Introduction:

[« suivant l'affirmation de Wikipédia sur cette conjecture de Goldbach :

« Le théorème des nombres premiers affirme qu'un entier ***m*** sélectionné aléatoirement d'une manière brute possède (**1/Ln m**) chance d'être premier. Ainsi, si ***n*** est un grand entier pair et ***m***, un nombre compris entre **3** et ***n*/2**, alors on peut s'attendre à ce que la probabilité que ***m*** et ***n* – *m*** soient tous deux premiers soit égale à **1 /** (**Ln n . Ln m**) . Cet argument heuristique n'est pas rigoureux pour de nombreuses raisons; par exemple, on suppose que les événements que ***m*** et ***n* – *m*** **soient premiers sont statistiquement indépendants l'un de l'autre.** »

l'algorithme de Goldbach permet de prouver ***que cette affirmation est « Fausse »*** *pour une limite* ***n*** *fixée*, ainsi que la fonction asymptotique du TNP qu'il faudra donc modifier dans ces ***8 Fam (i)*** de nombres premiers **p' > 5**, représentant 26,666… % des entiers naturels non nuls; soit ***n / 3,75 .***

On peut déjà en déduire et affirmer, avec l'algorithme de Goldbach et celui d'Ératosthène, que cette fonctions ci-dessus (**1/**Ln **m** * Ln **n**) est parfaitement justifiée et permet d'estimer une quantité positive d'entiers **m** premiers mais aussi d'être ***non congru à n (mod P)*** , Conséquence directe du TNP, on obtient : **m /** Ln **n** une *proportion d'entiers* ***m*** **≢ n [P]** *et par conséquent une estimation non nulle d'entiers* ***m*** *qui précède un entier* ***m' = p'*** ⇔ **p' + q = n .** Le nombre d'entiers **m** *non congrus à* ***n*** *modulo* ***P*** *qui impliquent les nombres premiers* **q** ∈ **[*m ; n]*** *est équivalent à* ***m* / Ln n,** *lorsque* ***m*** → **+ ∞.«** *Ce que l'on verra ci-après avec les deux cribles qui caractérisent les fonctions du TNP ainsi que les définitions relatives à ces algorithmes dans ces 8 Fam(i).***»**

On en déduit le raisonnement suivant : ces entiers **m = p'** ou pas et tel que **m ≢ n [P]** fait de par cette congruence, que ***statistiquement*** il s'agit d'un même événements que **m** et **n** soit premiers, *alors on peut s'attendre suivant le TNP, à ce que la probabilité que* ***m*** *et* ***n* – *m*** *soient tous deux premiers soit égale à* ***1 / (Ln n . Ln m),*** caractérisé par le crible de Goldbach et Ératosthène, Cribles que l'on va utiliser pour recribler les nombres **m = p' ≤ n/2** ayant été criblés par Ératosthène, selon le même principe.**»]**

Ce que l'on verra ci-après avec les deux cribles qui caractérisent les fonctions du TNP ainsi que les définitions relatives à ces algorithmes dans ces 8 Fam(i).

Pour en déduire en fonction d'une limite ***n*** *fixée la fonction asymptotique* ***n* /** *(Ln* ***n*** **.** *Ln* ***2n)*** *qui estime le nombre d'entiers naturels non nul* ***A' premier p' et non congruent modulo P et ⇒ q = 2n – p'.*** *Peut on prouver que cette fonction «conséquence du TNP» pour toute limite* ***n ≥ 3 fixée ne peut être nulle, ie : il existe toujours p'+ q = 2n*** *?*

****

Légende de l'illustration exécuté avec le crible de : **G**oldbach, **É**ratosthène et EG2 unifiés.

Le crible G est une variante du crible d'Ératosthène, mais qui utilise les congruences pour une limite n:
«Propriété que l'on va utiliser sur les entiers ***A*** *impairs non nul de* ***1 à n*** *, congrus ou pas à* ***2n*** *modulo* ***P****:*
si ***A est congru = 0 ; sinon = 1.»***

Pour dans un premier temps marquer les entiers ***A*** *congrus à* ***2n*** *modulo* ***P*** *≤√* ***2n, où A et 2n*** *partagent le même reste dans la division par* ***P,*** *par conséquence et indirectement dans un deuxième temps : indiquer les multiples de* ***P*** *tel que* ***2n – A = B*** *.*
Propriété des congruences 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.*** *Inversement si* ***y*** *n'existe pas , alors* ***P*** *ne divise pas la différence* ***2n – A = B => q ,*** *qui est donc un nombre premier* ***q ∈ [n;2n].***

Ce document a pour but de vérifier pour une limite **n** fixée, les deux ensembles d'entiers naturels $\mathbf{A_n} \leq \mathbf{n}$ non nul, définis ci-après en fonction de leur congruence, représentés par des **1** ou des **0** en progression arithmétique de raison 30 et **P** un nombre premier $\leq \sqrt{2n}$. Afin d'en déduire l'impossibilité d'infirmer cette conjecture.

