

# Prime vs Random Sets

Sebastien Plaasch

Maxime Rubio

Lucas Villiere

30 mai 2020

## Résumé

Resume de notre projet..

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>4</b>
	Introduction	4
<b>2</b>	<b>Creation d'ensembles aléatoires suivant la distribution de <math>\pi(x)</math></b>	<b>5</b>
2.1	Approche analytique . . . . .	5
2.1.1	Méthode de création d'un ensemble aléatoire $Q$ . . . . .	5
2.1.2	Définition de la fonction $\sigma$ . . . . .	5
2.1.3	Écart entre $\pi(x)$ et $\sigma(x)$ . . . . .	5
2.1.4	Le Théorème des Nombres Premiers . . . . .	5
2.2	Approche probabiliste . . . . .	8
2.2.1	Ensembles aléatoires . . . . .	9
2.2.2	Ensembles probabilistes impairs . . . . .	10
2.3	Ressemblance et différences entre ensembles aléatoires et nombres premiers . . . .	12
<b>3</b>	<b>Ensembles aléatoires et conjectures</b>	<b>13</b>
3.1	Les nombres premiers jumeaux . . . . .	13
3.2	Seconde conjecture . . . . .	17
3.2.1	Introduction . . . . .	17
3.2.2	Analyse . . . . .	17
3.2.3	Conclusion de l'analyse . . . . .	20
<b>4</b>	<b>Conclusion</b>	<b>20</b>
<b>A</b>	<b>Appendice</b>	<b>21</b>
A.1	Code : Création d'ensembles aléatoires . . . . .	21
A.1.1	Création d'ensembles probabiliste . . . . .	21
A.1.2	Création d'ensembles probabiliste impairs . . . . .	22
A.1.3	Création d'ensembles probabiliste impairs . . . . .	23
A.2	Seconde conjecture . . . . .	24
A.2.1	Raport de conjecture . . . . .	24
A.3	Test conjecture . . . . .	26
A.3.1	Error mapping . . . . .	27

# 1 Introduction

Ici on fait l'introduction

## 2 Creation d'ensembles aléatoires suivant la distribution de $\pi(x)$

### 2.1 Approche analytique

Cette approche pour créer des ensembles aléatoires (désignés par  $Q$ ) qui partagent la même distribution que celle des nombres premiers, est basée sur un théorème (théorème 1) issu de ... .

**Théorème 1.** *L'hypothèse de Riemann est équivalente à l'assertion*

$$\forall n \geq 11, |p_n - \text{ali}(n)| < \frac{1}{\pi} \sqrt{n} \log^{5/2}(n)$$

où  $p_n$  représente le  $n$ -ième nombre premier.

#### 2.1.1 Méthode de création d'un ensemble aléatoire $Q$

Les onze premiers éléments d'un ensemble  $Q$  sont choisis arbitrairement. Pour  $n > 11$ , voici la méthode de sélection de l'élément  $q_n \in Q$  :

- on pose  $a = \max \left\{ q_{n-1}, \lceil \text{ali}(n) - \frac{1}{\pi} \sqrt{n} \log^{5/2}(n) \rceil \right\}$  (où  $\lceil x \rceil$  désigne la partie entière supérieure de  $x$ );
- on pose  $b = \lfloor \text{ali}(n) + \frac{1}{\pi} \sqrt{n} \log^{5/2}(n) \rfloor$  (où  $\lfloor x \rfloor$  désigne la partie entière inférieure de  $x$ );
- $q_n$  est choisi aléatoirement entre  $a$  et  $b$ .

En procédant de la sorte, le théorème 1 sera toujours vrai pour tout ensemble aléatoire  $Q$ .

Par cette méthode, nous avons créé 200 ensembles, jusqu'à  $10^7$ , dont pour 100 d'entre-eux on a imposé la condition suivante :  $\forall q_n \in Q$  tel que  $n > 11$  :  $q_n$  est impair.

