

Pour démontrer la conjecture de Goldbach **CG**, $2n > 4$ est la somme de deux nombres premiers impairs. On va utiliser en duo le crible d'Ératosthène **CE** avec l'algorithme de Goldbach **AG** pour une limite n , progressant modulo 15.

Algorithme que l'on va construire afin d'utiliser ses propriétés dans les congruences avec des raisonnements qui permettent d'en déduire la résolution de la **CG** par l'absurde.

1_) On va introduire les éléments entrant dans le crible d'Ératosthène **CE** et **AG**

Pour ce faire, on va utiliser des ensembles d'entiers naturels non nuls, en progression arithmétique de raison 30 et de premier terme (**i**), appartenant à **8 familles** distinctes ; excluant les multiples de 2, 3 et 5.

Le **crible d'Ératosthène** vise à marquer l'ensemble des entiers naturels non premiers.

Soit P_n l'ensemble des nombre premiers $p \in [7, \sqrt{n}]$.

- On utilisera P_n pour **CE** et P_{2n} (l'ensemble des nombres premiers $P \in [7, \sqrt{2n}]$) pour l'**AG**

Soit P'_p l'ensemble des nombres premiers $P' \in [7, n]$

Soit $A_n < n$ l'ensemble des entiers impairs en progression arithmétique de raison 30 de premier terme (**i**) appartenant à **[1; 29]** formant 8 ensembles disjoints, excluant les multiples de 2, 3 et 5.

- Ainsi, tout $A \in A_n$ est ou bien premier $P' \in P'_p$, ou bien c multiple d'un nombre premier $p \in P_n$

- Démonstration:

- o si A est premier, A est inférieur ou égale à n par définition, et \geq à 7 par définition (puisque A n'est ni 2, 3 ou 5)
- o Si A n'est pas premier il existe un nombre premier p qui le divise (par définition d'un nombre non premier). p est \geq à 7 car il n'est pas divisible par 2, 3 et 5.
 - Si A n'est pas premier il existe une combinaison de nombres premiers p_i tels que $A = p_1 p_2$ soit leur produit. Prenons p le plus petit de ces nombres premiers.
 - Si p est strictement supérieur à n , A est strictement supérieur à p^2 , donc $(\sqrt{n})^2$, donc n . Dans ce cas A est strictement supérieur à n , ce qui est contraire à sa définition. Donc tout nombre premier dont A est multiple, est inférieur ou égal à n ; car au moins l'un des diviseurs p_i de A est plus petite que \sqrt{n} .
 - On définit que ce nombre premier $p \in P_n$, est utilisé pour le **CE**.

Cet ensemble A_n sera aussi utilisé par le crible **AG**, qui sera criblé en utilisant les congruences, ie : les restes R de $2n$ par P selon le principe d'Ératosthène, mais avec $P \in P_{2n}$, tel que : P_{2n} l'ensemble des nombres premiers $\in [7, 2n]$; on aura donc $p \leq P \leq P'$ en fonction du crible.

Soit B_{2n} l'ensemble des entiers naturels B impairs en progression arithmétique de raison 30, $\in [n, 2n]$ qui est la suite d'un des **8 ensembles** de A_n ; (Pas nécessairement le même).

On définit $B \in B_{2n}$: il est soit un multiples C d'un élément P de $P_{2n} < 2n$;

Soit un nombre premiers q , tel que : $p \leq P \leq P' < q$. («Un entier qui n'est pas divisible par $P \leq 2n$ est premier.»)

Si n est un nombre premier P' alors, $2n = P' + P'$.

A_n et B_{2n} en progression arithmétique de raison 30, appartiennent aux 8 familles arithmétiques tel que définies ci-dessous, «mais pas nécessairement la même, en fonction de la forme de n , voir ci-dessous **2_)** »

2_) Introduisons la notion de famille Fam(**i**).

Fam(i) est l'ensemble des nombres A_n et B_{2n} entrant dans la progression arithmétique de raison 30 et de début i.
 «Ce qui représente 26,6666...% de l'ensemble de ces entiers naturels divisé par 3,75 excluant 2,3 et 5 avec leurs multiples.»

Nous allons nous intéresser à ces 8 familles **Fam(i): Fam(1), Fam(7), Fam(11), Fam(13), Fam(17), Fam(19), Fam(23) et Fam(29).**

(«**Note : Sans perte de généralité** une seule Fam(i) est suffisante afin de vérifier la CG pour une limite $n = 150$ fixée, elle représente donc 3,3333...% de ces entiers ainsi définis et que l'on dénombre en divisant n par 3,75. Puis par 8»)»

Démontrons que tout nombre premier supérieur à 30 appartient à l'une de ces familles.

Soit p un nombre premier supérieur à 30. Il n'existe donc pas de décomposition en nombre premier de p .

30 étant le produit de 2, 3 et 5. Tout nombre premier supérieur à 30 n'est pas divisible par 2, 3 et 5 et s'écrit donc de la forme $2^i \cdot 3^j \cdot 5^k + pr$ avec pr non divisible par 2, 3 et 5 et inférieur à 30.

Donc pr étant l'un des nombres premiers inférieurs à 30 et différents de 2, 3 et 5 $\in [7;29]$

[«Ces 8 Fam(i) le nombre d'entiers de ces 8 fam(i) s'obtient simplement par la division de n par 3,75.

D'où, $30 / 3,75 = 8$, donnera les 8 débuts (i) tel que : 1,7,11,13,17,19,23,29 . "Sans les multiples de 2,3 et 5"

Pour tout nombre $2n \text{ modulo } 30 \geq 300$, il y a par conséquent un couple de nombre $P' + q = 30k + 2(i)$ appartenant à ces Fam(i).

Pour fixer la Fam(i) en fonction de la forme de $n = 15k + (i) \geq 150$, il suffit de procéder par soustraction avec $2n[30]$, tel que :

$(30k+2i) - \text{Fam}(1) = \text{Fam}(1)$ «modulo 30 bien sûr, ainsi que i défini ci-dessous en 4_)»; ou encore : $(30k+2) - \text{Fam}(13) = \text{Fam}(19)$.

L'AG donnera un couple $P' + q = (30k+2)$ en criblant les congruences jusqu'à n .

«**Note : dans la Fam(1), 1 ne peut constituer un couple $P' + q$, mais on l'utilise dans l'AG, pour dénombrer les nombres premiers de n à $2n$.**»

$(30k+4) - \text{Fam}(11) = \text{Fam}(23)$; $(30k+4) - \text{Fam}(17) = \text{Fam}(17)$, L'AG donnera un couple $P' + q$ vérifiant $(30k+4)$

ou encore: $(30k+6) - \text{Fam}(7) = \text{Fam}(29)$; $(30k+6) - \text{Fam}(13) = \text{Fam}(23)$ ou : $(30k+6) - \text{Fam}(17) = \text{Fam}(19)$;

L'AG donnera un couple $P' + q$ vérifiant $(30k + 6)$...etc etc. Il est clair, que deux éléments de ces Fam(i) peuvent appartenir à une seule Famille ou à deux distinctes. **Les 15 formes de n , sont page 9 annexe 1»]**

3_) Rappelons certaines propriétés de la division et des congruences

(«Avec $2n, P, q, r$ des entiers naturels non nuls, P divise $2n$ si le reste r est nul. $2n = Pq + r$, on dit aussi que $2n$ est congru à r modulo P d'où P divise $2n - r$.

Propriétés des congruences, que l'on va utiliser en utilisant les restes R de $2n$ par P ainsi que les nombre premiers $P \in P_{2n}$ pour calculer les entiers $A \in A_n$, qui partagent le même reste R avec $2n$ et tel que, $2n$ et A seront égaux modulo P , que l'on écrit : $2n \equiv A [P]$, donc congrus modulo P ce qui $\Rightarrow 2n$ et A sont congrus à R modulo P et P divise bien $2n - A$; preuve : $2n = P*y + R$ et $A = P*y' + R \Rightarrow 2n - A = P*(y - y')$ donc P divise bien $2n - A$.»)