Résumé :
1_) l'ensemble des entiers $\mathbf{A_n}$ avec les $\mathbf{A \not\equiv 2n\ [P]}$ **premiers p'** vérifiant la conjecture **p'+ q = 2n .**

2_) l'ensemble des $\mathbf{A_n}$ criblés**,** avec les $\mathbf{A \not\equiv 2n\ [P]}$ pas nécessairement premiers, mais qui précèdent un nombre premier **(A + 30 = P'')** pour la même limite **n,** ***qui vérifieront la conjecture*** pour la limite ***suivante*** **n = 15(k+1) + i** ce qui donnera (**p'' + q = 2n + 30 = 30(k+1) + 2i**. Ces deux ensembles sont en moyenne de même densité pour une même limite **n** lorsque $n \to \infty$, de la forme **15k + i**, avec **i {1; 7; 11; 13; 17; 19; 23; 29}.**
On va utiliser les congruences et le principe d'Ératosthène pour cribler les entiers $\leq$ **n, tel que: A** $\equiv$ **2n[P].**

Le premier ensemble $\mathbf{A_n}$ ligne **E** qui vérifient la conjecture **(p'+ q =2n)** pour la limite **n** criblée, est constitué uniquement des entiers **A** premiers **P'** *(non congru à 2n mod P),* $\mathbf{2n \not\equiv A[P]}$.
Alors que le deuxième ensemble $\mathbf{A_n}$ **ligne E,** ne contient que des $\mathbf{A \not\equiv 2n[P]}$ **précédant** un nombre premier **p''; d'où (A+30)+q** vérifiera la conjecture pour la limite suivante **2n + 30 ligne E,** devenant ainsi **A+30 = p''** non congru modulo **P, dû** au décalage d'un rang des congruences.«C*es A ne sont pas obligatoirement premier, l'important est qu'ils soient non congruent à P*»
On obtiendra le résultat **(p''+ q = 2n+30)** pour la limite suivante criblée n+15 ⇒ 2n + **30.**

Ce deuxième ensemble d'entiers $\mathbf{A_n}$**,** à l'avantage de ne pas tenir compte uniquement des nombres premiers **P'** $\leq$ **n.**
Il contient pour une même limite **n** fixée, les entiers **A premiers ou pas** qui ne sont pas égaux **[P]** avec 2n appartenant à $\mathbf{A_n}$ et **qui précèdent** des nombres premiers **P'** suivant l'illustration ci-dessous.

1a_):
Première ligne G:
Ce sont les entiers $\mathbf{A \in A_n}$ non nul de **[1 à n]** appartenant à une famille Fam(i); en progression arithmétique de raison 30, de premier terme **(i)** $\in$ **{1,7,11,13,17,19,23,29}** qui seront criblés par $\mathbf{P \in P2n \leq \sqrt{2n}}$ pour une limite **n = 15k** ou **15k+(i)** en utilisant les congruences.

Les **A = 1** sont les **A non congrus** modulo **P** avec **2n**: $\mathbf{(2n \not\equiv A\ [P])}$ **d'où 2n – A = q > n,** premier car non divisible par **P.** On aura compris, que les **A = 0** , sont les entiers qui partagent le même reste **R** avec **2n** dans la division euclidienne par **P**: $\mathbf{2n \equiv A\ [P]}$ ou encore, **2n – A** n'est donc pas premiers car congru **0** modulo **P .**

2a_):
Ligne E: en dessous de la **ligne G**, c'est la **même ligne** $\mathbf{A_n}$ criblés pour la même limite **n=15k, mais** par le crible **E** Ératosthène avec $\mathbf{p \in Pn}$ l'ensemble des nombres premiers $\leq \sqrt{\mathbf{n}}$.
Les **A = 1** sont les nombres premiers **P'** $\leq$ **n ,** ***non congru (mod P)*** *et* si ils sont *congrus* ***(mod P)*** ils seront marqués en rouge **1**.
Les **0** en rouge ou **0** en noir, bien sûr sont les multiples de **p**, **congrus** ou **pas (mod P)** par le crible **G**.

Autrement dit dans l'illustration ci-dessous, un **1** de la ligne G au-dessus d'un **1 de la ligne E,** indique que ce nombre premier **1 = P'** d'Ératosthène est **non congru [P].**

Ils formeront avec leur complémentaire **q** premier, un couple (**P'+ q**) **= 2n**. Dans le cas contraire un **0** de la **ligne G** au-dessus d'un **1** ligne **E** devient **1 congru ,** donc **2n – p' ≠ q.**

Les **A = 1** ou **0** qui précèdent un nombre **A** premier **P''** représenté par un **1** ou **1 dans la ligne E** indiquent tout simplement, ***et ce***: pour la limite suivante **n =15(k+1)** les couples (p''+q) = 2n + 30 qui vérifieront la conjecture.
Cela permet de ne pas tenir compte uniquement des nombres premiers **P'**, c'est une égalité récurrente dû au décalage d'un rang des congruences, sur leurs successeurs **A + 30** lorsque la limite **n** augmente de 15.
Cette égalité rend impossible l'infirmation de la conjecture pour la limite suivante **n =15(k+1) + i.**

La propriété récurrente de l'algorithme G *ou égalité est prouvée de façon élémentaire* ci-après**.**
*(« elle est triviale, mais faute de la connaissance de l'algorithme **G**, elle n'a pu être étudiée par la communauté Mathématique.»)*
*Les congruences permettent de cribler uniquement jusqu'à **n** au lieu de **2n,** par Fam(i) sans perte de généralité, cela permet formellement de prédire la vérification de la conjecture pour la ou les limites suivantes: **n =15(k+1...+n) +(i)**, ce qui n'a jamais été fait.*

Dans l'illustration ci-dessous, nous avons à droite de la ligne **E**, le résultat réel de l'algorithme **EG2**.
On en déduit la fonction asymptotique qui donne environ le nombre **A non congrus[P]** La fonction :

G(n) vaut ~ $\frac{n}{\ln 2n}$, Ce qui implique le nombre de **nombre de premiers q ∈[n ; 2n],** pour une limite **n = 15k + (i)** fixée ; ainsi que sa **Fam (i)** déterminée en fonction **de la forme de n**.

