, 2, 1, 4)
x = 41 [210], (1, 2, 1, 6)
x = 17 [210], (1, 2, 2, 3)
x = 47 [210], (1, 2, 2, 5)
x = 29 [210], (1, 2, 4, 1)

Références

[1] Denise Vella-Chemla, Réécrire, http://denise.vella.chemla.free.fr/jade1.pdf.

[2] Denise Vella-Chemla, Continuer de suivre Galois, http://denise.vella.chemla.free.fr/invariante.pdf.

[3] Denise Vella-Chemla, Dancing Links pour conjecture de Goldbach, http://denise.vella.chemla.free.fr/DLpourCG.pdf.

[4] Donald Knuth, The Art of Computer Programming, vol. 1, Fundamental algorithms, Third edition, 1997 (pages 13 à 18).

[5] Donald Knuth, The Art of Computer Programming, vol. 2, Seminumerical algorithms, Third edition, 1997 (pages 342 à 354).

[6] Gilles Bailly-Maitre, vidéo pédagogique "Théorème des restes chinois, version pratique", https://www.youtube.com/watch?v=xIzoYxOqmXs.