Rappelons les ensembles A_n, B_{2n} et P_{2n} tel que défini dans 1_)

Si $A \in A_n, 2n - A$ appartient à $[n, 2n]$ et $2n - A$ appartient à B_{2n} .

Comme il existe un nombre premier $P \in P_{2n}$ tel que $2n \equiv A [P]$, alors $2n - A = B$ est un multiple C de P ;

Sinon : si $2n \not\equiv A [P]$ alors $2n - A = B$, est un nombre premier = q par la définition 1_) et 3_) car non divisible par P .

Montrons par principe si besoin est :

- Si A et $2n$ partagent le même reste R de la division de $2n$ par $P \in P_{2n}$, en utilisant les congruences : on dit qu'ils sont égaux modulo P , noté $[P]$.
« On utilisera couramment l'expression A congruent à P ou non congruent à P , en fonction de la congruence de A calculé par l'AG ; c'est-à-dire les A qui partagent le même reste R avec $2n$.»
- Pour le calcul des congruences de A par l'AG, on utilise $P \in P_{2n}$.
- Si $2n \equiv A [P]$, alors il existe un nombre q_i entier naturel tel que $2n = Pq_i + A$ et $2n - A = Pq_i$; donc :
- P divise $2n - A = C$ qui est un multiple de P .
- Si $2n \not\equiv A [P]$ il n'existe pas un nombre Pq_i , donc P ne divise pas $2n - A = q$ qui est un nombre premier, selon la définition en 1_)

On définit R_{2n} l'ensemble des restes R de la division de $2n$ par $P \in P_{2n}$ qui seront utilisés par AG afin de marquer les $A \equiv [P]$.

4_) Explication et propriété de l'algorithme de Goldbach :

« On sait que la CG est vérifiée jusqu'à $2n \leq 300$. Sans perte de généralité on utilise la limite $n \geq 150$ tel que définie ci-dessous, afin de parcourir l'ensemble des nombres pairs $2n \geq 300$ et une seule Fam(i) par rapport à la forme de n .»

AG progresse modulo 15 :

Soit une limite $n \in \mathbb{N}$. n peut s'écrire comme $15k + i$, ($k \in \mathbb{N}^*$) avec i appartenant à $[0; 14]$. En fonction de la forme de n et pour une limite n fixée, on fixe une Fam(i) définie en (2_).

(«il y a 15 formes de n , voir annexe 1 page 9, si n est de la forme $15k + i$, i désigne une des 8 fam(i) au choix.»)

Puis on va calculer la congruence des A , appartenant à la Fam(i) fixée pour la limite n .

(« Soit $P' \in P'_p$, si n est un nombre premier P' , alors $2n = P' + P'$ vérifie la CG.»)

On définit un couple de premiers $P' + q = 2n$, vérifiant CG selon l'égalité :

Si $A = P'$ et tel que : $2n \not\equiv A [P]$ alors selon 3_) et la définition 1_) $2n - A = q$, donc $P' + q$ vérifie la CG.

(«Note: Si $2n - A = B$ multiple d'un nombre premier, n'appartient pas à une de ces 8 Fam(i), c'est donc un multiple de 2, 3 ou 5 qui sont exclus de l'algorithme, ce qui implique que l'on n'a pas fixée la bonne Fam(i) en fonction de la forme de n .»)

$B \in B_{2n}$ ne sera pas nécessairement de la même Fam(i) fixée pour $A \in A_n$ en fonction de la forme de n

On a $2n - A = B$, soit C ou q suivant la congruence de A .

(«exemple pour $n = 151$, on fixe Fam(i) = 13, d'où $2n = 302$, donc $B \in B_{2n}$ et \in Fam(i) = 19 qui est donc la Fam(i) complémentaire dans ce cas.»)

4a_) L'AG va calculer les congruences de A_n comme défini ci-dessus en 3.

Ce qui nous intéresse, ce sont les $A = P' \in P'_p$ ou Pas , sous la condition: qu'ils soient non congruents à P , tel que $2n \not\equiv A [P]$, $P \in P_{2n}$, d'où cela $\Rightarrow q$ premier selon 3_).

Par conséquent : si $A = P' \in P'_p$ et qu'il est non congruent à P , nous aurons donc un couple $P' + q = 2n$ qui vérifie la CG, pour la limite n fixée sans avoir besoin de le vérifier.

«Car par la définition de 1 : q est premier en fonction de A si et seulement si il est non congruent à P ; A et $2n$ ne partagent pas le même R de $2n$ par P ».

Si $A \neq P'$: mais qu'il est lui aussi non congruent à P , ie : $2n \not\equiv A [P]$ ce cas nous intéresse, car il va permettre de vérifier la CG pour la limite $n = 15(k+1) + i$; Si et seulement Si il précède $A+30 = P'!$

Car suivant la propriété de l'algorithme et des congruences dans ces suites arithmétiques :

5a_)

Si A est **non congruent** à P et qu'il précède un nombre premier $A+30 = P'$; lorsque la limite n va progresser modulo **15**, il s'ensuit par obligation, un décalage d'un rang des congruences qui se reportent sur son successeur $A+30$. («on appelle congruence, les A , congruents ou pas $[P]$ »)

6a_) Propriété «récurrente des congruences» de **AG**:

[Quel que soit $A \in A_n$ avec $P \in P_{2n}$; tel que $2n \not\equiv A [P]$ 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** et se reporter à la limite n précédente avec sa $Fam(i)$ ayant vérifiée la CG, expliqué ci-après page 6 : **On peut montrer et illustrer cette propriété quel que soit la $Fam(i)$ fixé** . Lorsque n augmente de **15** , les congruences se décalent d'un rang, dans un intervalle fermé, délimité par la limite n et le nombre d'éléments fixés, criblé par $P \in P_{2n}$ limité par la racine de $2n$. »)

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

Premier cas : $2n \equiv A [P]$ ce qui $\Rightarrow P \in P_{2n}$ divise la différence $B \in B_{2n}$, donc $B = C$ multiple de P pour la limite $n = 15k + i$.

Lorsque la limite n augmente de 15, nous avons donc $(2n+30) \equiv (A+30) [P]$ ce qui $\Rightarrow P$ qui divise **toujours** cette même différence C , car multiple de P qui sont **toujours les mêmes !**

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 $B = C$ qui est le même, il est toujours multiple de P , donc non premier.

Deuxième cas : $2n \not\equiv A [P]$ donc P ne divise pas la différence $2n - A$ ce qui $\Rightarrow 2n - A = B = q$.

Or $(2n+30) - (A+30) = q$, toujours le même, non divisible par P .

«Alors que pour la limite précédente on avait bien $2n \equiv (A+30) [P] \neq q$.»

Là aussi si le décalage ne s'effectuait pas, alors $(2n+30)$ serait encore congrus à $(A+30) [P]$ et par conséquent q n'existerait plus et serait divisible par P , tout autant absurde et contraire au TFA.

D'où et dans ce cas, **même** si on avait $A \neq P'$ lors de la limite $n = 15k + i$ fixée, mais qu'il précède $A+30 = P' \in P_p$ ce nombre $(A+30)=P'$ congru ou pas: **suite au décalage il serra non congru** et donc vérifiera la CG pour la nouvelle limite $n = 15(k+1) + i$; il formera avec q un couple de premiers $P' + q$ tel que : $(2n+30) - (A+30) = q$ selon **1_)**.

(«**Cette preuve** de récurrence de l'algorithme « propriété des congruences » est aussi montrée si besoin, en page 13. »)

(« Autrement dit, on peut dire que $B = C$ ou q a pour antécédent A . Si $2n \equiv A [P]$; $B = C$ est un multiple de P ayant pour antécédent A ; sinon si $2n \not\equiv A [P]$, donne $B = q$ qui est un nombre premier, ayant pour antécédent A non congruent $[P]$. D'où: q dépendra bien de $P' \leq n$, si ce dernier est non congruent à P . »)

L'algorithme produit une image récurrente, pouvant être utilisé quel que soit la limite $n = 15(k+1) + i$ fixée avec sa $Fam(i)$, que l'on peut vérifier avec le programme de l'algorithme ci-après.