Cette fonction asymptotique ci-dessous, est une variante de la fonction ***π (n),*** c'est un corollaire du TNP et elle ne peut être nulle, cela est expliqué en fin de document.
On utilisera son résultat, pour calculer le nombre de couples **(p'+q) qui a vérifié 2n ou (p''+q) le nombre de couples qui vérifiera 2n +30 ,** illustré ci-dessous **a une variation près.** Elle est une conséquence directe de cette propriété récurrente des congruences qui se décalent d'un rang et de la fonction ***π (n).***
Cette fonction est caractérisée par ces trois algorithmes et la propriété récurrente de l'algorithme dans les congruences par Famille (i):

Car en prenant le résultat de la fonction **G(n),** alors le nombre de couples **(p'+q)** décomposant 2n; ou 2n + 30 en modifiant cette fonction, elle devient **G(2n)** et elle vaut au minimum : ~ $\lim_{n \to +\infty} ¿ \frac{G(n)}{\ln(n)}$
qui estime le nombre de **A'≢** 2n[P] = **P' ⇒ P' + q = 2n.**

Ou la solution générale, *qui est la conséquence des deux fonctions du TNP indique que le nombre de couples **P'+ q = 2n** ; est équivalent à* $\frac{n}{(\ln(n) * \ln(2n))}$ *lorsque **n** tend vers + l'infini.*

<u>Illustration:</u>

Limite **n =15k** , en progression arithmétique de raison 15.

Les 1 montrent le début du décalage d'un rang des congruences de la **ligne G** pour chaque changement de limite: **n = 15(k+1);** alors que la ligne **E,** ne se décale pas bien évidemment.
Ce qui permet de prédire la vérification de la conjecture sur plusieurs limite 2n +30 successives.

Fam(i) = fam 30k +7 qui sont criblés par **G** et **E**: 7; 37; 67; 97; 127; 157; 187…..etc 30k +7

G[**1, 1**, 0, **1**, 0, **1**, 0, **1, 1**, 0] n = 300 ; 600 EG: = 4 p'+q; fonct: G(n) = **6**; ***G(n) /ln G(n) = 3,348…***
E[**1, 1, 1, 1, 1, 1, 0, 0, 0, 1**] eg2 réel = 4 et pour n+15, on aura avec G(n) = 4; ***G(n) /ln G(n) = 2,885…***

G[0, **1**, 1, 0, **1**, 0, **1**, 0, **1, 1**] n+15= 315; 630 EG: = 5 p'+q au minimum; G(n) = **6**; ***G(n) /ln G(n) = 6/ Ln 6 = 3,348….***
E[**1, 1, 1, 1, 1, 1, 0, 0, 0, 1**] eg2 réel = 4 *le dernier* 1 *ne permet pas de voir si il précède un* **1 = P' ou 0**

G[1, 0, **1**, 1, 0, 1, 0, 1, 0, 1, 1]n=330; 660 EG: = 4 p'+q; G(n) = **7**; G(n) /ln G(n)
E[**1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1**] eg2 réel = 6;

G[1, 1, 0, **1**, 1, 0, 1, 0, 1, 0, 1]n=345; 690 EG: = 5 p'+q; G(n) = 7; G(n) /ln G(n)
E[**1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1**] eg2 réel = 5;

G[0, 1, 1, 0, **1**, 1, 0, 1, 0, 1, 0, 1] n=360; 720 EG = 4 p'+q; G(n) = 7; G(n) /ln G(n)
E[**1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1**] eg2 réel = 6

G[1, 0, 1, 1, 0, **1**, 1, 0, 1, 0, 1, 0] n=375; 750 EG: = 5 p'+q; G(n) = 7; G(n) /ln G(n)
E[**1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1**] eg2 réel = 5 [1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0]

G[1, 1, 0, 1, 1, 0, **1**, 1, 0, 1, 0, 1, 0] n=390; 780 EG: = 6 p'+q; G(n) = 8; G(n) /ln G(n)
E[1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1] eg2 réel = 6 [1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0]

G[0, 1, 1, 0, 1, 1, 0, **1**, 1, 0, 1, 0, 1] n=405; 810 EG: = 5 p'+q; G(n) = 9; G(n) /ln G(n)
E[1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1]eg2 réel = 6 [0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1]

G[0, 0, 1, 1, 0, 1, 1, 0, **1**, 1, 0, 1, 0, 1] n=420; 840 EG: = 5 p'+q; G(n) = 9; G(n) /ln G(n)
E[1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1] eg2 réel = 6 [0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1]

G[1, 0, 0, 1, 1, 0, 1, 1, 0, **1**, 1, 0, 1, 0] n=435; 870; EG: = 6 p'+q; G(n) = 9; G(n) /ln G(n)
E [1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1] eg2 réel = 6 [1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0]

G[0, 1, 0, 0, 1, 1, 0, 1, 1, 0, **1**, 1, 0, 1, 0] n=450; 900 EG: = 5 p'+q ; G(n) = 9; G(n) /ln G(n)
E[1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0] eg2 réel =6 [0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0]

Pour n = 465 j'aurais 5 p'+q en réel, car les deux derniers A = 0 ligne **G** et **E** sont congrus **2n [P].**
En règle générale le nombre de p'+q et différent d'une unité avec le nombre de p'+q de la limite précédente.
l.g

Propriété récurrente de l'algorithme utilisant les congruences:
An ensemble des entiers naturels non nul, en progression arithmétique de raison 30