#### 2.1.2 Définition de la fonction $\sigma$

On peut désormais définir  $\sigma : [0, \infty[ \rightarrow \mathbb{R}, x \mapsto \# \{n \in Q : n < x\}$ . Nous parlerons systématiquement de la fonction  $\sigma$  alors que cette fonction n'est bien entendu pas unique, elle dépend à chaque fois de l'ensemble aléatoire  $Q$  sur lequel on travail. Cependant, nous avons pu remarquer que les différentes fonctions  $\sigma$  sont souvent très proches les unes des autres. À titre d'exemple, pour un grand nombre d'ensembles aléatoires  $Q$ , nous avons calculé  $\sigma(1000)$ . Pour 45% des ensembles,  $\sigma(1000) = 148$  et parmi 42% d'entre eux,  $\sigma(1000) = 147$ .

Afin de visualiser  $\sigma(x)$  en la comparant à  $\pi(x)$  et  $\frac{x}{\log(x)}$ , voici leur graphe respectif :

La figure 1 devrait apparaître ici.

#### 2.1.3 Écart entre $\pi(x)$ et $\sigma(x)$

La fonction  $\sigma$  semble suivre la même allure que  $\pi(x)$ . Cependant, lors de nos expérimentations, nous avons dessinés des graphes (que vous trouverez en annexe) pour des valeurs de  $x$  inférieures à celles de la figure 1. Pour des petites valeurs de  $x$ , la courbe de  $\sigma$  était presque confondue avec celle de  $\frac{x}{\log(x)}$ . Lorsque les valeurs de  $x$  sont de plus en plus grandes,  $\sigma(x)$  tend vers  $\pi(x)$ . Pour analyser l'écart entre  $\pi(x)$  et  $\sigma(x)$ , nous avons tracé le graphe de la fonction  $\frac{\pi(x)}{\sigma(x)}$  :

La figure 2 devrait apparaître ici. On peut en conclure que  $\pi(x) \sim \sigma(x)$ .

#### 2.1.4 Le Théorème des Nombres Premiers

L'objectif de cette section est de prouver la fidélité de nos ensembles aléatoires  $Q$  à la répartition des nombres premiers. Pour ce faire, nous allons vérifier si le Théorème des Nombres Premiers, cité ci-après, est vrai quand on remplace  $\pi(x)$  par  $\sigma(x)$ .