(7a_)

[« Partant de ces égalités ci-dessus, on va construire l'algorithme /crible, où les entiers A_n seront représentés par des **1** au départ de l'algorithme, qui les changera en **0** si $2n \equiv A [P]$ autrement dit on marquera **0** tous les **1** qui partagent le même reste R avec $2n$. De la même manière avec Ératosthène lorsque $A \in A_n$ est multiple de $p \in P_n$.

Un **1** indiquera par conséquent un nombre premier **P'** dans **CE** ; ou un **A non congruent [P]** dans **AG**,
d'où $2n \equiv A [P] \Rightarrow q$, appartenant à **B_{2n}** dans **[n ; 2n]**.»]

L'**AG**, utilise le même principe qu'**Ératosthène** pour marquer les congruences, ie : les $A \equiv R [P]$ mais en partant
d'un index en fonction du $R \in R_{2n}$ selon la définition (3_) ; déterminé par le programme de l'algorithme.

[« **a_)**: Principe de base de la fonction du crible dans l'ensemble des les entiers naturels de 1 à n, on établit un
tableau de 1 représentant les entiers naturels, on fixe la limite n:

On calcule le reste **R** de $2n$ par $P \leq 2n$.

On part de ce reste **R** où l'on aura remplacé ce 1 par 0, puis par pas de **P** on remplace les 1 par 0 : de **R** à n.

(«Principe Ératosthène»)

«Si $R = 0$ on part **directement de P** qui sera marqué 0».

On aura marqué 0, tous les entiers congruents **[P]** avec $2n$, de cet ensemble d'entiers. Tous les 1 restants sont
donc **non congruents** à **P**, ils seront dénombrés par la fonction du crible, impliquant le nombre de nombres
premiers **q** de n à $2n$.

D'où : un **1**, $\Rightarrow b = q$ premier et **complémentaire**, appartenant à **[n ; 2n]** ayant pour antécédent un entier
appartenant à **[1 ; n]**.

ils formeront par conséquent en re criblant Ératosthène, si et seulement si : **1**, = **p' non congruent [P]** les couples
 $p' + q = 2n$, quel que soit $n = 15k + i > 15$, dans ce principe de base.

Les 2 cribles pour une limite n fixée vérifient donc : $2n = p' + q$. Ou directement avec le crible **EG** en fin de
document.»]

8a_) : Principe du fonctionnement de l'**AG** par **Fam(i)** modulo 30 en fonction de la limite $n \geq 150$:

Sans perte de généralité une condition suffisante est le criblage des **A d'une seule Fam(i)**.

La limite $n = 15k + i$ étant fixée ; on fixe la **Fam(i)**. « en fonction d'une des 15 formes de n : Voir annexe1 ci-
dessous page 9 ».

On établit un tableau de **[1]** de 1 à $n//30$.

Dans le programme de **AG**, on appelle les nombres $P \in P_{2n}$ « que l'on peut indiquer », criblés préalablement par
Ératosthène en début de programme.

Puis on calcule les restes $R \in R_{2n}$ de la division de $2n$ par $P \in P_{2n}$.

Si **R** est pair, on calcule: $J = R + P$ puis $J + 2P + \dots$. Si **R** est impair, $J = R$; puis $J + 2P + \dots + 2P$; jusqu'à $j \% 30$
 $\equiv \text{Fam}(i)$: «La **Fam(i)** qui a été fixée par rapport à la forme de n.»

si $j \% 30 \equiv \text{Fam}(i)$; on calcule l'index : $j//30 = \text{idx}$

Puis en partant de cet index que l'on a marqué 0, on crible les $A \in A_n$ (« congruent à **P** ou suivant l'équivalence :
 $A \equiv R [P]$ ») par pas de **P**, de l'**idx** à $n//30$: en remplaçant le 1 par 0. «ce qui revient à cribler modulo $P*30$ par
Fam(i), tel qu'illustré ci-dessous.»

On va détailler les différentes parties du criblage par l'**AG**,

(« Ces $A \in A_n$ sont aussi préalablement criblés par le **CE** mais avec $p \in P_n$, donc modulo $p*30$.

Selon le même principe mais où on crible les multiples de $p \in P_n$ avec le programme du **CE** que le lecteur
comprend »)

Fixons la **fam(i) 7[30]** ; limite $n = 15k + 7 = 907$; $2n = 30k + 14 = 1814$.

Pour $P_1 = 7$, on a $R_1 = 1$ impair, donc j et $j + 2P + 2P \dots \rightarrow 127$ où $127 \% 30 \equiv \text{Fam}(7)$ et 127 est congruent à **[7]** ;
 $127 \equiv 1[7]$ et donc : $1814 \equiv (127 + 210)[7] \dots + 210) \dots$ congruent **[7]** .

(« cette Fam(i) est donc suffisante pour valider $n = 15k + 7$, c'est à dire $2n = 30k + 14$, sachant que l'on pourrait disposer des deux autres Fam(i) = 1[30] et / ou 13[30] qui sont complémentaires afin de valider la CG pour $2n = 30k + 14$. »)

Étape n°1_) illustration :

lorsque l'on crible ces A de 7 à n, ils sont représentés par des [1111111...] de $7 \rightarrow n // 30$

Ces A $\in A_n$ sont les mêmes pour Ératosthène de $7 \rightarrow n // 30$ contenant les nombres premiers P' $\in P'p : [1,]$ et leurs multiples marqués [0,]

Ce qui donne pour l'exemple ci-dessous : cette image de 7 à n // 30 extraite du document, où la fonction G a remplacé le 1 par 0,

par pas de P si A est congruent à P, depuis l'idx jusqu'à n // 30 relative à la ligne n°1 ci-dessous.

étape n°1_) En gras les A $\in A_n : 7, 97, 127$ marqué 0, car congruents [7] criblé par les nombres P $\in P_{2n}$

[7, 37, 67, 97, 127, 157, 187... etc... modulo 30 $\rightarrow n // 30$]

n° 1 : [0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1] $15k + 7$ fonction de l'AG

étape n°2_)

la ligne n°2 sont les A $\in A_n$ de la même fam(i) avec la même limite ; mais criblé par les nombres p $\in P_n$ du CE de $7 \rightarrow n // 30$.

[7, 37, 67, 97, ... mod 30... $\rightarrow n // 30$]

n°2 : [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] $15k + 7$: fonction E.

La fonction CE remplace le 1 par 0 les multiples de p ; par pas de p $\in P_n$, (« On connaît ce principe, pour le CE avec P_n et non P_{2n} »).

(« L'ordre du criblage entre n°1 et n°2 n'a aucune importance, « sauf en programmation où : on crible avec Ératosthène puis on re crible avec Goldbach » ; cela permet de voir les 2 fonctions du criblage. Le résultat de la fonction n°3, est tout simplement la fonction n°1 superposée sur n°2 afin de voir l'image des A congruents [P] marqués en rouge ou pas si les 1, ou les 0, dans l'AG sont non congruent [P], ces 0 multiples de P_n , mais non congruent [P] sont tout autant important suivant l'égalité du point 6a_), si ils précèdent un $A+30 = 1 =$ premier P' dans Ératosthène »).

n° 1 : [0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1] $15k + 7$ fonction G

n°2 : [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] $15k + 7$: fonction E

n°3: [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] $15k + 7$; 8 couples P' + q = 2n

Explication étape n°3 :

(« où on peut vérifier la propriété des congruences au rang 7, 8 et 9 pourtant non premiers = 0, mais non congruents à 7 et qui pour les limites n suivantes = $15(k+1) + 7$; $n = 15(k+2) + 7$; $n = 15(k+3) + 7$, ce qui permet d'affirmer que la CG sera vérifiée, dû à ce décalage d'un rang, lorsque n augmente de 15. »)

la ligne n°2 CE est à nouveau criblé n°3, mais par la fonction G, ou plus simplement : il suffit de mettre en rouge chaque élément de la ligne n°2 correspondant au 0 de la ligne n°1 qui \Rightarrow A congruents [P] par AG

(« ex ci-dessus : on superpose la ligne n°1 sur la ligne n°2 ; ce qui donne le résultat ligne n°3 criblée par AG ; Où les 1 restants, sont donc les A du CE, tel que $2n \neq A[P]$ restituant les couples P' + q = 2n . On en déduit une troisième fonction du TNP »)

Note et remarque:

Les **1 non congruents [P]**, dans **n°1**, impliquent les nombres premiers **q** appartenant à **[n ; 2n]** suivant **3_)** . Les **1** **représentent** aussi les nombres premiers **P'** dans **n° 2 du CE** de **[7 à n]**; ainsi que les **1 non congruents [P]** dans **n°3** ; ils vont représenter les couples **(P'+q)** qui **décomposent 2n en somme de deux nombres premiers**, pour la limite **n** fixée avec cette Fam(i).