B_{2n} ensemble des entiers naturels complémentaires, appartenant à **[n ; 2n]** en progression arithmétique de raison 30 tel que **2n – A = B** d'où pas nécessairement de la même famille en fonction de la forme de **n = 15k + i.**
P_{2n} ensemble des nombres premiers ≤ **$\sqrt{2n}$**. **P'_p** ensemble des nombres premiers ≤ **n** tel que: **5 < P' ≤ n.**
[**Théorème :** Pour tout **A ∈ An** avec **P ∈ P_{2n} ;** tel que **2n ≢ A [P]** où **A** précède **A + 30 = P'**, la **CG** sera vérifiée quel que soit la limite **n = 15(k+1) + i** , succédant à la limite ***n=15k + i*** ayant été vérifiée**.**]
(«*On peut bien entendu, augmenter **n** de **1*** tel que n =15k + 1*et se reporter à la limite **n'** précédente avec sa **Fam(i)** ayant vérifiée la **CG,** **On peut montrer et illustrer cette propriété quel que soit la Fam(i) fixée** . Lorsque **n** augmente de **15** , les congruences se décalent d'un rang sur leur successeur, dans un intervalle fermé et délimité par la limite **n** avec le nombre d'éléments **fixés: criblés par les nombres P∈ P_{2n},** limité par la racine de 2n.*»**)**

Supposons : que ce décalage d'un rang des congruences modulo 30 ne se produise pas sur leur successeur **A+30**, ou que cette égalité soit fausse. On a deux cas possible :

Premier cas : **2n ≡ A[P] ce qui ⇒ P ∈ P_{2n}** divise la différence **B ∈ B_{2n},** donc **B = C est** multiple de **P** pour la limite **n = 15k + i fixée. («** *propriété des congruences bien connue,* ***2n = Py + A*** **;** *il existe y et y' tel que* ***: 2n = P*y + R et A*** *= P*y' + R =>* ***2n – A = P*(y – y');*** *d'où* ***P divise 2n – A. »)***

Lorsque la limite **n** augmente de 15, nous avons donc **(2n+30) ≡ (A+30) [P] ce qui ⇒ P** divise cette différence **C,** car multiple de **P** qui est toujours le même **!** Le contraire serait absurde et contraire au TFA**.**

Ce nombre **A**+30 premier ou pas, ne pourra donc vérifier la conjecture pour la nouvelle limite **n** augmenté de 15 soit **15(k+1) + i** , car son complémentaire : (2n + 30) – (A +30) = **B = C** qui est le même, il est toujours multiple de **P,** donc non premier**.**

Deuxième cas : **2n ≢ A [P] donc P ne divise pas la différence 2n – A, ce qui ⇒ 2n – A = B = q** qui n'est pas un multiple de **P ,** c'est donc un nombre premier**.**

Or **(2n+30) – (A+30) = q** premier qui est le même et non divisible par **P.**

Là aussi, si le décalage des congruences ne s'effectuait pas, ce **(A+30) qui était congru à 2n (mod P),** *il serait encore congru à* **(2n+30) (mod P) d'où : (2n+30) – (A+30)= q** qui est pourtant le même, il deviendrait divisible par **P**, tout autant absurde et contraire au TFA.

D'où et dans ce cas, **même** si on avait **A ≠ P'** lors de la limite **n = 15k + i** fixée, mais qu'il précède **A+30 = P' ∈ P'_p ;** suite au décalage d'un rang des congruences, ce nombre **(A+30) = P'** devient non congru **[P],** il vérifiera donc la **CG** pour la nouvelle limite **n = 15(k+1) + i** ; il formera avec **q** un couple de premiers **P' + q = 2n +30,** tel que : **(2n+30) – (A+30) = q.**

(« ***Autrement dit, on peut dire que B = C ou q ont pour antécédent A. Si 2n ≡ A [P] ; 2n – A = B qui est un multiple C de P , ayant pour antécédent A ; sinon si 2n ≢ A [P], 2n – A = B est un nombre premier q, ayant pour antécédent A non congruent [P].»***

On en déduit le constat suivant :
*Supposons que la conjecture soit fausse pour la limite **2n + 30 ou 30k + 2i :***

- *Il ne faut pas de **A premiers ou pas,** qui soit **non congrus (mod P)** précédant **A=P' .** Il ne faut pas non plus de **P'** consécutifs **non congrus (mod P) .***
- *Or le crible est récursif , il recommence au début de chaque limite **n+15** avec les **index** qui se décalent d'un rang.*
- *Il faudrait donc pouvoir réutiliser les restes **R** de **2n** par **P,** pour la limite **2n +30** afin d'être à peu près sûr, que cette supposition est vraie et ne soit pas invalidée par un de ces deux cas ci-dessus…*

- *Or il est impossible d'utiliser pour **2n + 30,** les restes **R de 2n, 2n -30 et 2n-60 …etc..** La division de **2n+30 par P,** ne donne qu'un reste **R** par nombre premier **P** qui crible et ceci, serait contraire au décalage d'un rang des congruences lorsque **n** augmente de 15, propriété prouvée de l'algorithme.*
- *Le nombre de premiers **P** qui criblent est limité par la racine de **2n , ou 2n +30** dans cette supposition, le contraire invaliderait le TNP.*
- *Dernière solution, il faut par conséquent marquer tous les A par ces nombres P qui criblent, **ie :** il faut qu'ils soient tous **congrus à (2n+30) (mod P). (« Ce qui est absurde,** il n'y aurait plus de nombres premiers **q ∈[n ; 2n]** alors qu'il y en avait **n / log 2n selon le TNP et serait contraire au TFA»)***
- ***D'où ceci est clairement impossible !** Le nombre de premiers **P** qui marquent les **A congrus** est limité par la racine de 2n et cela depuis le début de la limite n ≥ 150 **;** il viendrait aussi que l'égalité démontrée ci-dessus serait fausse, or l'index de départ des deux **algorithme ou cribles** n'est pas le même « voir ci-après page 5 »*
-

