Préambule :

Pour tout A⩽ N, entier naturel positif impair , qui progresse **modulo 2 de 1 à N**, fixons l'entier **2N=18** qui va vérifier la conjecture , lui aussi progresse modulo 2, prenons un nombre premier **P ⩽ √2N**, que l'on va utiliser avec leur reste **R** de la division Euclidienne de **2N par P** pour calculer ces congruences .
L'algorithme de Goldbach, va donc utiliser les congruences, pour construire un axe des ordonnées dans le sens ↓: de l'amont vers l'avale.
(« Indiquer aussi le nombre de nombres premiers **q ∈ [N ; 2N]** pour toute limite 2N décomposée en somme de deux nombres premiers (p+q). **»)**
On va donc pour tout entier **A⩽ N** calculer le reste de **2N** par **P** afin de vérifier si **[,1]** est congru ou pas à **2N** modulo **P**. Chaque entier **[,1,]** tel que **1 ≡ 2N[P]**, initialisera un (rayon ou diagonale) d'entiers **A** congrus à 2N modulo P, sur **l'axe des ordonnées** dans ce sens ↓, qui seront marquées **0** ; voir illustration ci-dessous .

Il vient par obligation que : si **1 ≡ 2N[P]** par conséquent lorsque N augmente de 1 et donc 2N augmente de 2 , **(1+2) ≡ 2N+2 [P].** (« *Exemple : 1≡ 28 [P] d'où (1+2) ≡ 30 [P], 28 – 1 ⇔ 30 – 3 ≠ q premier, le contraire contredirait le TFA et le TNP.* »)

Inversement : si **[,1,]** tel que **1 ≢ 2N[P]**, il initialisera une diagonale d'entiers **A** non congrus à 2N modulo P, qui seront marquées **1**. Donc : (« *Exemple : 1≢ 30 [P] d'où (1+2) ≢ 32 [P] , 30 – 1 ⇔ 32 – 3 = q premier.* ») . Conséquence directe du décalage d'un rang des congruences, lorsque 2N augmente de 2 , on a l'égalité :

(2N – A) ⇔ (2N +2) – (A + 2) , équivalent à ***(A+2) ≢ (2N+2) [P]*** *ou l'inverse* ***(A+2) ≡ (2N+2) [P].***

Chaque diagonale parcourt l'ensemble des entiers **A** impairs de **1 à N**, pour toutes limites **N** fixée , ***qui viendront croiser*** l'axe des abscisses d'Ératosthène criblé de **1 à N** .
Quelque soit l'entier **2N** > 4 qui vérifie la conjecture, tel que **2N=p′+q** ; alors la conjecture est aussi vérifiée pour **2N+2** .

Conséquence du décalage récurrent des congruences, il existe pour la suite **N** qui a été criblée et donné le nombre de décompositions **p′+ q = 2N,** des entiers **A ≢2N[P]** qui ***précèdent*** **A+2** ***premier p′*** et de part le fait de ce décalage d'un rang des congruences qui s'ensuit, donne le nombre de solutions pour la limite **N+1** suivante, c'est à dire le nombre de décompositions de **2N + 2** , à une unité près.

Ce nombre de décompositions d'un entier **2N** est par conséquent toujours vérifié lors de la limite précédente **N–1** criblée, dû à ce décalage d'un rang des congruences correspondant à l'égalité suivante :

(2N – A) ⇔ (2N +2) – (A + 2).

De la même manière, que l'on connaît le nombre de nombres premiers ***q ∈ [N,2N],*** *qui a été vérifié et ne peut varier au maximum que de* ***1*** pour la limite suivante **2N + 2** ; ce qui implique le même nombre de premiers ***q ∈ [(N+1) , (2N+2)]*** à une unité près. Ce qui serait impossible si la conjecture était fausse, car elle est une conséquence directe du TNP.

Propriété des congruences petit rappel, ***2n – A = B*** *:*

Il existe y et y' tel que : ***2n = P*y + R et A = P*y' + R => 2n – A = P*(y – y'); donc P divise 2n – A = B.*** *Inversement si* ***y*** *n'existe pas , alors* ***P*** *ne divise pas la différence* ***2n – A = B => q*** *qui est donc un nombre premier* ***∈ [n;2n]*** *ayant pour antécédent* ***A≢ 2n[P].***

1.) le programmes Goldbach : sert à initialiser les diagonales d'entiers A, congrues ou non à 2N modulo P sur l'axe des ordonnées ↓ de Goldbach , dans le sens inverse du plan ci-dessous .

2.) Cet axe vient par conséquent croiser l'axe des abscisses → *représentant* les nombres premiers **p'** d'Ératosthène de 1 à N qui ont été criblés. Chaque rang des ordonnées correspond à chaque rang des abscisses :

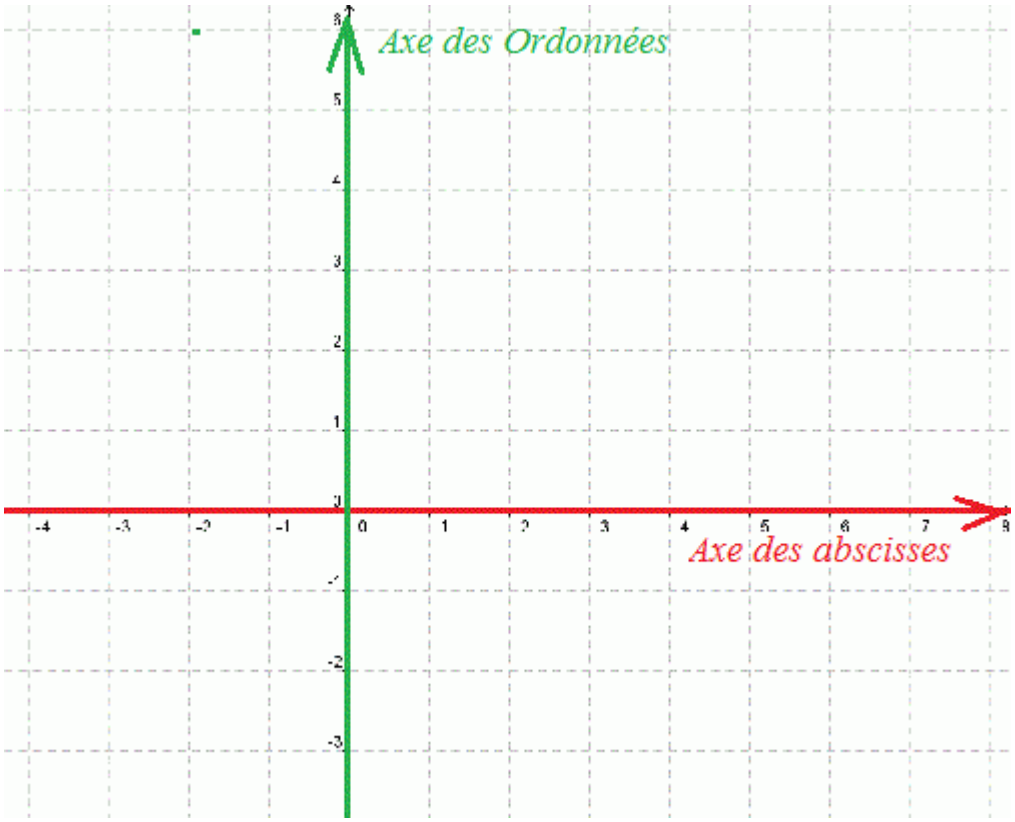
Par exemple : le point 4 d'ordonnée vient croiser le point d'abscisse 4 lorsque 2N = 32, 3= 0[3] et 32 = 2 [3] ce qui