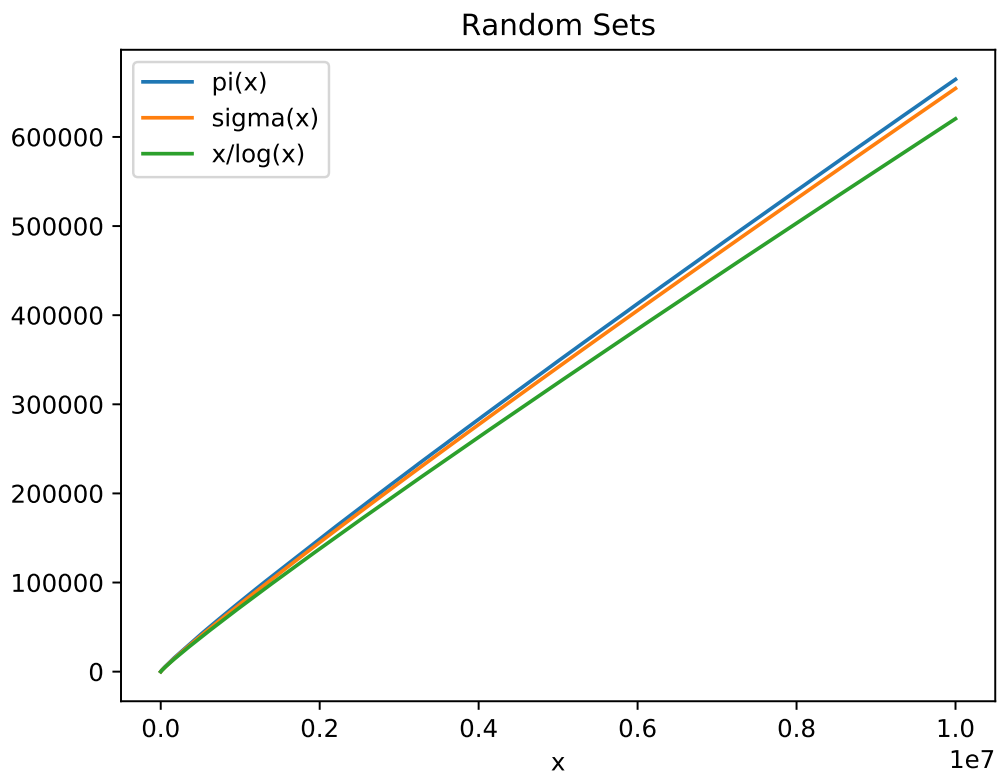


FIGURE 1 – Graphes de  $\sigma(x)$ ,  $\pi(x)$  et  $\frac{x}{\log(x)}$

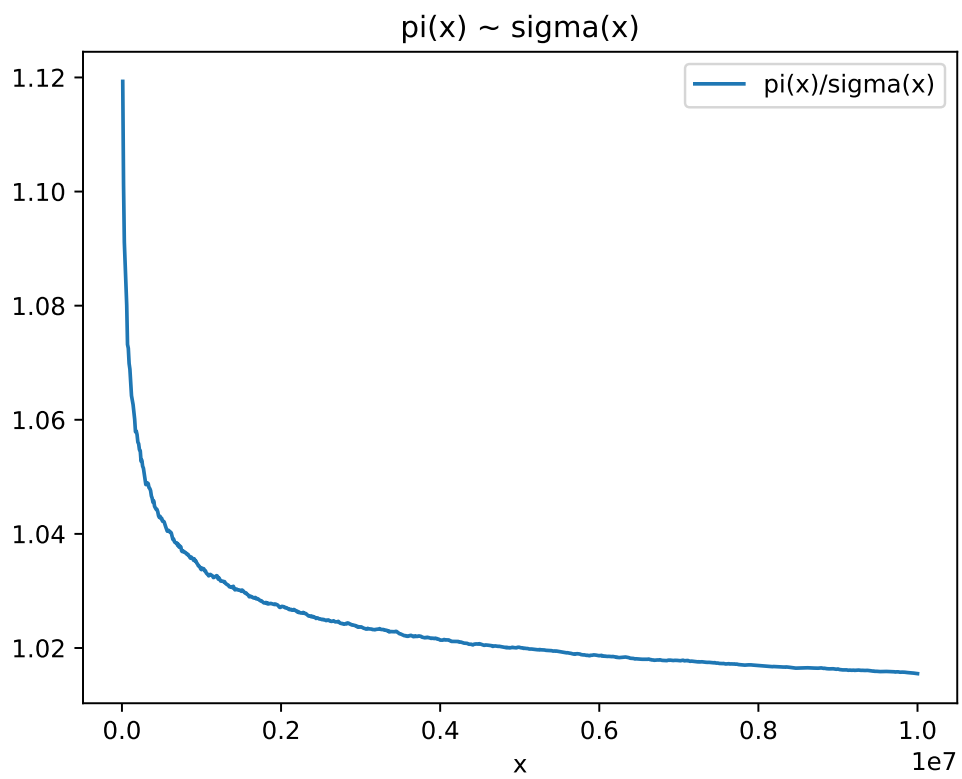


FIGURE 2 – Graphe de  $\frac{\pi(x)}{\sigma(x)}$

**Théorème 2** (Théorème des Nombres Premiers). *Quand  $x$  tend vers l'infini :*

$$\pi(x) \sim \frac{x}{\log(x)}$$

Pour démontrer le Théorème des Nombres Premiers, il a été démontré que, quand  $x$  tend vers l'infini,  $Li(x) \sim \frac{x}{\log(x)}$ . Nous pouvons montrer, graphiquement, que lorsque  $x$  tend vers l'infini,  $\sigma(x) \sim Li(x)$  (voir figure 3), ce qui implique que  $\sigma(x) \sim \frac{x}{\log(x)}$ .