*(« On crible par Fam(i), et on veut prouver que ceci est vraie en utilisant une seule Fam(i), au lieu des 8 au maximum et de 3 au minimum en fonction de la forme de **n** avec sa limite **≥ 150 .»)***

- **Conclusion la conjecture ne peut être infirmée. D'autant que le décalage des congruences, et sa propriété, se produit sur plusieurs limites 2n + + + … + 30 consécutives, donc qui auront vérifier la conjecture !**
- Mais cette propriété, fait en sorte que le cardinal de la fonction **π(n)** ainsi que celle du **TNP,** qui a été défini et vérifié pour une limite **n = 15k + i ,** ainsi que pour **n = 15(k -1) + i** ne peut plus être modifié a une exception près pour la limite suivante **: n = 15(k+1) + i,** qui par supposition infirmerait la conjecture **! *Le contraire serait absurde.***
- Ce qui clairement, modifierait de façon importante ce cardinal de nombre premiers **[n ; 2n]** ayant été défini et vérifié lors des deux limites précédentes et antérieur, ainsi que pour la fonction **G(n)** et sa fonction d'estimation ci-dessous !
- *Est-elle indécidable ? Il faudrait qu'à une certaine limite **n,** donc pour **2n** il ne soit plus possible de vérifier les résultats des deux ensembles de **A** pour toutes les **Fam(i).***
- *Or cela ne serait pas suffisant, car suite à ce décalage qui se produit sur plusieurs limites successives, on a déjà les résultats pour ces limites **2n+30, 2n + k*30** et on ne peut pas redescendre indéfiniment en arrière car on tombe sur les résultats qui ont vérifiés la conjecture, **principe de la descente infinie**.*
- *Par conséquent, on peut aussi en déduire que la fonction **G(n)** qui* vaut $\sim \lim_{Gn \to +\infty} \square \frac{Gn}{\ln(Gn)}$ *donnant le nombre de couples **P'+ q = 2n – 30** d'où **P'+ q = 2n** donnera tout autant **P'+ q** pour la limite **2n** qui vient d'être vérifiée, d'où par la conséquence de la récurrence, vérifieront **2n+30,** donc cette fonction ne sera jamais nul et en utilisant toutes les **Fam(i)** au lieu d'une seule ……*
- *On peut d'ailleurs se limiter à la fonction **π(n)** par **famille i,** tel que le nombre de couples **P'+ q = 2n,** vaut*

  $\sim \lim_{n \to +\infty} \square \frac{\pi(n)}{\ln(2n)}$ conséquence de la fonctions ci-dessus avec la fonction $\frac{n}{(\ln(n) * \ln(2n))}$
- *en ne prenant en compte que le nombre de nombres premiers **p'** par fam(i) avec 2n / 30;*
- *Par exemple 2n = 600 ; nombre de P' pour la fam(i =7) = 7 < n=300 ;* $\frac{7}{\ln(2n)}$ *=2,3366….*

****

Annexe 1:

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 de $A \not\equiv 2n[P] \Leftrightarrow$ premiers $q \in [n ; 2n]$

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

Le TNP dit que $\pi(n) = \frac{n}{(\log n)} + o\left(\frac{n}{\log n}\right)$,

donc le nombre de nombres premiers dans]*n*,2*n*] vaut

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

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

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

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

*Tout nombre pair 2n ≥ 180 peut s'écrire comme la somme de deux nombres premiers (**P'+q**)) appartenant à une famille Fam(i) tel que définie en début de document.*

$C_2 \frac{G(n)}{\ln G(n)}$**;** où C_2 ≈ 1,320323… constante premiers jumeaux.

*On peut appliquer **en fonction** de* $G(2n)$ ***par Fam(i),** directement* **:** $\lim_{n \to +\infty} ¿ \frac{G(n)}{\ln G(n)} * C_2$

*Cette fonction, qui n'est aussi qu'une conséquence du TNP. Sinon il faut utiliser le facteur correspondant au nombre de familles qui décomposent **2n**. Exemple pour la limite **n =15k + 7**, il n'y a que trois familles qui décomposent **2n = 30k+2.***
La fonction serra donc utilisée avec le coefficient de 0,375 = 3/8.

$\lim_{n \to +\infty} ¿ \frac{G(n)}{\ln G(n)} * C_2 * 0{,}375$.

Exemple pour n = 496 , 2n = 992 , nombre de premiers [n ; 2n]des 3 familles {1 , 13 et 19 } =33. Alors que π(2n - n) vaut :73
Résultat :
*(33 /Ln 33) * 1,320323 = 12,… couples , pour un réel de 13.*
*(73 / Ln 73) * 0,375 *1,320323 = 8,..*

*Lorsque 2**n** tend vers l'infini, iI vaut mieux utiliser la fonction générale avec **π(2n)** et l'un des 3 coefficients (0,375 ; 0,5 ; 0,75) et aucun si 2n = 30k.*

$\lim_{n \to +\infty} \frac{\pi(2n)}{\ln \pi(2n)} * C_2 * 0{,}5$.