donne une diagonale initialisée par **3 non congru à 32 [P]**

D'où l'axes des ordonnées de Goldbach initialisé par A = 3, représentera une diagonale non congrue à 2N [P], qui viendra croiser l'axe des abscisses Ératosthène afin de correspondre à l'égalité lorsque 2N augmente de 2: ***« 32 – 3 = 29 et 34 – 5 = 29 » Cette égalité est donc récurrente, lorsque 2N augmente de 2.***

3.) Autrement dit lorsque 2N augmente de 2, l'axe des ordonnées « *va descendre d'un rang* » sur l'axe des abscisses , ce qui décale d'un rang la diagonale des congruences sur cet axe d'abscisses, initialisée par 3 qui est non congrus à 2N+2 ; qui en venant se décaler d'un rang, croisera 5 sur l'axe des abscisses d'Ératosthène ; où A+2 = 5 est le successeur de 3 lorsque 2N augmente de 2, ce qui donnera comme complémentaire de 5 par rapport à 2N + 2 , la même différence , le contraire est impossible.

L'axe des ordonnées, représente les diagonales de congruences des entiers A impairs de : **[1.3.5.7.9.11....etc.. N]**



4.) Il n'est pas réaliste de supposer la conjecture fausse inférieur à une limite K, qui serait le point où, le nombre de solutions 2N = p' + q commenceraient à décroître si cette supposition serait réalisable au point, 2N + 2, + 4 + 6…..+**X**. Conséquence immédiate : le nombre de nombres premiers **q** par rapport à 1, appartenant à **[N , 2N]** chuterait pour atteindre pratiquement **0** au point **X ,** si les entiers **A sont congrus à 2N+X [P],** dans l'hypothèse d'une conjecture fausse, ce qui serait contraire au **TNP.**

Ce que l'on va voir avec cet algorithme : les congruences criblées par Goldbach , sont des diagonales de congruences, qui sont initialisées à la base par le nombre [1], ce nombre de diagonales congrues ou pas à 2N[P], tend vers l'infini,

Si le reste de 2N par P = **1**, on aura une diagonale d'entiers $A \equiv 2N[P]$ qui est marqué **[,0,]** ;
dans le cas contraire, si dans la division Euclidienne de 2N par P , le reste n'est pas égal à **1**, on initialise une diagonale de $A \not\equiv 2N[P]$ qui sont marqué **[,1,]**. Ce qui est illustré ci-dessous.

5.) Le crible de Goldbach représente toutes ces diagonales de congruences, lorsque l'on crible un entier de **1 à N** les congruences, permettent de donner le nombre de solutions qui vérifient **2N = p'+ q**

6.) il existera par conséquent une infinité de diagonales , soit congrues , soit non congrues, à **2N[P]** lorsque 2N tend vers l'infini , qui viendront , se décaler d'un rang sur l'axe des abscisses Ératosthène pour chaque 2N + 2.

Ce décalage récurrent d'un rang des congruences, permet de donner le nombre de solutions qui décomposera 2N + 2 = p + q, pour toute limite N criblée et de vérifier par là même plusieurs entiers pairs 2N+2 ,+4 ,+6...etc 2N +X , avec X < au nombre de rang de la limite N criblée.

7.) Par conséquent même si ***A n'est pas*** *un nombre premier ,* ***mais si*** *il précède* ***A+2*** *premier et qu'il est d'ordonnée* ***non congru à 2N [P] :***
Lorsque 2N augmente de 2 , soit 2N+2, on crée une nouvelle diagonale, l'axe d'ordonnées descend d'un rang , avec l'axe d'abscisses ce qui provoque le décalage d'un rang de la diagonale d'ordonnée sur cet axe d'abscisse et qui a pour antécédant ***(A) ≢ (2N) [P].***
D'où cette diagonale non congrue à 2N (mod P) et avec ***(A+2) ≢ (2N+2) [P] ⇒ ((2N+2) – (A+2) = q)*** *vérifiera donc la conjecture pour 2N+2 ...!*

8.) *Donc, p*our supposer la conjecture fausse , il faut qu'aucune diagonale $A \not\equiv 2N[P]$ avec ***A*** un nombre premier **p'≤ N** , ne vienne croiser ***p'*** un nombre premier sur l'axe des abscisses .
Autrement , dit il ne faut plus à partir d'une limite N = K de [,1,] non congrus à 2N modulo P.
(« Pour ce faire, il faudrait utiliser les restes R des limites précédente 2N ***– 2, – 4 ..etc,*** *ce qui est impossible. »)*

Lorsque 2N augmente de 2, *il faut aussi qu'aucune diagonale de A ≢ 2N[P] avec* ***A*** *premier ou pas , du point* ***7.)*** *ci-dessus relatif à 2N,* ***ne précède*** *une diagonale avec* ***A+2 = p'*** *un nombre premier* ***, qui viendrait valider la conjecture pour 2N+2, du fait de ce décalage récurrent d'un rang qui s'ensuit.!***
Il en serait ainsi de toutes les limites N – 1 , – 2 ,– 3 … etc.. précédentes, où aucune diagonale ne doit être non congrue à 2N (mod P) ; d'où, la fonction du TNP serait fausse , voir ci-après.

Ou encore que les restes de 2N par P soit toujours = 1 , qui donnerait que des 0 sur l'axe des ordonnées de Goldbach (" *mais c'est impossible, car cela entraîne la disparition des nombres premiers* ***q ∈ [N,2N]***") !

9.) Ces deux axes , « ne peuvent dans l'immédiat prouver » la conjecture de façon rigoureuse.