Or que se passe-t-il si **n** augmente de **15** soit : **15(k+1) + 7 ; 2n = 1844 ; les congruences se décalent modulo 30** sur l'entier suivant selon l'égalité du point **6a_)**, illustré ci-dessous suivant la propriété récurrente de ces congruences.

Illustration de l'égalité 6a_) à partir du deuxième rang, l'**AG** reproduit dans l'intervalle fermé : l'image récurrente ayant vérifiée la conjecture précédemment qui s'est décalé d'un rang, ayant une influence négligeable sur le résultat du nombre de couples :

P' + q = 2n + 30 :

n° 4 : (0,) [0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1] **15(k+1) + 7** , décalage d'un rang de **n°1**

n°5 : (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] c'est la même image que **n°2**

n°6 : (1,) [1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1] **15(k+1) + 7** on a répliqué l'image **n°3** avec le décalage d'un rang relatif au criblage **n°3**, de ce fait **au rang 9** le **1** qui était congru pour la limite **n = 15k + 7** devient **1 non congru** pour la limite **n = 15(k+1) + 7**, c'est-à-dire : **on réplique l'image ayant vérifiée la conjecture précédemment mais décalé d'un rang.**

Avec comme conséquence dans ce cas précis ; **une variation négligeable** du nombre de couples **P' + q = 2n. 9**, du simple fait **que seul** le premier élément dans l'**AG** des lignes **n°4** et **n°6** est inconnu au niveau de sa congruence avec le même nombre d'éléments **P** qui criblent.

L'AG étant récursif, pour **15(K+2) + 7** on peut donc prédire à nouveau les couples qui vont vérifier la CG, et affirmer que la CG sera aussi vérifiée pour ce troisième cas et cette limite **n = 15(K+2) + 7 et 2n = 1874** . Que l'on peut illustrer ligne **n°7** simplement, sans même tenir compte du premier élément marqué **X** et du deuxième élément = **0,**)

n° 7 décalé de 2rangs : [x , 0[, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1,] **on perd le dernier élément**

n°8 : [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,]**X**] encore 9 couples **P' + q**

On comprend l'intérêt de cet AG où on note l'intérêt des **0 non premier mais qui sont non congruent à P** précédant un nombre **premier P' = 1** qui avec la fonction décalant les congruences d'un rang sur leurs successeurs **A+30**, permet de donner des couples **(P' + q)** vérifiant la CG et aussi d'en déduire une fonction asymptotique qui ne peut être nulle, pour estimer le nombre de couples qui vérifie la conjecture quel que soit **n = 15(K+1) + i** avec sa **Fam(i)**.

On peut montrer et illustrer cette propriété, si par exemple **n** augmente de **1**, tel que défini dans ces 3 exemples ; soit **n = 15k + 8**

Sachant que jusqu'à **n = 15k + 7** la conjecture à été vérifiée. On prend par conséquent **(n=15(k-1) + 8) < (n = 15k + 7)** et sa **Fam(i)** correspondant à « **30k + 16** »: **Fam 17 ou 29 ; ou Fam 23 + Fam 23** ; dont la somme des deux éléments donnera **30k+16**.

Soit **n = 15(k-1) + 8) = 908 -15 = 893 ; 2n = 1786, Fam(i) , i = 23**

«quel que soit l'une des 3 Fam(i), cela ne modifiera que de façon négligeable le résultat, la densité de couples est la même pour les 8 Fam(i) en moyenne générale ! (Même densité de premiers dans ses Fam(i) en progression arithmétique de raison 30, suivant Dirichlet.)»

Illustration :

A : «qui par supposition a vérifié la CG pour cette limite N avec fam(23)»

Donnez N: 893 Fam(23) **AG** {13.43.73.103.....etc mod 30 $\rightarrow n//30$ }

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

On va marquer en rouge les A congruents [P] \in **CE**, c'est-à-dire correspondant au **A = 0** de l'**AG**

Donnez N: 893 Fam(23) **CE** {13.43.73.103.....etc mod 30 $\rightarrow n//30$ }

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

«11 couples et 9 couples qui vérifieront pour la limite suivante $2n + 30$; Suivant la propriété et égalité de l'algorithme de Goldbach la Preuve du décalage des congruences est démontrée ci-après, page 13. »

B :

Donnez N: 908 **AG**

crible **G**: [0, [0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 10]

Donnez N: 908 **CE**: il est clair que la propriété de l'**AG** fait ressortir une récurrence jamais étudiée dans la solution de cette CG

crible **E**: [1, [1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 10] « 9 couples qui sont les 9 couples ci-dessus »

C :

Donnez N: 923 **AG**

crible: [1, 0, [0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1] 1] On perd un élément modulo 30

Donnez N: 923 **CE**

crible: [1, 1, [1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1] 0] «10 couples, sans compter le premier élément qui est venu se rajouter !»

[« On montre clairement l'intérêt des antécédents A_n de B_{2n} , notamment pour les $A \neq P'$ dans **CE**, mais non congruents à P qui vont se décaler sur leurs successeur $A+30 = P'$. On notera que le résultat des congruences dans **CE**, se décalera aussi pour afficher l'image précédente ayant vérifiée la CG.

Cette propriété récurrente qui se montre élémentairement, est en principe suffisante pour prouver la CG.

Prédiction faisable sur plusieurs criblages successifs. « Suivant le principe d'une preuve par récurrence. »

Ceci va conduire à une contradiction par l'absurde, en supposant la conjecture fausse.

La deuxième fonction asymptotique du TNP donne l'estimation du nombre de A non congruents [P] page 8 ci-dessous. »]

(« On comprend le principe de fonctionnement de ces deux fonctions **AG** et **CE** où on a vérifié formellement que l'on n'a fait que reproduire une image par récurrence de l'ensemble A_n qui a vérifié la CG pour $n = 15k + 7$, ou , $n = 15(k-1) + 8$ avec la propriété de l'égalité (6a_).»)

De part ce constat on va faire ressortir d'autres contradictions, en utilisant ce principe de fonctionnement et la propriété de l'**AG** dans les congruences:

1_) On suppose la conjecture fausse:

le fonctionnement de l'**AG** et sa propriété, en duo avec le crible d'Ératosthène permet d'en déduire une contradiction,

Supposons que la conjecture soit fausse pour une limite $n = 15(k+1) + i$ et sa Fam(i) fixée :

Dans ces illustrations ci-dessus, lors du troisième criblage ligne $n^{\circ}6$ relatif à $n = 15(k+1) + 7$:

Tous les **1** d'Ératosthène doivent devenir des **0**, ou **marqués en rouge**, c'est à dire $A \equiv R [P]$, par l'application de la **fonction de l'AG**!

Mais on connaît le nombre de **A** tel que $2n \neq A[P]$ précédent un **A+30 premier**, qui va donc vérifier la **conjecture pour la limite suivante $n = 15(k+1) + i$** supposait infirmer la conjecture !

Sans avoir besoin d'utiliser l'**AG**, pour le vérifier en ligne $n^{\circ}6$, ou de même suivant l'illustration **B** et **C**; le décalage d'un rang ligne **4** doit mettre le **0** sur chaque **1** de la ligne du **CE** et de supprimer tous ces **A**, sinon cela va contredire la supposition ! *Ce qui clairement va se montrer impossible.*

D'où au minimum :

Une condition obligatoire est nécessaire, il ne faut pas de **1 premiers** consécutifs ou encore pas de **1** d'Ératosthène congruent **[P]**, qui soit précédé d'un **0 non congruent** à **P** dans l'**AG**. Afin que le décalage d'un rang des **A non congruents à P** de Goldbach ne vienne invalider cette supposition, en se superposant (« se décalant ») sur les **1 premiers** consécutif ou pas d'Ératosthène validant la **CG**.