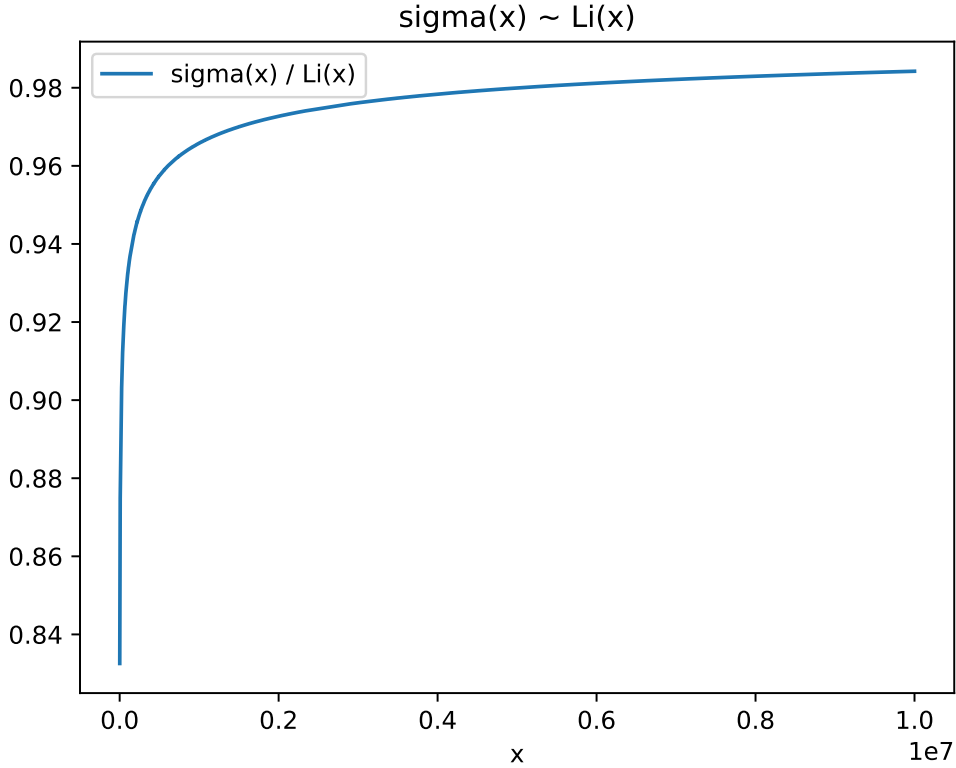


FIGURE 3 – Graphe de  $\frac{\sigma(x)}{Li(x)}$

## 2.2 Approche probabiliste

Soient  $E_n := \{k \in \mathbb{N}^* \mid k < n\}$  l'ensemble des entiers inférieurs à  $n$ ,  $P_n := \{k \in E_n \mid k \text{ est premier}\}$  l'ensemble des nombres premiers inférieurs à  $n$ , et la fonction  $\pi(n) := \#P_n$ , le nombre de premiers inférieurs à  $n$ .

On a vu que la fonction  $Li(x) = \int_2^x \frac{dt}{\log t}$  donne une bonne approximation de  $\pi(x)$ .

Cette fonction peut être approximée par la somme de Riemann de pas constant égal à 1.

$$S\left(\frac{1}{\log x}\right) = \sum_{k=0}^{n-1} \frac{1}{\log(2+k)} = \frac{1}{\log 2} + \sum_{k=3}^{n-1} \frac{1}{\log k} \quad (1)$$

La fonction  $x \mapsto \frac{1}{\log x}$  est une fonction continue, décroissante et positive sur l'intervalle  $[2, \infty[$ . L'erreur entre  $Li(x)$  et la fonction en escalier ci-dessus est donc bornée.

$$\left| S\left(\frac{1}{\log x}\right) - Li(x) \right| \leq \left| \sum_{k=2}^{n-1} \frac{1}{\log k} - \frac{1}{\log(k+1)} \right| = \frac{1}{\log(2)} - \frac{1}{\log(n)} < \frac{1}{\log 2} < 2$$



### 2.2.1 Ensembles aléatoires

Nous allons utiliser (1) pour générer des ensembles aléatoires  $R_k \subset \mathbb{N}, k \in \{1, \dots, 100\}$ , de sorte que chaque entier  $n$  ait une probabilité de  $1/\log(n)$  d'appartenir à l'ensemble.

$$\forall i \in \mathbb{N}, P(i \in R_k) = \begin{cases} 0 & \text{si } i = 1 \\ 1 & \text{si } i = 2 \\ \frac{1}{\log i} & \text{si } i \geq 3 \end{cases}$$

Ces ensembles seront construits jusqu'à  $i = 10^7$  : on a donc  $R_k \subset \{1, \dots, 10^7\}$ .