Mais : *Il est absolument formel de prévoir, à partir d'une limite N criblée et du nombre de solutions qui décomposent un entier pair* ***2N = p'+ q ;*** *le nombre* ***de solutions*** *qui vérifient les entiers suivants :* ***2N +2, + 4, + 6….+ X*** *, avec X inférieur au nombre de rangs de la limite* ***N qui vient d'être criblée.***
(« Illustré fin de page 4 à 5 pour N = 300 , 390 , 1140. »)

10.) **Ce que l'on sait** : l'axe des ordonnées qui initialise chaque ("diagonale") représentées par les entiers $A \not\equiv 2N[P]$ vaut ***N / Ln 2N*** *équivalent au nombre de nombres premiers* ***q ∈ [N;2N] où N = n, qui est comme la conjecture de Goldbach, une conséquence directe du TNP.***

****

La fonction 2 du théorème de Goldbach est une conséquence directe du TNP: (log = *logarithme naturel)*
G(n): la fonction de compte du nombre de nombres ***A ≢ 2n[P]*** ⇔ **q ∈ [n;2n]**

Corollaire du TNP : **G(n)** vaut $\lim_{n \to +\infty} \frac{n}{(\log 2n)}$

Le TNP dit que $\pi(n) = \frac{n}{(\log n)} + o(\frac{n}{\log n})$, donc le nombre de nombres premiers dans]*n*,2*n*] vaut

$$\pi(2n) - \pi(n) = \left(\frac{2n}{\log(2n)} - \frac{n}{\log n}\right) + o(\frac{n}{\log n})$$

$$= n \times \left(\frac{2}{\log 2n} - \frac{1}{\log n}\right) + o(\frac{n}{\log n})$$

$$= n \text{ x } \frac{2\log n - \log(2n)}{\log(2n)\log n} + o(\frac{n}{\log n})$$

$$= \frac{n}{(\log 2n)} + o\left(\frac{n}{\log n}\right)$$

Alors que celui des abscisses Ératosthène, relatif aux nombres premiers ***p'*** vaut vaut $\frac{n}{\log n}$

*L*a probabilité d'avoir une diagonale de **A** non congrue à ***2N (mod P)*** qui vérifie la conjecture est donc :

de $\frac{1}{(\log(2n)*\log n)}$ *Autrement dit : On peut admettre que le nombre de couples p+q qui décomposent*

l'entier 2N qui sert de limite de l'algorithme, vaut ≈ $\frac{n}{(\log(2n)*\log n)}$

Par exemple : Si on fixe la limite N = 91 , 2N = 182 , on aurait au minimum

$\frac{91}{(\log(182)*\log 91)}$ **= 3, 87653….** *couples p+q = 182 au lieu d'un réel de* ***6*** *couples.*

Si la conjecture était fausse , on aurait 12 nombres premiers q entre N et 2N au lieu de 18 et donc inférieur à la fonction du TNP qui en donne un équivalent de 17,4865…. pour cette limite 2N.
Alors que l'on sait, que pour la limite précédente (2N – 2) ce nombre de premiers q était identique à une unité près, ce qui serait absurde si la conjecture était supposée fausse.

Voila ..! Ce qu'il en est pour mon algorithme de Goldbach à l'heure actuelle.
La seule certitude, c'est que l'on peut montrer à partir d'une décomposition de Goldbach et des diagonales qui recoupent l'axe des abscisses, jusque où la conjecture est vérifiée sans avoir besoins de tester ou de cribler pour la limite suivante N+1, le nombre de solutions qui vérifie **2N + 2** .
Sachant que cette limite se repousse systématiquement pour tendre vers l'infini , que le nombre de nombres premiers vérifiés lors de la limite précédente [N – 1 , 2N – 2] et où le nombre de premiers ***q ∈ [N,2N] ne peut varier que de 1 entre ces deux intervalles : q ∈ [N+1 , 2N+2]*** on peut en conclure que cette conjecture ne peut être fausse, sans contredire le TNP ; ce qui serait impossible dans le cas contraire.

Voir Simulation page 5 .! $\mathscr{LG}$.

Exemple : on utilisera, **n** ou **N**

Secteur des diagonales d'entiers impairs, initialisée sur l'axe des ordonnées de **Goldbach** : par les congruence =[1,] ou [0,]
où à chaque augmentation de 1, de la limite **n** criblée , les diagonales de congruences se décalent d'un rang sur l'axe des abscisses qui descend d'un rang.

[0, 1, 0, 0, 1, 1, 0, 0, 1, 0
[1, 3, 5, 7 ,9 , 11, 13, 15, 17, 19 ,

illustration : *on va juste décaler la diagonale des congruences d'un rang lorsque N augmente de 1, donc N+2*

Donnez N: 20
[3, 5] <….**nombre P ≤ √2n**
1 <…..*reste r de 2n par P*
0

*crible: [0, **1**, 0, 0, **1,** **1**, 0, 0, **1**, 0] de 1 à 19 diagonales de Goldbach initialisée sur l'axe des ordonnées ?*
*[1, **3**, 5, 7 ,9 , **11**, 13, 15, **17**, 19] , axe d'abscisses d'Ératosthène*

Nombres non congru 2n[P] 1 à 20 premiers q de 20 à 40: 4 , 3 couples (p+q) = 40

Donnez N: 21
[3, 5] P
0
2
*crible: [**1,**[0, **1**, 0, 0, **1,** **1**, 0, 0, **1 ,** 0]*
[1, 3, **5**, **7** , 9 , **11**, **13**, 15, 17, **19** ,21]
Nombres non congru 2n[P] 1 à 21 premiers q de 21 à 42: 5 ; 4 couples (p+q) = 42

*Donnez **N: 22***
[3, 5] P
2
4
crible:
*[**1, 1**,[0, **1**, 0, 0, **1**, **1**, 0, 0, **1**]*
[1, 3, **5**, **7** , 9 , **11**, 13, 15, **17**, **19** ,21]
Nombres non congru 2n[P] 1 à 22 premiers q de 22 à 44: 6 ;
***3 couples p+q** = 44 sont encore vérifiés , sans compter les deux nouveaux couples avec 5 et 7, ; le but est de montrer la décalage de l'axe des ordonnées sur l'axe des abscisses,à chaque fois que 2n augmente de 2.*

Donnez N: 23
[3, 5]
1
1
crible
*[0, 1, 1, [0, **1**, 0, 0, 1, **1**, 0, 0 , **1**]*
[1, 3, **5**, [**7** , 9 , [11, **13**, 15, **17**, **19** ,21, **23**]

Donnez N: 24
[3, 5]
0
3
crible:
*[1, 0, 1, 1, [0, **1**, 0, 0, **1**, **1**, 0 , 0 , 1]*
[1, 3, **5**, **7** ,[9 , **11**, 13, 15, **17**, **19** , 21 , **23**]

Congruences de Goldbach , entiers A impairs modulo 2:

Nombre premiers **p'** ≤ N où 1 n'est pas un nombre premier, Axe des abscisses d'Ératosthène,
Entiers A impairs:
[1, **3**, **5, 7**, 9, **11, 13**, 15, **17, 19**, 21, **23**, 25, 27, **29, 31**, 35, **37**, 39, **41 43**, 45, **47**, 49, 51**,]** *53, 59, 61, 67, 71, 73, 79, 83, 89, 97**]***

[ce qui se traduit par l'axe d'abscisses des nombres premiers suivant] **: n ≤ 30**
[1, 1, 1, 1, 0**, 1, 1,** 0**, 1, 1,** 0**, 1,** 0, 0**, 1] 15** ***p' = 1,** ou multiples de P = 0*