Ou encore que des **1** d'Ératosthène libérés de leur congruence lors du décalage, deviendraient un **1**, donc non congruent à **P**.

C'est-à-dire : il ne faut pas dans Ératosthène un **0 non congruent [P]** qui précède un **1 = congruent à P**, qui par le décalage viendrait par obligation se décaler sur ce successeur et vérifierait la conjecture, contraire à la supposition.

- **Mais cette propriété, fait en sorte que le cardinal de la fonction $\pi(n)$ 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 cardinale de nombre premiers $[N ; 2n]$ ayant été défini et vérifié lors des deux limites précédentes, ainsi que pour la fonction $G(n)$ et sa fonction d'estimation ci-dessous !**

Or plus **n** tend vers l'infini, il y a une multitude de **A marqué 1 = P'** consécutifs dans **CE**, ainsi que des **A = 0 ≠ P' non congruents [P]** qui précèdent des **1 = P'** et qui vont donc, se décaler lors de la limite $n = 15(k+1) + i$. Il vient que cette supposition serait fausse ! On peut même calculer avec la fonction asymptotique non nulle qui en estime le nombre, lorsque $n \rightarrow \infty$.

L'image **récurrente** de l'**AG** contredit cette première hypothèse. A chaque nouveau criblage lorsque la limite **n** augmente de **15**, on **repart du début** avec tous ces **A premiers** consécutifs dans **CE**, ainsi que dans l'**AG** avec les **A** consécutifs **non congruents [P]** qui vont se décaler d'un rang.

On peut supposer que ce n'est pas suffisant et supposer, qu'à partir d'une certaine limite **n** il n'y a plus de **A** marqués **1** consécutifs $< n$ dans **CE**. Ce qui est absurde car **CE** et l'**AG** recommence depuis le début lorsque **n** augmente de **15**, cela ne laisse aucun choix !

Ainsi que pour cette même raison : si il n'y avait plus de **A marqués ,0, mais non congruent [P]**, précédents un **1 congruent [P]** dans **CE**, qui viendraient se décaler sur ce **1** suivant la limite $n + 15$ contredisant la **supposition**, le décalage des congruences existera.

Ce qui est pour le moins absurde, car le crible recommence chaque fois du début, avec ses premiers consécutifs, ou des ,0, non congruent à P précédents des , 1 premiers, congruents à P dans EC « qui en se décalant libèrent les uns et marquent les autres ».

La fonction de l'AG, ne fait que répliquer une image récurrente ayant vérifiée la conjecture à partir du deuxième rang

Par conséquent cette supposition est totalement insuffisante et ne permet pas de remettre en cause cette contradiction ci-dessus.

Mais admettons, sans aucun argument pour le prouver : où on suppose que ce n'est pas formellement suffisant pour accepter cette preuve ci-dessus de la CG.

Or cette première hypothèse qui contredit l'infirmité de la conjecture, est une conséquence de ce qui suit :

Il faut par conséquent : que tous les A premiers ou pas du CE soient tel que : **2n et A sont congrus [P]** donc marqués en rouge par l'AG, par conséquent **congruents à P**.

On va montrer rigoureusement une deuxième contradiction que nous allons détailler suivante cette supposition d'une CG fautive, pour cette limite $n = 15(k+1) + i$:

- a L'AG doit donc marquer d'un 0 les A marqués 1 d'Ératosthène qui n'était donc pas congruent à P lors de la limite : $n = 15K + i$ du criblage précédent ayant vérifié la CG ; avec les nouveaux R de $2n=30(k+1)$ par $P \in P2n$.
- b Or suite à ce décalage des congruences pour $15(k+1) + i$; l'AG doit aussi et encore, marquer d'un 0 tous les A = (0) qui étaient avec $2n$ congrus [P]. Ce qui est impossible !
Car $2n$ a augmenté de 30, par conséquent les $R \in Rn$ ne sont plus les mêmes, donc A et $2n+30$ ne sont plus égaux [P] ; les A = 0 qui n'étaient pas congrus à P et qui précédaient $A+30 = 1= P'$ pour la limite n précédente, congrus ou pas modulo P, doivent aussi être marquer $A+30 = 1= P'$ pour $n = 15(k+1) + i$, c'est-à-dire congrus à P ; sinon on risque de contredire la supposition.
- c Ce qui est clairement impossible ! «les congruences se décalent d'un rang, ainsi que les index du programme, suite au changement de restes R »
Car : 1) Ce A = 0, qui était non congru à P, implique q premier dans $[n, 2n]$; d'où par récurrence :
- d $(2n+30) - (A+30 = 1= P')$ impliquera toujours le même nombre premier q , avec les nouveaux restes R, et en 2): si ce $A+30 = 1= P'$ était congruent à P précédemment, il est évident qu'il deviendra non congrus à P suivant 1) ; les restes R ayant changés il ne peut plus être congru R [P], par cette propriété récurrente du décalage, le contraire serait absurde suivant 1).
On en déduit que si la conjecture est vérifiée pour $2n$, elle le sera d'autant pour $2n + 2$ suite au décalage.
- e Autrement dit : pour être rigoureusement sûr, il faut marquer d'un 0 l'ensemble An d'Ératosthène, pour la limite $n = 15(k+1) + i$ ou $n = 15(k+2)+ i$; ce qui est absurde !
- f il n'y aurait plus de nombres premiers q de n à 2n, alors qu'ils y en avaient $n/\log 2n$ pour la limite n précédente suivant le TNP et ce, uniquement en augmentant $2n$ de 30.!
- g Il n'y a pas suffisamment de nombres $P \in P2n$ qui criblent et les restes $R \in R2n$ changent obligatoirement, ce qui implique le changement des index notés Idx, qui positionnent le départ de P pour le criblage ; on sait que ceci occasionne le décalage d'un rang des congruences et le nombre de P est limité par la racine de $2n + 30$ qui limite par conséquent le nombre de A congrus [P] qui sera le même +1, pour $n=15(k+1)$

- h** Or les nombres **P** qui criblent, sont les mêmes pour $n = 15(k+1) + i$! « Dans le pire des cas il y en aura qu'un de plus, donc totalement insuffisant et négligeable pour infirmer la CG. Le nombre de premiers : $q \in [n ; 2n]$ ne peut varier que d'un élément lorsque n augmente de 15.»
- i** La division de $2n = (30(k+1)+2i)$ par **P** ne donne qu'un **R** par nombre **P** ; elle ne restitue pas les restes **R** des criblages successifs et précédents, relatif à $n = 15k ; 15(k-1) ; 15(k-2)$ etc.. il est donc impossible de marquer d'un **0** tous les $A \in An$.

On prouve par-là, qu'il est impossible d'infirmer la CG et que l'on ne fait que répliquer un résultat ayant vérifié la conjecture lorsque n augmente de 15 et qui se décale d'un rang; ce qui implique une variation très négligeable du nombre de couples $P' + q = 2n$.

On en déduit que la CG est vraie, simplement en décalant d'un rang les congruences sur leur successeur $A+30$ relatif à la limite $n = 15k + i$ qui a été vérifiée, car les entiers $A \equiv 2n[p]$ qui précèdent un nombre premier P' ont aussi été vérifiés pour la même limite n !

Un corollaire avec le TFA : tout entier naturel strictement positif est congru à $2n$ (modulo P) de manière unique à l'ordre près de ses facteurs

«Le troisième algorithme complet est la fusion des deux algorithmes où le décalage des congruences est moins visible, mais identifiable, le programme sera mis en fin de document, et une annexe dans un autre dossier, qui montre l'image récurrente de Goldbach.»)

En conséquence directe avec le Théorème de Dirichlet, les $A = 1$ d'Ératosthène non congruents $[P]$ étant dans une suite arithmétique de raison 30, il y a une infinité de $A=1$ qui représentent les couples $P' + q = 2n$ par $fam(i)$ avec une même densité à partir de la limite $n = 150$.