Il est à noter deux cas particuliers :

- Le nombre 1 est exclu. En effet,  $\frac{1}{\log 1}$  n'est pas défini. Par définition, 1 n'est pas un nombre premier.
- Le nombre 2 est inclus par défaut. En effet,  $P(2 \in R_k) = 1$  car  $\frac{1}{\log 2} > 1$ . De plus, le nombre 2 est par définition, un nombre premier.

La fonction  $\sigma_{R_k}(n) := \#\{i \in R_k | i < n\}$  mesure donc la taille des ensembles jusqu'à un certain  $n$ . Cette fonction est donc une variable aléatoire strictement inférieure à  $n$  dont l'espérance, que nous noterons  $\sigma_R(n)$ , est donnée par :

$$\sigma_R(n) = \sum_{i=1}^{n-1} 1 \cdot P(i \in R_k) = 1 + \sum_{i=3}^{n-1} \frac{1}{\log i}$$

L'erreur entre  $\sigma_R(n)$  et la somme de Riemann (1) est constante, égale à  $\frac{1}{\log 2} - 1 < 1$ .

Les ensembles aléatoires générés de cette manière suivront donc une distribution similaire à  $\text{Li}(x)$ , et donc à  $\pi(x)$  (voir figures 4 et 5).

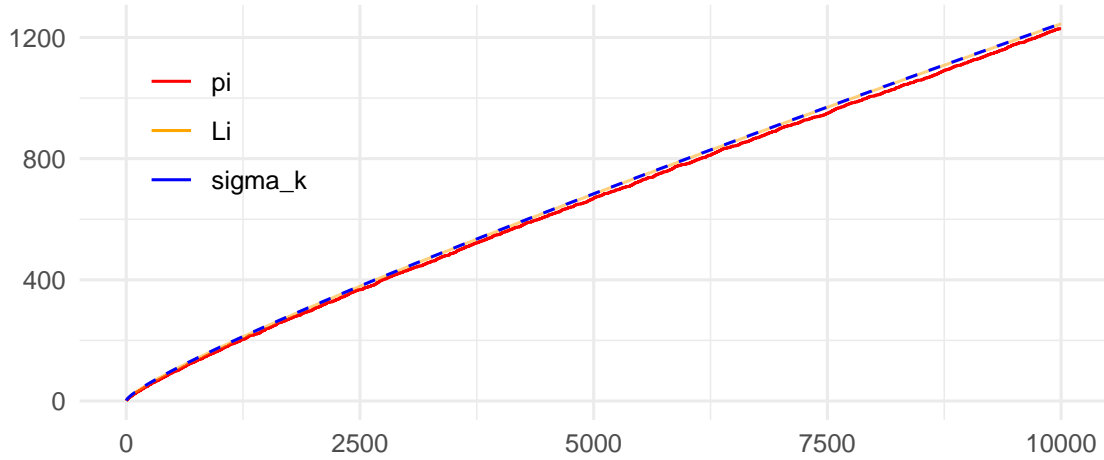


FIGURE 4 – Graphes des fonctions  $\pi$ ,  $\text{Li}$  and  $\sigma_R$ .  $\text{Li}$  et  $\sigma_R$  sont superposées.