N point k, de 15 à X conjecture fausse par supposition,

A : impairs, **congrus** = 0 , **non congrus** =1

[entiers impairs de **1 à n**] et ils sont ***indicé de 0 à n.*** On calcule le reste **R** de 2n = 30 par **P ≤ √2n**
P = 3 , 5 ; **R** = 0 , 0 :

*Si **R**%2= 0 on fait (**R+P**) // 2 et **on part de l'indice** en le marquant 0 , ainsi que tous ses suivants modulo P ,*
*si **R**%2 != 0 , **R**//2 on part de l'indice que l'on marque 0 , ainsi que tous ses suivants modulo P.*
*On aura marqué **[,0,]** tous les entiers **congrus à 2n [P]**, à la fin on relève les **[,1,]** qui sont les entiers **non congrus à 2n[P]***

Illustration :

ordonnées
↓
↓
[1, 3, 5, 7, 9,11,13] → → abscisses
[*secteur des diagonales de congruences qui montre le décalage pour 2n +2:*
*représentés par les A impairs **congru** = 0* **sinon = 1**] en partant de l'axe d'ordonnées initialisé par la cellule du nombre **[1,** et ce , pour tout 2n +2 , qui initialise donc une diagonale de congruences , ***par le crible de Goldbach***

Ordonnées, de l'algorithme de Goldbach, chaque diagonale représente les A impairs en progression arithmétique de raison 2.
Crible G
↓
↓
↓
[1], 0, 0, 1, 0, 1, 1,] **15**]n =15 // 2 = 7 +1 entiers impairs de 1 à 15
[**1**], **1**, **0**, **0**, 1, 0, 1, 1,] 16 … P = 3 et 5 , R = 2 et 2 ; indice (2+3)//2 = **2** et (2+5)//2 = **3 ;** *ensuite par pas de 3 et de 5*
[**0**], 1, **1**, 0, **0**, 1, 0, 1,] **17**] … P = 3 et 5 , R = 1 et 4 ; indice (1)//2 = **0** et (4+5)//2 = **4**
[**0**], **0**, 1, **1**, 0, 0, 1, 0, 1,] 18 … P = 3 et 5 , R = 0 et 1 ; indice (0+3)//2 = **1** et (1)//2 = **0**
[**1**], 0, 0, 1, **1**, 0, 0, 1, 0,] **19**] s,
[**0**], 1, 0, 0, 1, **1**, 0, 0, 1, 0,] 20
[**1**], 0, 1, 0, 0, 1, **1**, 0, 0, 1,] **21**]
[**1**], 1, 0, 1, 0, 0, 1, **1**, 0, 0, 1,] 22
[**0**], 1, 1, 0, 1, 0, 0, 1, **1**, 0, 0,] **23**]
[**1**], 0, 1, 1, 0, 1, 0, 0, 1, **1**, 0, 0,] 24
[**0**], 1, 0, 1, 1, 0, 1, 0, 0, 1, **1**, 0,] **25**]
[**0**], 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, **1**, 0,] 26
[**1**], 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, **1**,] **27**]
[**0**], 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, **1**,] **28 P.**
[**0**], 0, **1**, 0, 0, **1**, 0, **1**, **1**, 0, 1, 0, 0, 1, **1**] **29**] … 7 nombres premiers ***q*** ∈ *[N,2N].(« Si la conjecture était fausse il faudrait supprimer 4 nombres premiers q, ainsi que pour les limites précédentes N =28 , 27 , 26… etc. Ce qui est impossible car déjà vérifiées lors de ces limites précédentes .!*

[**1**], 1, **1,** 1, 0, **1,** 1, **0, 1,** 1, **0,** 1, 0, **0, 1,]31] → → → abscisses < N des nombres premiers p' Ératosthène**

crible **É** et **G** : [0, 0, **1**, 0, 0, **1,** 0, 0, **1**, 0, 0, 0, 0, 0]
Nombres p' non congru 2n[P] ou couple P'+q = 2N, de (i) à 29 : **3**

Simulation et illustration des diagonales de congruences ,Goldbach modulo →30, en progression arithmétique de raison 30 de premier terme 7
à partir d'une limite n = 300 , qui a vérifié tous les entiers pairs < 600 ,

Simulation et vérification en utilisant simplement la famille **30k + 7** et les nombres **P ≤ √2n**
les entiers pairs 2N, seront vérifiés par les diagonales de congruence → 2N = y = 780 ,
on **repousse la limite de 6**

Secteur gradué de : **1** = entier **non congrus** à **2n[P]** ; 0 = entier **congru à 2n[P]**

Donnez **N: 300** , 600 →; [630; 660; 690; 720; 750; 780] y = 780

secteur des congruences représentant les entiers non congru =**1**,ou congrus à 2n[P] = 0

[**1, 1, 1, 1, 1, 1**, 0, 0, 0, **1**] diagonales des congruences initialisées par Goldbach
[**7,37,67,97,127,157**,187, 217, 247, **277]** → *Abscisses* → ∞

on décale d'un rang, les congruences pour tout 2N+2 :

crible G: [**1**, 1, 0, 1, 0, 1, 0, 1, 1, 0] stop *fin du décalage des diagonales vérifiant la conjecture jusqu'à 2n = 780* ,
crible E: [**1, 1, 1, 1, 1, 1**, 0,] 0, 0, **1**] ⇒ entiers d'Ératosthène = [**7,37,67,97127,157**,187, 217, 247,**277]**

En vérifiant pour **N = 390** , les entiers pairs < 2N = 780 , entiers pairs vérifiée → y = 1140 on **repousse la limite de 12**

On réitère à partir de N = 390 dernière limite N vérifiée.

Donnez **N: 390**, **2N = 780** → 810, 840, 870, 900, 930, 960, 990 , 1020 , 1050 ...1080 ,1110 , 1140 , y = 1140
crible G: [[**1**, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0]
crible E: [1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, **1**]

Donnez **N: 1140**, 2N = 1140 → entiers pairs vérifiés jusqu'à y = (1140 +1110) **on repousse la limites de 37** , quasiment le double , C'est à dire : que lorsque l'on vérifie les entiers pairs inférieur à une limite 2n fixée , on vérifie par la même tous les entiers pairs qui seront vérifié **<** à (2n/30)*2 par le décalage des diagonales de congruences .