*(«Le programme «va dénombrer» calculer le cardinal des **A non congruents** $[P]$ qui \Rightarrow **q** appartenant à $[n ; 2n]$, en marquant les **A congruents** $[P]$ tel que : $2n \equiv A [P] = B$ multiples de **P** ; comme le fait Ératosthène de 1 à n qui marque les multiples de $p \in Pn$ selon le programme de l'algorithme d'Ératosthène modulo 30.*

*On déduit de l'algorithme, une fonction asymptotique, conséquence directe du TNP, permettant d'estimer ce nombre de nombres **A non congruent** $[P]$ qui implique **q premier**, appartenant à $[n ; 2n]$ qui vaut environ :*

$$\frac{n}{(\log 2n) - 0,6875} \text{ lorsque } n \rightarrow \infty. \text{ Nous verrons ci-dessous ces formules.}.)$$

Une fonction heuristique $n / (\log n)^2$ donne une estimation **minimale** de solutions vérifiant la conjecture.

Cette fonction heuristique n'est donc pas nulle suite à cette résolution, elle est une conséquence directe du TNP.

On a une relation entre les nombres premiers $q \in B2n$ et P' qui dépendent de leur congruence... C'est à dire qu'un nombre premier **q** à pour antécédent un **A** de $[7; n]$ tel que $A \equiv [P]$ quel que soit $n[15]$

On en déduit plusieurs fonction asymptotiques nombre de **A** non congruents à **P** ou nombre de premiers **q** de n à $2n$:

$$\frac{n}{(\log .2n) - 0,6875} < \frac{n}{(\log n) - 0,6875} . \text{ Soit : } G(n) < \pi(n).$$

Où : **G(n)** est la fonction qui compte le nombre d'entiers **A** non congrus à $2n[P]$, qui a comme conséquence directe du TNP : la fonction qui compte le nombre de couples $P' + q = 2n$, vaudrait environ :

$\frac{G(n)}{\log G(n)}$ fonction qui estime le nombre de $A = P' \in An$, tel que $2n \neq A[P]$ de l'AG, et par Fam(i).

Le nombre réel de couples $P' + q = 2n$ se situe donc dans l'intervalle borné par ces deux fonctions ci-dessus.

« **Ce qui rend donc possible** l'analyse probabiliste sur un entier A de 7 à n pris au hasard, d'être non congru à $2n [P]$ et de ce fait, il devient inutile de savoir si un nombre premier $q \in B2n$ dépend ou pas de $A = P'$ car c'est son antécédent. »

Théorème : tout nombre pair $2n \geq 6$ peut s'écrire comme la somme de deux nombres premiers ($P' + q$)

Corollaire : tout nombre pair $2n \geq 180$ peut s'écrire comme la somme de deux nombres premiers ($P' + q$) appartenant à une famille Fam(i) tel que définie en 2.)

En information complémentaire : sont donnés les différents indexes (idx) de départ des deux fonctions 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 Voici les indexes de départ de P du crible G ; pour $n = 900 + i$, $i = 7$ et fam = 7 : relatif aux $j\%30 == 7$ « du premier exemple ci-dessous. »

Goldbach :

Ératosthène :

$P = 7$, idx = 4 puis par pas de 7
par pas de 7 et de $31 \rightarrow n//30$

dans Ératosthène $7*31 = j$; $j\%30 == \text{fam}$, partent de idx = 7, puis

$P = 11$, idx = 10 puis par pas de 11
par pas de 11 et de $17 \rightarrow n//30$

dans Ératosthène $11*17 = j$; $j\%30 == \text{fam}$, partent de idx = 6 puis

$P = 13$, idx = 0 puis par pas de 13
par pas de 13 et de $19 \rightarrow n//30$

dans Ératosthène $13*19 = j$; $j\%30 == \text{fam}$, partent de idx = 8 puis

$P = 17$, idx = 3 puis par pas de 17
par pas de 23 et de $29 \rightarrow n//30$

dans Ératosthène $23*29 = j$; $j\%30 == \text{fam}$, partent de idx = 22 puis

$P = 19$, idx = 14 puis par pas de 19

chaque j est donc un produit fam[30] contrairement à Goldbach.

$P = 23$, idx = 15 puis par pas de 23
ne marquent que l'idx de départ.

$31 \rightarrow >$ racine de 907 ne marquera rien, ainsi que 23 et 29 qui

$P = 29$, 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 et en dernier $P = 41$, idx > 914 d'où il ne peut cribler. $P \leq 1814 = 42, \dots$

Dans Goldbach : Si R est pair, $j = R + k*P$. (« voir programme si R est impair $j=R$ puis $+ 2*P \dots \text{etc.}$ »)

Si $j\%30 == \text{fam}$; alors :

$j//30 = \text{début d'index} = \text{idx}$. Puis :

P part de idx, par pas de P où on remplace le 1 par 0 $\rightarrow n // 30$. Ensuite on réitère avec P 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 criblés et $n \geq 150$:

On a $8 \text{ fam} \equiv 1 \text{ ou } P[30]$ avec P appartenant à $[7 ; 29]$

Pour $n = 15k + i \rightarrow \{1,7,11,13,2,4,8,14\}$ coef = 0,375 ; *ce qui donne 3 fam(i) sur 8 qui peuvent être criblées*

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

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

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

Il y a donc 15 formes de n.

Note : on peut démontrer:

[(« Même si suivant le principe du crible d'Ératosthène, 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 ;

Ou 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. B est non congru 0 modulo P

Car si : $2n \not\equiv A[P], 2n - A = B$ ne peut donc être divisible par P . (« Le reste de la division de $2n$ par P , n'est pas égal au reste de la division de A par P ; $2n = Py + A$, où P divise $2n - A$...etc ») **(1)**]

On suppose $2n \not\equiv A[P] \Rightarrow$ il n'existe pas de y tel que $2n - A = Py$,

Donc aucun nombre premier P ne divise $2n - A$, d'où $2n - A = q$ selon la définition **(1)**.

On veut démontrer $2n - A$ premier, on veut démontrer qu'il n'existe pas de $P > \sqrt{2n}$ 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 q premier $> \sqrt{2n}$ car on a démontré **(1)** . Donc prenons

P et q les plus petits possibles, soit $\sqrt{2n}$.

On a $(\sqrt{2n})^2 = 2n$, or $2n - A \leq 2n - 1$, **Ce qui est impossible. Conclusion $2n \not\equiv A[P] \Rightarrow 2n - A = q$ premier, car P ne divise pas $2n - A$ »] .**

Preuve du décalage des congruences :

On veut montrer simplement, la propriété récurrente de l'AG, lorsque l'on crible une suite arithmétique de raison r , pour une limite n ; en utilisant les congruences.

Les congruences se décalent d'un rang lorsque n augmente de r dans l'intervalle fermé, fixé par la limite n , ainsi que du nombre d'éléments $P \in P_{2n} \leq \sqrt{2n}$, donc fixés par la racine de $2n$.

Prenons la suite U_n : avec $a' \in [1, n]$

$P \in P_{2n} \leq \sqrt{2n}$; $P_0 = 2$. Le reste R de 4 par 2 = 0, ce qui implique que $a' = 1$ est non congruent à 2

$U_0 = 4$, soit : $4 \not\equiv a'[2] \Leftrightarrow 4 \not\equiv 1[2]$

Donc **1** est non congruent à P soit : $4 \not\equiv 1[2]$; $a'=1$ et $4 - 1 = 3$ qui n'est pas divisible par 2 donc premier $q \in [n, 2n]$

$U_{n+1} = 5$, la congruence se décale d'un rang $a' + 1 = 2$, $5 \not\equiv 2[2]$ et $5 - 2 = 3$ qui est le même nombre premier non divisible par 2

$U_{n+2} = 6$, décalage d'un rang ; $a' + 1 = 3$ et $6 \not\equiv 3[2]$ et $6 - 3 = 3$ non divisible par 2

$U_{n+3} = 7$, décalage d'un rang ; $a' + 1 = 4$ et $7 \not\equiv 4[2]$, $7 - 4 = 3$ non divisible par 2

$U_{n+4} = 8$, décalage d'un rang ; $a' + 1 = 5$ et $8 \not\equiv 5[2]$, $8 - 5 = 3$ non divisible par 2

$U_{n+5} = 9$, décalage d'un rang ; $a' + 1 = 6$ et $9 \not\equiv 6[2]$, $9 - 3 = 3$ non divisible par 2

On va créer un autre intervalle fermé avec racine de 9 et $P = 3$

(« $U_n + 5 = 9$, avec $P_2 = 3$; $a' + 1 = 7$ et $9 \neq 7$ [3], $9 - 7 = 2$ qui est non divisible par 3, mais sans intérêt avec ce nouveau P_2 car c'est la même conséquence.»)