Exemple 2n = 1 000 000 010 la fonction ci-dessus avec **0,5** de coefficient du fait qu'il y ai 4 familles [1,7,13, et 19} qui criblent, donne comme résultat : 1891734 couples < 2422662 réels

Pour 2n = 3 000 000 000 on aura par la fonction **sans le coef** car 2n = 30k : ≈ 10150924 pour un réel 12 224533

Pour **2n + 2**, on aura par la fonction avec le coef de **0,375** : ≈ 3 806 596 pour un réel 4 584 281 .

Ce qui n'est que la conséquence des 8 Fam(i) divisées par 8 et multipliées par 3, conséquence du TNP caractérisé par le crible Ératosthène et sa fonction d'estimation de **π(n)**.
On en déduit que la variation du nombre de couples qui décomposent **2n**, serra minime pou **2n + 2**.

Il est par conséquent impossible de l'existence d'un entier **2n**, où la fonction ci-dessus soit nulle étant donnée qu'elle vérifie la conjecture lors de la limite **n =15(k-1)** ayant donné le résultat de la limite suivante **n = 15k** ainsi que le résultat dela limite précédente **n =15(k+1)** supposée infirmer la conjecture.

****

En information complémentaire : *sont donnés les différents indexes (idx) de départ des deux fonctions relatives aux cribles (programmes) de **G** et **E** pour : 15k = 900 et **fam 7** ;*
Afin de comprendre la différence entre les deux fonctions qui criblent et le principe de base du fonctionnement.
Les index de départ pour chaque premier P du crible G ; limite n = 900 + i, i = 7 et fam = 7 : relatif aux j%30 == 7 «du premier exemple ci-dessous.»

Exemple: 2n= 1814, 1814 mod 7 = R =1 ; on calcul l'index de départ :1+2P =15 ; +2p = 29 ; + + + …. + 2p = 127 congru 7[30]

127%30 = Fam7; idx = 127//30 = 4 l'index de 127, que l'on marque d'un 0, puis par pas de 7 → 907//30.

On réitère avec P = 11 etc…etc

***Goldbach* :**

*P = 7 , idx = **4** puis par pas de 7* **31→n//30**

*P = 11 , idx = **10** puis par pas de 11* de 17 →**n//30**

*P = 13 , idx = **0** puis par pas de 13* de19 →**n//30**

*P = 17 , idx = **3** puis par pas de 17* de 29 →**n//30**

*P = 19 , idx = **14** puis par pas de 19 pas du même idx.*

*P = 23 , idx = **15** puis par pas de 23 l'idx de départ.*

*P = 29 , idx = **9** puis par pas de 29*

*P = 31 , idx = **22** puis par pas de 31*

*P = 37 , idx = **9** puis par pas de 37*

Ératosthène : (on a besoins que de 4 couples de premiers **[7;31])**

dans Ératosthène **7*31** = j ; j%30 ==fam, partent de idx **= 7,** puis par pas de 7 et de

dans Ératosthène **11*17** = j ; j%30 == fam, partent de idx **= 6** puis par pas de 11 et

dans Ératosthène **13*19** = j ; j%30 == fam, partent de idx **= 8** puis par pas de 13 et

dans Ératosthène **23*29** = j ; j%30 == fam, partent de idx **= 22** puis par pas de 23 et

***chaque j est donc un produit fam7[30] contrairement à Goldbach** et ne part*

***Et 31** > racine de 907, ne marquera rien, ainsi que 23 et 29 qui ne marquent que*

*et en dernier P = 41 **, idx > 914** d'où il ne peut cribler. P≤ 1814 = 42,….*

*Dans Goldbach **:** Si R est pair, j = R + k*P . (« voir programme si R est impair j=R puis + 2*P ….etc.»)*
Si j%30==fam ; alors :
j//30 = début d'index = idx . Puis :
*P_i part de idx, par pas de P_i où on remplace le **1** par **0** →n // 30 . Ensuite on réitère avec P_i et R_i suivant.*

En fonction de la forme de **n**; on appliquera un coefficient multiplicateur.

On utilise ces coefficients **en fonction des Fam(i)** de nombres premiers criblées et **n ≥ 150** :

On a 8 fam(i) ≡ **1 ou P**[30] avec P appartenant à [7 ; 29]

Pour **n = 15k +** n' → {1,7,11,13,2,4,8,14} coef = 0,375 ; ***ce qui donne 3 fam(i) sur 8*** qui peuvent être criblées.
les 5 autres ne peuvent donc pas donner de couples p+q = 2n ; car leur complémentaire serait multiple de 3 ou de

Pour **n=15k +** n' → *{5,10}* coef = 0,5 ; ***ce qui donne 4 fam(i) sur 8*** qui peuvent être criblées

Pour **n = 15k +** n' →{3,6,9,12} coef = 0,75 ; ***ce qui donne 6 fam(i) sur 8*** qui peuvent être criblées

Pour n=15k + n', n' = 0 pas de coefficient : ***les 8 fam(i) peuvent être criblées***

Il y a donc 15 formes de *n'. Mais on peut travailler qu'avec une fam (i) sans perte de généralité*

****

Autre illustration *:*

Illustration ***Crible G*** *pour n = 12000 ; fam 7[30]: « 7, 37, 67, 97, 127, ….etc ..11977 < 12000.*
Le résultat de ce criblage de la fam 7[30] donnera la famille 23 [30] complémentaire, soit : q premier mais ≡ 23[30].
Le cinquième élément ***1 = 127 non congru 12000 [P_i] donne q = 11873 . 127 est l'antécédent de 11873 par la fonction G.***