Crible G, Diagonales des congruence qui se sont décalées pour tout 2n + ...2 :
,, [**1**, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1]

crible E: [1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, **1**].

n=2250 , 2250/30 = 37*2 = 74 et **74*2** → donne la prochaine limite N= 3840 et tous les entiers pairs < 7680 sont vérifiés
G :[[1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1]

É :[1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0]

Famille 7 modulo 30 :
Diagonales des congruences en progression arithmétique de raison 30

on peut vérifier avec Ératosthène à quel moment la conjecture devient fausse, c'est à dire lorsque l'axe des ordonnées des congruence [0] ne croise plus un nombre premier d'Ératosthène sur l'axe des abscisses à la ligne Ln+1 = point X

crible: des diagonales, non congru = 1, sinon 0
↓ axe des ordonnées
↓
(**7 ,37 ,67 ,97 127 ,157** ,187] →→ *axe des abscisses, avec A modulo 30.*
↓
1[0, 1, 1, 0, 1, 1, 1] → n=210 ; **2n = 420** [« *l'algorithme progresse donc modulo 15(k), diagonale initialisée par 7,*
2[**1**, 0, 1, 1, 0, 1, 1] n=225 [« *pour tout 2n + 2,* on réinitialise une diagonale de congruences
3[0, **1**, 0, 1, 1, 0, 1, 1]
4[**1**, 0, **1**, 0, 1, 1, 0, 1]
[0, **1**, 0, **1**, 0, 1, 1, 0, 1]
: [1, 0, **1**, 0, **1**, 0, 1, 1, 0]
: [1, 1, 0, **1**, 0, **1**, 0, 1, 1, 0] n=300
: [0, 1, 1, 0, **1**, 0, **1**, 0, 1, 1]
: [1, 0, 1, 1, 0, **1**, 0, **1**, 0, 1, 1]
: [1, 1, 0, 1, 1, 0, **1**, 0, **1**, 0, 1]

: [0, 1, 1, 0, 1, 1, 0, **1**, 0, **1**, 0, 1]
: [1, 0, 1, 1, 0, 1, 1, 0, **1**, 0, **1**, 0] 375
: [1, 1, 0, 1, 1, 0, 1, 1, 0, **1**, 0, **1**, 0] 390
: [0, 1, 1, 0, 1, 1, 0, 1, 1, 0, **1**, 0, **1**]
: [0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, **1**, 0, **1**]420
: [1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, **1**, 0]435
: [0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, **1**, 0] 450
: [0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, **1**] 465
: [1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, **1**]
: [1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0]
: [1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0]510
: [0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1]525
: [**0**, **0**, **1, 1, 1**, **0**, 0, 1, 0, **0, 1, 1, 0, 1**, 1, **0, 1**, 1] **X = 540 conjecture vraie ,** (avec A = 7, il suffit de décaler d'un rang ces congruences sur l'axe d'abscisse d'Ératosthène ci-dessous , pour vérifier jusqu'où la conjecture est vraie ...

[0, 0, 1, 1, 1, 0, 0, 1**, 0, 0, 1, 1, 0,** 1, **1, 0, 1,** 1] **X = 555** (*avec A = 37 et qui augmente de 30 pour tout X=15(k+1)*

:[1, 1**,** 1**, 1, 1, 1**, 0**,** 0**, 0**, 1, 1**, 1, 1, 1**, 1**, 0, 1**, 1**, 0]** Ératosthène ; p' < 15k = 540 , Fam 30k + 7**

On comprend pourquoi cet effet boule de neige, augmente le nombre de solutions qui tend vers l'infini :
lorsque **N** → ∞ **.**
Mais aussi pourquoi si elle était fausse, le nombre de nombre premiers ***q*** ∈ *[N,2N] serait faux , alors qu'il a été vérifié lors des limites N précédentes .!*

****

Crible G, entiers A = impairs de 1 à N modulo 2, N = 600

[0, 0, 0, **1**, 0, 0, **1**, 0, 0, **1**, 0, 0, 0, 0, **1**, 0, 0, 0, **1**, 0, 0, 0, 0, **1, 1**, 0, 0, 0, 0, 0, 0, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, **1**, 0, 0, 0, 1, 0, 0,
[1, **1, 1, 1**, 0, **1, 1**, 0, **1, 1**, 0, **1**, 0, 0, **1, 1**, 0, 0, **1**, 0, **1, 1**, 0, **1,** 0, 0, **1**, 0, 0, **1, 1**, 0, 0, **1**, 0, **1, 1**, 0, 0, **1**, 0, **1**, 0, 0, **1**, 0, 0, 0,

1, 0, 0, **1**, 0, **1, 1**, 0, **1**, 0, 0, 0, 0, 0, 0, 0, 0, **1**, 0, 0, **1, 1**, 0, 0, 0, 0, **1, 1**, 0, 0, 0, 0, **1**, 0, 0, **1, 1,** 0, 0, 0, 0, **1, 1**, 0, 0, **1**, 0, **1**,
1, 0, **1, 1**, 0, **1, 1**, 0, **1**, 0, 0, 0, 0, 0, 0, **1**, 0, **1,** 0, 0, **1, 1,** 0, 0, 0, 0, **1, 1**, 0, 0, **1,** 0, 0, **1**, 0, **1**, 0, 0, **1**, 0, 0, **1, 1**, 0, 0, 0, 0, **1**,

0, 0, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, **1**, 0, **1**, 0, 0, 0, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, **1**, 0, **1**, 0, 0, 0, 0, **1**, 0, 0, 0, 0, **1**, 0, 0, 0,
1, 0, **1, 1,** 0, 0, 0, 0, 0, **1**, 0, 0, 0, 0, 0, **1**, 0, **1, 1**, 0, **1,** 0, 0, **1, 1**, 0, 0, 0, 0, 0, **1,** 0, 0, **1**, 0, 0, **1**, 0, **1**, 0, 0, **1**, 0, 0, **1, 1,** 0, 0, **1**, 0, **1, 1**, 0, 0,

1, 0, **1,** 0, 0, 0, 0, 0, 0, 0, 0, 0, **1,** 0, **1, 1**, 0, **1**, 0, 0, 0, 0, 0, 0, **1**, 0, **1, 1**, 0, **1**, 0, 0, 0, 0, 0, 0, **1**, 0, 0, 0, 0, **1, 1**, 0, **1, 1**, 0, 0,
0, 0, **1**, 0, 0, 0, 0, 0, 0, **1**, 0, **1, 1**, 0, **1,** 0, 0, 0, 0, 0, 0, **1,** 0, 0, **1**, 0, 0, 0, 0, **1, 1**, 0, **1**, 0, 0, **1**, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, **1**, 0, **1**,

0, 0, **1, 1**, 0, 0, 0, 0, 0, **1,** 0, 0, 0, 0, 0, **1**, 0, 0, 0, 0, 0, 0, **1**, 0, **1**, 0, 0, 0, **1**, 0, **1**, 0, 0, **1**, 0, 0, 0, **1**, 0, **1,** 0, 0, **1**, 0, 0, **1,** 0, 0, 0,
0, 0, **1**, 0, 0, 0, **1**, 0, **1,** 0, 0, 0, **1**, 0, 0, 0, 0, 0, **1, 1**, 0, 0, 0, 0, 0, **1, 1**, 0, 0, **1,** 0, **1,** 0, 0, **1,** 0, 0, 0, **1**, 0, **1, 1**, 0, **1**, 0, 0, 0, 0, 0, 0, **1,**