L'algorithme de Goldbach a donc une propriété récurrente qui est démontrée en utilisant les congruences.

Ce que l'on a illustré avec **L'AG**, page 6, selon la propriété définie et montrée au point 6a_)

Lefeu gilbert le 23 /03/2018.

Annexe I:

La fonction 2 du théorème de Goldbach est une conséquence directe du TNP: ($\log = \text{logarithme naturel}$)

$G(2n)$: la fonction de compte du nombre de nombres premiers $q \in [n ; 2n]$

$$G(2n) \text{ vaut } \sim \lim_{n \rightarrow +\infty} i \frac{n}{(\log 2n)}$$

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

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

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

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

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

Le nombre de couples de nombres premiers (p', q) ; vaut environ lorsque $\lim_{n \rightarrow +\infty} i$ pour $n = 15k + i$, avec $i \in [0 ; 14]$ et k entier naturel non nul.

$$\frac{\left(\frac{n}{\ln n}\right)}{\ln\left(2 * \left(\frac{n}{\ln n}\right)\right)} \text{ ou simplement } \frac{\pi(n)}{\ln \pi(n)} \text{ mais plus précisément : } C_2 \frac{G(n)}{\ln G(n)} ;$$

où $C_2 \approx 1,320323$ constante des premiers jumeaux.

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

On a donc calculé avec Ératosthène le nombre de premiers $P' \in [7 ; n]$ pour la fam(i) qui sera criblée par l'AG ; pour ensuite appliquer cette fonction, qui n'est aussi qu'une conséquence du TNP. Il faut donc utiliser la fam (i) correspondante à la fam(i) complémentaire qui décomposent $2n$ sans les multiples de {2, 3, et 5}. Exemple pour la limite $n = 15k + 7$, il n'y a que trois familles qui décomposent $2n = 30k + 2$.

La fonction sera donc utilisée avec le coefficient de $0,375 = 3/8$.

$$\lim_{n \rightarrow +\infty} \dot{\iota} \frac{G(n)}{\ln G(n)} * C_2 * 0,375 .$$

Exemple pour $n = 496$, $2n = 992$, nombre de premiers $[n ; 2n]$ des 3 familles {1, 13 et 19} = 33. Alors que $\pi(2n - n)$ vaut : 73

Résultat :

$$(33 / \ln 33) * 1,320323 = 12, \dots \text{ couples, pour un réel de 13.}$$

$$(73 / \ln 73) * 0,375 * 1,320323 = 8, \dots$$

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

$$\lim_{n \rightarrow +\infty} \dot{\iota} \frac{\pi(n)}{\ln \pi(n)} * C_2 * 0,5$$

Exemple $2n = 1000\ 000\ 010$ 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 : 1 891 734 couples < 2 422 662 réels

*

Annexe 2 :

Les deux programmes en Python : **Ératosthène $15k + i$** en fonction de la famille choisie par rapport à $2N$.

Attention de ne pas se tromper de fam (i) !

En effet il n'y a pas les $P' = 2, 3$ et 5 ni leur multiples. Si on se trompe de Famille, et que la Famille complémentaire sont des multiples de 3 , ou 5 il est évident que le résultat sera faux, du fait que l'on crible dans des fam(i) en progression arithmétique de raison 30 . Par conséquent les multiples de $2,3,5$ étant absents ils ne peuvent pas être criblés.

Pour changer de famille, on change la variable (**i**) à la fin du programme ci-dessous :

On crible

E_Crible (premiers, n, 7) **# on change le dernier terme 7 ; par {1.11.13.17.19.23.29}**

from itertools import product

from time import time

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)
    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))
    #print(f"On prend N = {n} (30 * {int(n/30)})")
    return n

```

```

def lprint(text="", liste=None):
    if len(liste) < 250:
        print(text + str(liste))

```

```

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,
                # On crible directement à partir de l'index
                for idx in range(index, lencrible, a): # index qui est réutilisé ici...
                    crible[idx] = 0
            #print(j)

    total = sum(crible)
    lprint("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)
    #lprint("premiers:", premiers)
    #print(f"nombres premiers entre 7 et n: {len(premiers)}")

    start_time = time()
    # On crible
    E_Crible(premiers, n, 7) # pour changer de Fam changer le dernier paramètre (1,11,...29)
    temps = time()-start_time
    print(f'--- Temps total: {int(temps*100)/100} sec ---')

main()

```

*******Goldbach : $n = 15k + i$ en fonction de la famille choisie**, ne pas se tromper !

Pour Goldbach : crible _G.T.Y_mod30 :

Exemple :

Je veux cribler $n = 15k + 1$; soit $2n = 30k + 2$
 $30k + 2 =$ soit $1+31$ donc fam 1 ; ou $13+19$: donc fam 13 donne $q = 19[30]$ *et inversement*.

Pour $n = 15k + 0$ chaque fam peut cribler les entiers A non congruent [P] ; q complémentaire = $30k$
 Fam 17, donne $q = 13[30]$ *et inversement*. Fam 11, donne $q = 19[30]$ *et inversement*...etc

```

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)
    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))
    #print(f"On prend N = {n} (30 * {int(n/30)})")
    return n

```

```

def lprint(text="", liste=None):
    if len(liste) < 200:
        print(text + str(liste))

```

```

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

    # On g n re un tableau de N/30 cases rempli de 1
    crible = n//30*[1]
    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
        # 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'index
        reste //= 30
        # On crible directement avec l'index
        for index in range(reste, lencrible, premier):
            crible[index] = 0

```

```

total = sum(crible)
lprint("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  $\sqrt{2N}$ 
```

```
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) # pour changer de fam changer le dernier paramètre (1,11,...29)
```

```
temps = time()-start_time
```

```
print(f"--- Temps total: {int(temps*100)/100} sec ---")
```

```
main()
```

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

```
*****
```

Annexe 3 :

ces deux programme en C++ utilisé avec code:bloc est l'Application Windows 64bits

Eratosthène :

```

-----
// -*- compile-command: "/usr/bin/g++ -g goldbach.cc" -*-
#include <vector>
#include <iostream>
#include <cmath>
#include <stdlib.h>
#include <time.h>
using namespace std;
// fill Erathosthene sieve crible for searching primes up to 2*crible.size()*32+1
// crible is a (packed) bit array, crible[i] is true if 2*i+1 is a prime
// crible must be set to true at startup
void fill_crible(vector<unsigned> & crible,unsigned p){
    crible.resize((p-1)/64+1);
    unsigned cs=crible.size();
    unsigned lastnum=64*cs;
    unsigned lastsieve=int(std::sqrt(double(lastnum)));
    unsigned primesieved=1;
    crible[0] = 0xffffffe; // 1 is not prime and not sieved (2 is not sieved)
    for (unsigned i=1;i<cs;++i)
        crible[i]=0xffffffff;

```