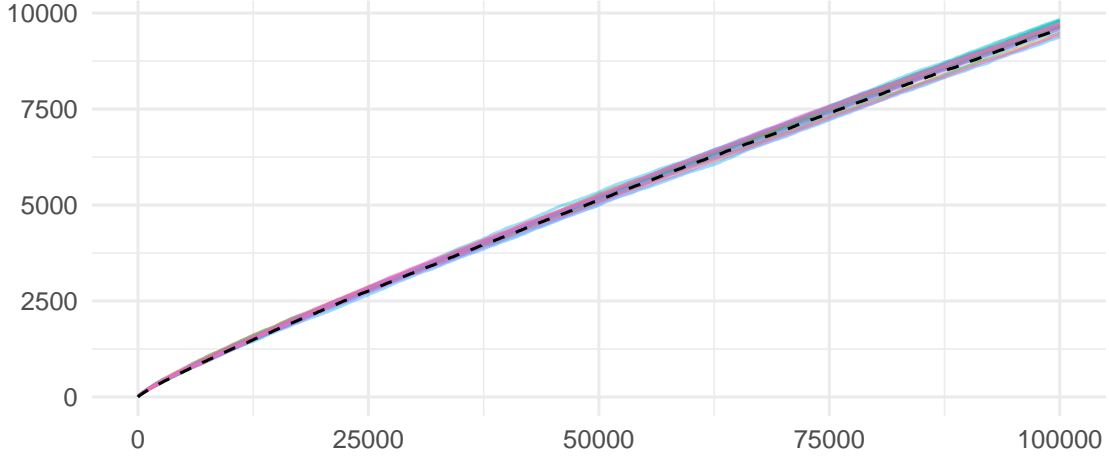


FIGURE 5 – graphes des fonctions  $\sigma_{R_k}$  pour  $k \leq 25$  (i.e. les 25 premiers ensembles) et  $\pi$  (en pointillé)

### 2.2.2 Ensembles probabilistes impairs

Bien que ces ensembles aléatoires suivent la distribution de  $\pi(x)$ , il manque une propriété importante des nombres premiers : à l'exception de 2, tous les nombres premiers sont impairs. Nous allons alors modifier l'algorithme mentionné précédemment afin de générer des ensembles ayant cette propriété. Soient alors les ensembles  $R'_k \subset \{(2\mathbb{N} + 1) \cup \{2\}\}$  tels que  $\forall i \in \{(2\mathbb{N} + 1) \cup \{2\}\}$ , la probabilité que  $i$  soit dans l'ensemble  $R'_k$  soit :

$$P(i \in R'_k) = \begin{cases} 0 & \text{si } i = 1 \\ 1 & \text{si } 2 \leq i < 9 \\ \frac{2}{\log i} & \text{si } i \geq 9 \end{cases}$$

Le fait que  $P(i \in R'_k) = 1$  pour les entiers impairs inférieurs à 9 découle du fait que  $2/\log(n) > 1$  pour ces entiers.

Ces ensembles débutent tous avec les mêmes éléments :  $(2, 3, 5, 7, \dots)$ , mais contiennent ensuite des éléments choisis aléatoirement à partir du 5<sup>e</sup> terme. Pour tout  $n \in 2\mathbb{N} + 1, n \geq 5$ , les fonctions  $\sigma_{R'_k}$  sont donc des variables aléatoires. Pour simplifier l'écriture dans les sommations ci-dessous, nous posons, pour tout  $n \in 2\mathbb{N} + 1, m := \lfloor n/2 \rfloor$ , de sorte que  $2m - 1$  soit bien le plus grand entier impair inférieur à  $n$ . Notons  $\sigma_{R'}(n)$  l'esperance de  $\sigma_{R'_k}(n)$ .

$$\sigma_{R'}(n) = 4 + \frac{2}{\log 9} + \frac{2}{\log 11} + \dots = 4 + \sum_{i=5}^m \frac{2}{\log(2i - 1)}$$

L'erreur entre l'esperance de  $\sigma_{R'}(n)$  et  $Li(n)$  peut aussi être bornée :

$$\begin{aligned}
|\sigma_{R'}(n) - Li(n)| &= \left| 4 + \left( \sum_{k=5}^m \frac{2}{\log 2k-1} \right) - \left( \int_2^n \frac{dt}{\log t} \right) \right| \\
&= \left| 4 + \left( \sum_{k=5}^m \frac{2}{\log 2k-1} \right) - (Li(9) + \int_9^n \frac{dt}{\log t}) \right| \\
&\leq \left| \left( \sum_{k=5}^m \frac{2}{\log 2k-1} \right) - \left( \int_9^n \frac{dt}{\log t} \right) \right| + |4 - Li(9)| \\
&= \left| \sum_{k=5}^m \frac{2}{\log 2k-1} - \int_{2k-1}^{2k+1} \frac{dt}{\log t} \right| + Li(9) - 4 \\
&< \left( \sum_{k=5}^m \frac{2}{\log 2k-1} - \frac{2}{\log(2(k+1)-1)} \right) + Li(9) - 4 \\
&= \frac{2}{\log 9} - \frac{2}{\log(2m+1)} + Li(9) - 4 \\
&< \frac{2}{\log 9} + Li(9) - 4 < 2
\end{aligned}$$

La première inégalité résulte de l'inégalité triangulaire et de  $Li(9) > 4$ . Le second inégalité vient du fait que  $x \mapsto 1/\log(x)$  est positive et décroissante sur l'intervalle  $[5, \infty[$ , et donc que  $\frac{2}{\log n} > \int_n^{n+2} \frac{dt}{\log t} > \frac{2}{\log(n+2)}$

Nous obtenons alors des ensembles aléatoires, constitués de nombres impairs (sauf 2), suivant la distribution de  $Li(x)$ . Les figures suivantes montrent la courbe de la fonction  $\sigma_{R'}$  (figure 4), puis la distribution des 25 ensembles aléatoires impairs générés par cet algorithme (7).