1, 0, 0, 0, 0, 0, **1**, 0, 0, 0, **1**, 0, 0, 0, 0, **1**, 0, 0, 0, **1**, 0, 0, **1**, 0, **1**, 0, 0, 0, 0, 0, 0, **1, 1**, 0, 0, **1**, 0, 0, **1**, 0, **1, 1**, 0, 0, 0, 0, 0, **1**, 0, 0, 0,
0, 0, 0, **1,** 0, **1**, 0, 0, 0, 0, **1**, 0, **1**, 0, 0, **1,** 0, 0, 0, 0, 0, 0, **1, 1**, 0, 0, 0, 0, 0, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, **1, 1,** 0, 0,

0, 0, **1, 1**, 0, **1,** 0, 0, **1**, 0, 0, **1, 1**]
1, 0, 0, 0, 0, **1,** 0, 0, **1,** 0, 0, **1**]

Crible E

[1, **1, 1, 1**, 0, **1, 1**, 0, **1, 1**, 0, **1**, 0, 0, **1, 1**, 0, 0, **1**, 0, **1, 1**, 0, **1,** 0, 0, **1**, 0, 0, **1, 1**, 0, 0, **1**, 0, **1, 1**, 0, 0, **1**, 0, **1**, 0, 0, **1**, 0, 0, 0,
1, 0, **1, 1**, 0, **1, 1**, 0, **1**, 0, 0, 0, 0, 0, 0, **1**, 0, **1,** 0, 0, **1, 1,** 0, 0, 0, 0, **1, 1**, 0, 0, **1,** 0, 0, **1**, 0, **1**, 0, 0, **1**, 0, 0, **1, 1**, 0, 0, 0, 0, **1**,
1, 0, **1, 1,** 0, 0, 0, 0, 0, **1**, 0, 0, 0, 0, 0, **1**, 0, **1, 1**, 0, **1,** 0, 0, **1, 1**, 0, 0, 0, 0, **1,** 0, 0, **1**, 0, 0, **1**, 0, 0, **1, 1**, 0, 0, **1**, 0, **1, 1**, 0, 0,
0, 0, **1**, 0, 0, 0, 0, 0, 0, **1**, 0, **1, 1**, 0, **1,** 0, 0, 0, 0, 0, 0, **1,** 0, 0, **1**, 0, 0, 0, 0, **1, 1**, 0, **1**, 0, 0, **1**, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, **1**, 0, **1**,
0, 0, **1**, 0, 0, 0, **1**, 0, **1,** 0, 0, 0, **1**, 0, 0, 0, 0, 0, **1, 1**, 0, 0, 0, 0, 0, **1, 1**, 0, 0, **1,** 0, **1,** 0, 0, **1,** 0, 0, 0, **1**, 0, **1, 1**, 0, **1**, 0, 0, 0, 0, 0, 0, **1,**
0, 0, 0, **1,** 0, **1**, 0, 0, 0, **1**, 0, **1**, 0, 0, **1,** 0, 0, 0, 0, 0, 0, **1, 1**, 0, 0, 0, 0, 0, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, **1, 1,** 0, 0,
1, 0, 0, 0, 0, **1,** 0, 0, **1,** 0, 0, **1**]

crible G N = 620
[0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1,[**0, 0, 0**, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1,
1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1,
1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0,
0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1]

Crible E
[1[, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0,[**1, 1, 0**, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0,
1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0,
1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, [*1, 0, 0, 1, 0, 0, 1, 0, 1, 1*]

Crible E et G ,N = 620 , **2N = 1240**
[0, **1**, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, **1**, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, **1**, 0, 0, **1**, 0, 0, **1**, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, **1**,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Nombres p' non congru 2n[P] ou couple P'+ q = 2N, de (i) à 620 famille 1 : 33 ----- 0.0

La répartition des nombres premiers et du TNP, est une conséquences directe d'une conjecture de Goldbach vraie et non l'inverse . !

En fonction de la forme de 2N on choisi la Fam 30k + i :

Annexe 1

En fonction de la limite **N = 15k** fixée, on fixe lune des **8 Fam (i)** correspondante à la forme de **N**

Pour **N** de la forme **15k**, on peut choisir n'importe la quelle des 8 Fam.

Le programme est fixé avec une limite **N** début = 3000000000 et Fin = 3000000330 il progressera de raison 15,
Il n'y a que la fam(i) à rentrer à la demande:

Pour toute Limite **N = 15k + n'**, avec **n' ∈ [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14]** on rentre la Fam(i) correspondante

N = 15k+1; Fam =(1,13,19); **2n = 30k + 2**
N = 15k+2; Fam =(11,17,23); **2n = 30k + 4**
N = 15k+3; Fam =(7,29,13,23,17,19); **2n = 30k + 6**
N = 15k+4; Fam =(1,7,19); 2n = **30k + 8**
N = 15k+5; Fam =(1,7,13,19); **2n = 30k + 10**
N = 15k+6; Fam =(1,11,13,19,23,29); **2n = 30k + 12**
N = 15k+7; Fam =(1,7,13); 2n = **30k + 14**
N = 15k+8; Fam =(17,23,29); 2n = **30k + 16**
N = 15k+ 9; Fam =(1,7,11,17,19,29); 2n = **30k + 18**
N = 15k+10; Fam =(11,29,17,23); 2n = **30k + 20**
N = 15k+11; Fam =(11,23,29); **2n = 30k + 22**
N = 15k+ 12; Fam =(1,7,11,13,17,23); 2n = **30k + 24**
N = 15k+ 13; Fam =(7,13,19); 2n = **30k + 26**
N = 15k+ 14; Fam =(11,17,29); 2n = **30k + 28**

Pour les multiples de 30, avec N = 15k; ***l'une des 8 Fam ,*** ***2n = 30k***

programme ci-joins, python modulo 2 utilisé :