Vérifions le nombre de **A** non congrus **modulo P** : les nombres **A ≠ P' = 0** et les **A = P'= 1** qui précèdent un nombre premiers **P'** congru ou pas, marqués **1**,**1**; illustrés ci-dessous par **0** ou **1** **.** *Ce qui permet de comprendre pourquoi cette conjecture ne pouvait avoir de contres exemples.*

On peut même vérifier avec ce criblage, jusqu'à quelle limite ***2n + k*30*** *la conjecture serait vérifier sans avoir besoins de procéder à un autre test de criblage, et ainsi de donner le nombre de couples* ***(P'+q) = 2n + k*30***

Crible G des congruences .
Et ligne en dessous:
Crible E

G [1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,
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, 0, 0, 0, 1, 0, 1, 1, 0, 0,

G [0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
E [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, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1,

0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1,
0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0,

1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0,
0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0,

1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0,
1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1,

1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0,

0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1,
1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0,

0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1,
0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1,

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

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

Résultat, au minimum pour cette famille : 60 couples sur ≈ 148 nombres premiers q, vérifieront la conjecture pour 2n + 30 ; *dont quasiment autant d'entiers* ***A = 0 non premiers P'****, mais* ***non congrus (modulo P)*** *qui précèdent un nombres premiers* ***P' , congru ou pas,*** *d'où l'intérêt de cet algorithme dans les congruences par rapport à Ératosthène classique , dont le résultat du nombre de premiers ≤ n, n'a que peut d'intérêt.*

****

Fam (i) = 1 de 300 à 600 : 1 n'est pas premier mais pour l'algorithme on l'utilise, pour vérifier 2n – 1.
A = 1, 31, 61, 91, 121, 151,181…..**fonction**: 1,55456 * (G(n) / Ln G(n))

Crible G :300 [1, 1, 0, 1, 1, 1, 1, 1, 1, 0] **7 couples p'+q** pour vérifier la limite 15(k+1)
Crible E :300 [1, 1, 1, 0, 0, 1, 1, 1, 1, 1] **6 couples réel** [1, 1, 0, 0, 0, 1, 1, 1, 1, 0]

Crible G: 315[0, 1, 1, 0, 1, 1, 1, 1, 1, 1] **6 couple p'+q**
Crible E: 315 [1, 1, 1, 0, 0, 1, 1, 1, 1, 1] **7 couples réel** [0, 1, 1, 0, 0, 1, 1, 1, 1, 1]

Crible G: 330[1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1] **5 couples p'+q**
Crible G: 330[1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0] **7 couples réel** [1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0]

Crible G: 345[0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1] **6 couples p'+q**
Crible E: 345[1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0] **5 couples réel** [0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0]

Crible G: 360[1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1] **6 couples p'+q**
Crible E: 360[1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1] **7 couples p'+q réel** [1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1]

Crible G: 375[0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1] **5 couples p'+q**
Crible E: 375[1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1] **6 couples p'+q réel** [0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1]

Crible G: 390[0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1] **4couples p'+q**
Crible E: 390[1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0] **5 couples p'+q réel ;** (1,55456 * G(n) / Ln G(n)) = 5,9806…..

Crible G: 405[1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1] **5 couples p'+q**
Crible E: 405[1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0] **5 couples p'+q**

Crible G: 420[1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1] **5 couples p'+q**
Crible G: 420[1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0] **6 couples p'+q**

Crible G: 435[0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1] **4 couples p'+q**
Crible E: 435[1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0] **5 couples p'+q**

Crible G: 450[0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1] **4 couples p'+q**
Crible G: 450[1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1] **5 couples p'+q**

Crible G: 465[1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1] **3 couples p'+q**
Crible E: 465[1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1] **5 couples p'+q**

Crible G: 480[0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1] **6 couples p'+q**
Crible E: 480[1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0] **3 couples p'+q**

Crible G: 495[0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0]
Crible E: 495[1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0] **6couples p'+q**
Crible G: 510[1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0]
Crible E: 510[1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0]

Crible G: 525[1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1]
Crible E: 525[1, 1, 1, 0, **0**, 1, 1, 1, 1, 1, 0, 1, 0, **0**, 1, **0, 0**]

Crible G: 540[0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1]
Crible E: 540[1, **1, 1**, 0, 0, **1**, 1, 1, **1**, 1, 0, 1, 0, 0, 1, 0, **0, 0**]

Crible G: 555[1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1]
Crible E: 555[**1**, 1, 1, 0, 0, 1, **1**, 1, 1, 1, **0**, 1, 0, **0**, 1, 0, 0, **0**]

Crible G: 570[0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1]
Crible E: 570[1, **1**, 1, 0, **0**, 1, 1, **1**, 1, 1, **0, 1**, 0, 0, **1**, 0, **0**, 0, **1**]

Crible G: 585[0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0]
Crible E: 585[1, 1, **1**, 0, **0**, **1**, 1, 1, **1**, 1, 0, **1, 0,** 0, 1, 0, 0, **0**, 1]

Crible G: 600[0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0]
Crible E: 600[1, 1, 1, **0**, 0, **1, 1**, 1, 1, 1, 0, 1, **0, 0**, 1, 0, 0, 0, **1**, 1]

Crible G: 615[1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1]
Crible E: 615[**1**, 1, 1, 0, **0**, 1, **1, 1**, 1, 1, **0**, 1, 0, **0**, **1**, 0, 0, **0**, 1, **1**]

****

Cette illustration permet de se rendre compte de l'importance de ces entiers **A non premiers = 0**, qui précèdent des nombres premiers P'. Sans cela, probablement qu'un contre-exemple aurait été trouvé.