```

for (;primesieved<=lastsieve;primesieved+=2){
// find next prime
unsigned pos=primesieved/2;
for (;pos<cs;pos++){
if (crible[pos/32] & (1 << (pos %32)))
break;
}
// set multiples of (2*pos+1) to false
primesieved=2*pos+1;
unsigned n=3*primesieved;
for (;n<lastnum;n+=2*primesieved){
pos=(n-1)/2;
crible[(pos/32)] &= ~(1<<(pos %32));
}
}
}
unsigned nextprime(vector<unsigned> & crible,unsigned p){
// assumes crible has been filled
++p;
if (p%2==0)
++p;
unsigned pos=(p-1)/2,cs=crible.size()*32;
if (2*cs+1<=p)
return -1;
for (;pos<cs;++pos){
if (crible[pos/32] & (1<<(pos%32))){
pos=2*pos+1;
// if (pos!=nextprime(int(p)).val) CERR << "error " << p << endl;
return pos;
}
}
return -1;
}

typedef unsigned long long ulonglong;

size_t ECrible(const vector<ulonglong> & premiers,ulonglong n,int fam){
int cl=clock();
size_t lencrible=n/30,nbpremiers=premiers.size();
vector<bool> crible(lencrible,true);
// ulonglong n2=2*n;
vector<ulonglong> indices(nbpremiers);
for (size_t i=0;i<nbpremiers;++i){
ulonglong p=premiers[i];
ulonglong produit;
int GM[]={7,11,13,17,19,23,29,31};
for (size_t j=0;j<sizeof(GM)/sizeof(int);j++){
produit = p*GM[j];
if (produit %30==fam){
produit /= 30;
break;
}
}
indices[i]=produit;
}
ulonglong nslices=lencrible/1500000,currentslice=0;
if (nslices==0) nslices=1;
for (;currentslice<nslices;++currentslice){
size_t slicelimit=currentslice+1;
slicelimit=slicelimit==nslices?lencrible:(currentslice+1)*(lencrible/nslices);
for (size_t i=0;i<nbpremiers;++i){
ulonglong p=premiers[i];
size_t index;
for (index=indices[i];index<slicelimit;index+=p)
crible[index]=0;
indices[i]=index;
}
}
size_t total=0;
for (size_t index=0;index<lencrible;++index)

```

```

    total += int(crible[index]);
    cout << "Nombre premiers criblés famille " << fam << " plus petits que "<< n <<": " << total << " time " << (clock()-cl)*1e-6<< endl;
    return total;
}

int main(int argc, char ** argv){
    vector<unsigned> crible;
    unsigned long long N;
    int fam=1;
    if (argc>1){
        N=atoll(argv[1]);
        if (argc>2)
            fam=atoi(argv[2]);
    }
    else {
        cout << "Syntaxe " << argv[0] << " N fam. Donnez N puis fam: ";
        cin >> N;
        cin >> fam;
    }
    double sqrtN=unsigned(std::sqrt(double(N)));
    fill_crible(crible, sqrtN);
    vector<unsigned long long> premiers;
    for (unsigned long long p=7; p<=sqrtN;){
        premiers.push_back(p);
        p=nextprime(crible, p);
        if (p==unsigned(-1))
            break;
    }
    ECrible(premiers, N, fam);
    cin >> N;
}

```

Goldbach en C++:

```

// -*- compile-command: "/usr/bin/g++ -g goldbachs.cc" -*-
#include <vector>
#include <iostream>
#include <cmath>
#include <stdlib.h>
#include <time.h>
using namespace std;
// fill Erathosthene sieve crible for searching primes up to 2*crible.size()*32+1
// crible is a (packed) bit array, crible[i] is true if 2*i+1 is a prime
// crible must be set to true at startup
void fill_crible(vector<unsigned> & crible, unsigned p){
    crible.resize((p-1)/64+1);
    unsigned cs=crible.size();
    unsigned lastnum=64*cs;
    unsigned lastsieve=int(std::sqrt(double(lastnum)));
    unsigned primesieved=1;
    crible[0] = 0xffffffe; // 1 is not prime and not sieved (2 is not sieved)
    for (unsigned i=1; i<cs; ++i)
        crible[i]=0xffffffff;
    for (; primesieved<=lastsieve; primesieved+=2){
        // find next prime
        unsigned pos=primesieved/2;
        for (; pos<cs; pos++){
            if (crible[pos/32] & (1 << (pos %32)))
                break;
        }
        // set multiples of (2*pos+1) to false
        primesieved=2*pos+1;
        unsigned n=3*primesieved;
    }
}

```

```

for (;n<lastnum;n+=2*primesieved){
    pos=(n-1)/2;
    crible[(pos/32)] &= ~(1<<(pos %32));
}
}
}
unsigned nextprime(vector<unsigned> & crible,unsigned p){
// assumes crible has been filled
++p;
if (p%2==0)
    ++p;
unsigned pos=(p-1)/2,cs=crible.size()*32;
if (2*cs+1<=p)
    return -1;
for (;pos<cs;+pos){
    if (crible[pos/32] & (1<<(pos%32))){
        pos=2*pos+1;
        // if (pos!=nextprime(int(p)).val) CERR << "error " << p << endl;
        return pos;
    }
}
return -1;
}
}

```

```
typedef unsigned long long ulonglong;
```

```

size_t GCrible(const vector<ulonglong> & premiers,ulonglong n,int fam){
int cl=clock();
size_t lencrible=n/30,nbpremiers=premiers.size();
vector<bool> crible(lencrible,true);
ulonglong n2=2*n;
vector<ulonglong> indices(nbpremiers);
for (size_t i=0;i<nbpremiers;++i){
    ulonglong p=premiers[i];
    ulonglong reste=n2 % p;
    if (reste %2==0)
        reste += p;
    ulonglong pi2=2*p;
    while (reste %30!=fam)
        reste += pi2;
    reste /= 30;
    indices[i]=reste;
}
ulonglong nslices=lencrible/150000,currentslice=0;
if (nslices==0) nslices=1;
for (;currentslice<nslices;++currentslice){
    size_t slicelimit=currentslice+1;
    slicelimit=slicelimit==nslices?lencrible:(currentslice+1)*(lencrible/nslices);
    for (size_t i=0;i<nbpremiers;++i){
        ulonglong p=premiers[i];
        size_t index;
        for (index=indices[i];index<slicelimit;index+=p)
            crible[index]=0;
        indices[i]=index;
    }
}
size_t total=0;
for (size_t index=0;index<lencrible;++index)
    total += int(crible[index]);
cout << "Nombre premiers criblés famille " << fam << " entre "<< n << " et " << n2 << ": " << total << " time " << (clock()-cl)*1e-6<< endl;
return total;
}
}

```

```

int main(int argc,char ** argv){
vector<unsigned> crible;
ulonglong N;
int fam=1;
if (argc>1){
    N=atoll(argv[1]);
    if (argc>2)

```

```

    fam=atoi(argv[2]);
}
else {
    cout << "Syntaxe " << argv[0] << " N fam. Donnez N puis fam: ";
    cin >> N;
    cin >> fam;
}
double sqrt2N=unsigned(std::sqrt(2*double(N)));
fill_crible(crible,sqrt2N);
vector<ulonglong> premiers;
for (ulonglong p=7;p<=sqrt2N;){
    premiers.push_back(p);
    p=nextprime(crible,p);
    if (p==unsigned(-1))
        break;
}
GCrible(premiers,N,fam);
cin>>N;
}

```

Crible de Goldbach EG2 complet en Python : *fusion des deux algorithmes*

Lefeu gilbert le 25/03/2019 et Rolland Coquard qui a fait tous les programmes Python

```

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) # (si on fusionne les deux cribles il faudra rentrer, int((2n)**0.5)
pour Goldbach.

```

```

    prime_E,prime_EG=[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]:
            if each_number>n1:
                prime_EG.append(each_number)
            else:
                prime_E.append(each_number)
                for multiple in range(each_number*each_number, n, each_number):
                    sieve_list[multiple] = False
    #print(prime_E,prime_EG)
    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
    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 à partir de l'index avec pi
        for index in range(reste, lencrible, premier):
            crible[index] = 0
    total = sum(crible)
    #print(nbpremiers)
    #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

```


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.

Ce qui est différent avec l'algorithme de Goldbach, puisque $n - m$ dépend de la congruence de m et où la fonction $(n/2) / \ln n$, qui donne environ $G(2n)$ fonction qui compte le nombre de nombre premiers $n - m$; est une conséquence du TNP.

Si l'on poursuit quand même ce raisonnement heuristique, on peut estimer que le nombre total de manières d'écrire un grand nombre entier pair n comme la somme de deux nombres premiers impairs vaut environ pour $m > 3$, $\approx n / 2 * \ln^2 n$