Programme Python (fam 1 **modulo 2):**

```
from time import time
from os import system
import math

def candidate_range(n):
    cur = 5
    incr = 2
    while cur < n+1:
        yield cur
        cur += incr
        incr ^= 6  # or incr = 6-incr, or however


def eratostene(n):
    n = int((2*n)**0.5)
    prime_list = [3]
    sieve_list = [True] * (n+1)
    for each_number in candidate_range(n):
        if sieve_list[each_number]:
            prime_list.append(each_number)
            for multiple in range(each_number*each_number, n+1, each_number):
                sieve_list[multiple] = False
    print(prime_list)
    return prime_list[0:]


def demander_N():
    n = input("Donnez N: ")
    n = int(n.strip().replace(" ", ""))

    return n


def G_crible(premiers, n, fam):
    start_crible = time()
    crible = (n//2)*[1]     ## Ou: on rappel le tableau Ératosthène criblé de N/2 cases
    lencrible = len(crible)

    ## On calcule les restes: R = 2*n modulo P
    nbpremiers = len(premiers)
    n2 = 2*n

    for i, premier in enumerate(premiers):
        reste = n2 % premier
        #print(reste)
        if reste % 2 == 0:
            reste += premier
        pi2 = 2*premier
        while reste % 2 != fam:   ## tant que R % 2 != fam on fait R+= 2P
            reste += pi2
                        ## Ensuite on divise R par 2 pour obtenir l'indexe , indice.
```

```
        reste //= 2
                        ## On crible directement à partir du n° indice l'index que l'on marque 0
        for index in range(reste, lencrible, premier):
            crible[index] = 0

    total = sum(crible)
    print("crible:", crible)
    print(f"Nombres non congru 2n[P] de {1} à {n} famille {fam} nbr premiers q de {n} à {n2}: {total} -----
{int((time()-start_crible)*100)/100}")


def main():
    ## On demande N a l'utilisateur
    n = demander_N()

    ## On récupère les premiers de 3 à √2N
    premiers = eratostene(n)
    #lprint("premiers:", premiers)
    #print(f"nombres premiers de3 à {int((2*n)**0.5)}: {len(premiers)}")

    start_time = time()
    ## On crible
    G_crible(premiers, n, 1)
    temps = time()-start_time
    print(f"--- Temps total: {int(temps*100)/100} sec ---")


main()
system("pause")
```

Programme Python (fam 1 **modulo 2) [Ératosthène et Goldbach unifié]:**
Attention j'ai repris: l'algorithme par famille 30k + i ; modulo 30 pour le transformer en crible modulo 2 , afin de l'utiliser dans les entiers A impairs de premier terme 1 en progression arithmétique de raison 2 , uniquement pour les petite valeurs afin d'illustrer les commentaires ci-dessus.

```
from time import time
from os import system

def candidate_range(n):
    cur = 5
    incr = 2
    while cur < n+1:
        yield cur
        cur += incr
        incr ^= 6  # or incr = 6-incr, or however

def eratostene(n):
    n = int((2*n)**0.5)  ##(si on fusionne les deux cribles il faudra rentrer, int((2n)**0.5) pour Goldbach.
    prime_E =[3]
    sieve_list = [True for _ in range(n+1)] ## c'est plus propre comme ça
    for each_number in candidate_range(n):
        if sieve_list[each_number]:
            prime_E.append(each_number)
            for multiple in range(each_number*each_number, n+1, each_number):
                sieve_list[multiple] = False
    print(prime_E[0:])
    return prime_E[0:]
```

```
def E_Crible(premiers, n, fam):
    start_crible = time()

    ## On génère un tableau de N/2 cases rempli de 1
    lencrible = ((n//2)*1)
    crible=[1 for _ in range(lencrible)] ## c'est plus propre comme ça
    GM = [3,5,7,11,13,17,19,23,29,31]   ## attention GM > 5 est utilisé pour l'algorithme modulo 30 par fam 30k +
i
    # On calcule les produits :
    for a in premiers:
        for b in GM:
            j = a * b
            if j%2 == fam:
                index = j // 2  ## Je calcule l'index n° indice et On crible directement à partir du n° indice
                for idx in range(index, lencrible, a):  ## index qui est réutilisé ici...
                    crible[idx] = 0

    total = sum(crible)-1
    print("crible Ératosthène :", crible)  ## pour éditer le tableau Ératosthène criblé
    print(f"Nombre premiers criblés >2, famille {fam} : {total} ----- {int((time()-start_crible)*100)/100}")
    return crible,lencrible

def GCrible_2N(premiers, crible, lencrible, n, fam):
    start_crible = time()
    ## On calcule les restes: R = 2*n modulo P
    nbpremiers = len(premiers)
    n2 = 2*n
    for premier in premiers:
        reste = n2 % premier
        print(reste)
        if reste % 2 == 0:
            reste += premier
        pi2 = 2*premier
        ## tant que reste % 2 != fam on fait reste += pi2 , pi = P(int((2*n)**0.5))
        while reste % 2 != fam:
            reste += pi2
        ## Ensuite on divise reste par 2 pour obtenir l'index
        reste //= 2
        ## On crible directement à partir de l'index
        for index in range(reste, lencrible, premier):
            crible[index] = 0

    total = sum(crible)-1  ## car 1 n'est pas premier donc il n'est pas un couple p'+ q
    print("crible É ET G:", crible)  ## éditer le tableau criblé É et G
    print(f"Nombres p' non congru 2n[P] ou couple P'+q = 2N, de (i) à {n} famille {fam} : {total} ----- {int((time()-
start_crible)*100)/100}")

def demander_N():
    n = input("Donnez N: ")
    n = int(n.strip().replace(" ", ""))
    #n = int(30 * round(float(n)/30))
    return n

def main():
    ## On demande N a l'utilisateur
    n = demander_N()
    ## On récupère les premiers de 7 à √2N
    premiers = eratostene(n)
    start_time = time()
    ## On crible
```

```
    fam=1 ## ou (1, 7, 11, 13, 17, 19, 23, 29, utilisé par famille 30k + i au choix en fonction de n)
    crible,lencrible=E_Crible(premiers, n, fam)
    GCrible_2N(premiers, crible, lencrible, n, fam)

main()
system("pause")
```

****

Programme Python Goldbach : Par Famille 30k + i , déjà réglé sur la famille 30k +7 **,** avec i ∈ [1,7,11,13,17,19,23,29]

```
from time import time
from os import system
import math

def candidate_range(n):
    cur = 5
    incr = 2
    while cur < n+1:
        yield cur
        cur += incr
        incr ^= 6  # or incr = 6-incr, or however


def eratostene(n):
    n = int((2*n)**0.5)
    prime_list = [2, 3]
    sieve_list = [True] * (n+1)
    for each_number in candidate_range(n):
        if sieve_list[each_number]:
            prime_list.append(each_number)
            for multiple in range(each_number*each_number, n+1, each_number):
                sieve_list[multiple] = False
    #print(prime_list[3:])
    return prime_list[3:]


def demander_N():
    n = input("Donnez N: ")
    n = int(n.strip().replace(" ", ""))
    #n = int(30 * round(float(n)/30))
    return n


def GCrible(premiers, n, fam):
    start_crible = time()
    crible = (n//30)*[1]     # Ou: on rapelle le tableau Ératosthène criblé de N/30 cases
    lencrible = len(crible)

    # On calcule les restes: ri = 2*n modulo P
    nbpremiers = len(premiers)
    n2 = 2*n

    for i, premier in enumerate(premiers):
        reste = n2 % premier
```

```
        #print(reste)
            # tant que ri % 30 != fam on fait R+= 2P
        if reste % 2 == 0:
            reste += premier
        pi2 = 2*premier
        while reste % 30 != fam:
            reste += pi2
        # Ensuite on divise R par 30 pour obtenir l'indexe, indice
        reste //= 30
        # On crible directement à partir du n°d'indice avec P où on remplace le 1 par 0
        for index in range(reste, lencrible, premier):
            crible[index] = 0

    total = sum(crible)
    print("crible:", crible)
    print(f"Nombres non congru 2n[pi] {1} à {n} famille {fam} premiers de {n} à {n2}: {total} ----- {int((time()-
start_crible)*100)/100}")


def main():
    # On demande N a l'utilisateur
    n = demander_N()

    # On récupère les premiers entre 7 et √2N
    premiers = eratostene(n)
    #lprint("premiers:", premiers)
    #print(f"nombres premiers entre 7 et {int((2*n)**0.5)}: {len(premiers)}")

    start_time = time()
    # On crible
    GCrible(premiers, n, 7)
    temps = time()-start_time
    print(f"--- Temps total: {int(temps*100)/100} sec ---")


main()
system("pause")
```