D'où on en déduit que la fonction $\frac{Gn}{\ln(Gn)}$ ne peut être nulle , supposons le contraire :

Il faut systématiquement revenir en arrière 2n – 30.
Or 2n – 30 a été vérifié, ainsi que 2n – 60 …etc 2n – 30k ; où cette fonction ne pouvait être nulle, sans cela la conjecture aurait été fausse.

****

<u>Note</u>, on peut démontrer :

*[(« Suivant le principe du crible d'**Ératosthène qui est largement connu,** il est inutile de démontrer que pour extraire le nombres premiers inférieur à une limite **n** donnée : il suffit d'utiliser* $p \in P_n \leq \sqrt{n}$ *pour cribler les* $A \in A_n$ *multiples de* **p** *de* **1 à n** *et avec* $P \leq \sqrt{2n}$ *pour cribler les* $B = C \in B_{2n}$ *multiples de* **P de n à 2n.**
Si $B \in B_{2n}$, *n'est pas divisible par* $P \in P_{2n}$, *alors* $B = q$ *un nombre premier par évidence et* **B** *est* ***non congru 0 modulo P.***
$2n \not\equiv A[P] \Rightarrow 2n - A = B$ ***qui*** *ne peut donc être divisible par* **P.**

(« (1) rappel : si est seulement si, **2n** *et* **A** *sont égaux modulo P , il existe y et y' tel que :* $2n = P*y + R$ *et* $A = P*y' + R \Rightarrow 2n - A = P*(y - y')$ *donc* **P divise 2n – A.**»)

Donc aucun nombre premier **P** *ne divise* **2n – A, d'où** $2n - A = q$

On veut démontrer pour **2n – A non** *premier, qu'il n'existe pas de premier* $P > 2n$ *tel que* **(1)** *que* $2n - A = Py$ *avec* $y \in \mathbb{N}$ **.**
on a : $n \geq 2$, $2n \geq 4$;
on a : $-n \leq -A \leq -1$;
$2n - n \leq 2n - A \leq 2n - 1$
$n \leq 2n - A \leq 2n - 1$

Supposons que $2n - A$ *non premier, alors il existe au moins* **P** *et* **p'** *premier* $\geq \sqrt{2n}$ *car on a démontré* **(1) .**
Donc prenons **P** *et* **p'** *les plus petits possibles, soit* $\sqrt{2n}$**.**
On a $(\sqrt{2n})^2 = 2n$ **, or** $2n - A \leq 2n - 1$ **, Ce qui est impossible ,** *on a donc* $P * p' \leq 2n - 1$ *et au moins :*

un diviseur P_i *de* $2n - A$ *est* $\leq \sqrt{2n}$.

Conclusion $2n \not\equiv A\ [P] \Rightarrow 2n - A =$ ***q premier, car P ne divise pas*** $2n - A$ »)].

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 C++ (*fourni à la demande*) est fixé avec une limite N **début = 3000000000** et Fin = **3000000330**

il progressera automatiquement 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 ; Fam = (**l'une des 8 fam(i)**) **2n = 30k**

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);

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 =((11,17,23,29)); 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**

****

Programmes des algorithmes :

Pour une limite n = 15k + i , on change la fam(i) correspondante en fin de programme :
EX : GCrible(premiers, n, 17) # au choix (1,7,11,13,17,19,23,29)

Par exemple : pour 15k + 17 on fixe la Fam(17);
pour n = 15k , une des 8 fam(i)

1_) Crible G :

```
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)*[1]     # Ou: on rapelle le tableau Ératosthène criblé de N/30 cases
    lencrible = len(crible)

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

    for i, premier in enumerate(premiers):
        reste = n2 % premier
        #print(reste)  # enlever dièse # pour imprimer les (reste)
            # tant que ri % 30 != fam on fait ri += 2pi
        if reste % 2 == 0:
            reste += premier
        pi2 = 2*premier
```

```
        while reste % 30 != fam:
            reste += pi2
        # Ensuite on divise ri par 30 pour obtenir l'indexe
        reste //= 30
        # On crible directement avec l'index
        for index in range(reste, lencrible, premier):
            crible[index] = 0

    total = sum(crible)
    print("crible:", crible) # mettre dièse # pour bloquer la fonction print
    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, 17) # 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")
```

****

2_) Crible E:

```
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)*[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
        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 Gold-
bacn, on return "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, 17) # 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")
```

****

3_) Crible EG2:

fusion des 2 programmes donnant directement le nombre de couples p+q = 2n

```
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):
    n1,n = int(n**0.5),int((2*n)**0.5)
    prime_E,prime_EG=[2, 3],[]
    sieve_list = [True] * (n+1)
    for each_number in candidate_range(n):
        if sieve_list[each_number]:
            if each_number > n1:
                prime_EG.append(each_number)
            else:
                prime_E.append(each_number)
            for multiple in range(each_number*each_number, n+1, each_number):
                sieve_list[multiple] = False
    #print(prime_EG[-1])
    return prime_E[3:],prime_EG


def Criblage_EG(Premiers,Premiers_suite, 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
    Premiers+=Premiers_suite
    del Premiers_suite # suppression du tableau devenu inutile
```

```
    nbpremiers = len(Premiers)
    n2 = 2*n
    for premier in Premiers:
        reste = n2 % premier
        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 avec l'index
        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 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 √n et de √n à √2nN
    Premiers_E,Suite_E = eratostene(n)
    # On crible
    fam = 17      # 1 ou 7, 11, 13, 17, 19, 23, 29, 1 au choix
    Criblage_EG(Premiers_E,Suite_E, n, fam)

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