Programme Python Ératosthène : *déjà réglé sur la famille 30k +7 , les deux familles doivent toujours être les mêmes, car on crible Goldbach en utilisant les congruences, ce qui évite de s'occuper de la famille d'Ératosthène complémentaire de p' premiers ≤ n, qui est donc sans intérêt.*

lorsque 7 est congru à 2n modulo 30 , on sait très bien que q = 23 [30], autrement dit 60 – 7= 53 =23[30] , attention pour la famille 1 [30] , 1 n'est pas premier donc enlever 1 au résultat final.

Ératosthène :

```
from itertools import product
from time import time
from os import system
import math

def candidate_range(n):
    cur = 5
    incr = 2
```

```
    while cur < n+1:
        yield cur
        cur += incr
        incr ^= 6  # or incr = 6-incr, or however


def eratostene(n):
    n = int(n**0.5)  ##(si on fusionne les deux cribles il faudra rentrer, int((2n)**0.5) pour Goldbach.
    prime_list =[2,3]
    sieve_list = [True] * (n+1)
    for each_number in candidate_range(n):
        if sieve_list[each_number]:
            prime_list.append(each_number)
            for multiple in range(each_number*each_number, n+1, each_number):
                sieve_list[multiple] = False
    #print(prime_list[3:])
    return prime_list[3:]


def demander_N():
    n = input("Donnez N: ")
    n = int(n.strip().replace(" ", ""))
    #n = int(30 * round(float(n)/30))
    return n


def E_Crible(premiers, n, fam):
    start_crible = time()

    ## On génère un tableau de N/30 cases rempli de 1
    crible = n//30*[1]
    lencrible = len(crible)
    GM = [7,11,13,17,19,23,29,31]
    ## On calcule les produits: j = a * b

    for a in premiers:
        for b in GM:
            j = a * b
            if j%30 == fam:
                index = j // 30  ## Je calcule l'index et On crible directement à partir de l'index, du n° d'indice
        for idx in range(index, lencrible, a):  # index qui est réutilisé ici...
            crible[idx] = 0
        #print(index)

    total = sum(crible)  ## à la place, pour utiliser le tableau d'Ératosthène criblé dans le crible de Goldbach, on re-
turn "crible:", crible
    print("crible:", crible)
    print(f"Nombre premiers criblés famille {fam} : {total} ----- {int((time()-start_crible)*100)/100}")


def main():
    ## On demande N a l'utilisateur
    n = demander_N()

    ## On récupère les premiers de 7 à √N
    premiers = eratostene(n)
    #print(f"nombres premiers entre 7 et n: {len(premiers)}")

    start_time = time()
    ## On crible
```

```
    E_Crible(premiers, n, 7)  ## au choix (1,7,11,13,17,19,23,29)
    temps = time()-start_time
    print(f"--- Temps total: {int(temps*100)/100} sec ---")


main()
system("pause")
```

Programme unifié Goldbach et Ératosthène , qui donne le résultat du nombre de décompositions pour tout entier pair 2N = 30k + i et pour toutes limite N = 15k ≥ 150.

```
from time import time
from os import system

def candidate_range(n):
    cur = 5
    incr = 2
    while cur < n+1:
        yield cur
        cur += incr
        incr ^= 6  # or incr = 6-incr, or however

def eratostene(n):
    n = int((2*n)**0.5)  ##(si on fusionne les deux cribles il faudra rentrer, int((2n)**0.5) pour Goldbach.
    prime_E =[2,3]
    sieve_list = [True for _ in range(n+1)] ## c'est plus propre comme ça
    for each_number in candidate_range(n):
        if sieve_list[each_number]:
            prime_E.append(each_number)
            for multiple in range(each_number*each_number, n+1, each_number):
                sieve_list[multiple] = False
    print(prime_E[3:])
    return prime_E[3:]

def E_Crible(premiers, n, fam):
    start_crible = time()

    # On génère un tableau de N/30 cases rempli de 1
    lencrible = ((n//30)+1)
    crible=[1 for _ in range(lencrible)] ## c'est plus propre comme ça
    GM = [7,11,13,17,19,23,29,31]
    # On calcule les produits :
    for a in premiers:
        for b in GM:
            j = a * b
            if j%30 == fam:
                index = j // 30  ## Je calcule l'index et On crible directement à partir de l'index
                for idx in range(index, lencrible, a):  ## index qui est réutilisé ici...
                    crible[idx] = 0

    total = sum(crible)
    print("crible Ératosthène :", crible)  ## pour éditer le tableau Ératosthène criblé
    print(f"Nombre premiers criblés famille {fam} : {total} ----- {int((time()-start_crible)*100)/100}")
    return crible,lencrible

def GCrible_2N(premiers, crible, lencrible, n, fam):
```

```
    start_crible = time()
    # On calcule les restes: ri = 2*n/pi
    nbpremiers = len(premiers)
    n2 = 2*n
    for premier in premiers:
        reste = n2 % premier
        #print(reste)
        if reste % 2 == 0:
            reste += premier
        pi2 = 2*premier
       ## tant que reste % 30 != fam on fait reste += pi2
        while reste % 30 != fam:
            reste += pi2
        ## Ensuite on divise reste par 30 pour obtenir l'index
        reste //= 30
        ## On crible directement à partir de l'index
        for index in range(reste, lencrible, premier):
            crible[index] = 0

    total = sum(crible)
    print("crible É ET G:", crible) ## éditer le tableau criblé É et G
    print(f"Nombres P' non congru 2n[pi] ou couple P'+q = 2N, de (i) à {n} famille {fam} : {total} -----
{int((time()-start_crible)*100)/100}")

def demander_N():
    n = input("Donnez N: ")
    n = int(n.strip().replace(" ", ""))
    #n = int(30 * round(float(n)/30))
    return n

def main():
    ## On demande N a l'utilisateur
    n = demander_N()
    ## On récupère les premiers de 7 à √2N
    premiers = eratostene(n)
    start_time = time()
    ## On crible
    fam=7 ## ou 1, 7, 11, 13, 17, 19, 23, 29, au choix en fonction de n
    crible,lencrible=E_Crible(premiers, n, fam)
    GCrible_2N(premiers, crible, lencrible, n, fam)

main()
system("pause")
```