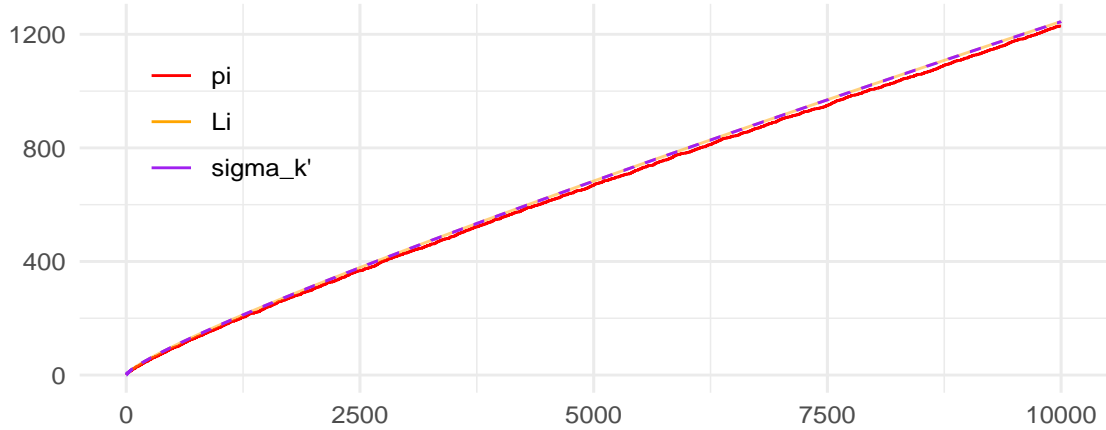


FIGURE 6 – Graphes des fonctions  $\pi$  et  $\sigma'_{R'}$ .

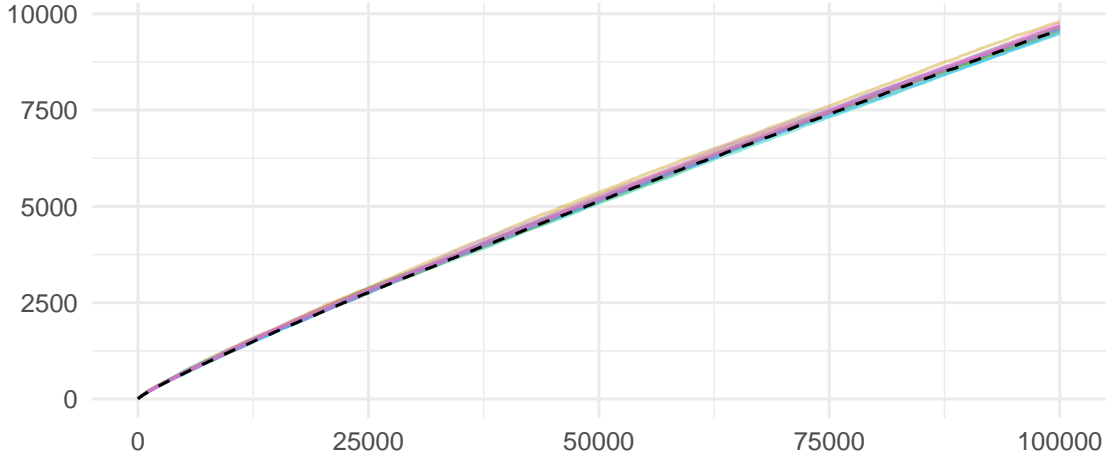


FIGURE 7 – graphes des fonctions  $\sigma_{R'_k}$  pour  $k \leq 25$  (25 premiers ensembles impairs) et  $\pi$

### 2.3 Ressemblance et différences entre ensembles aléatoires et nombres premiers

Nous souhaitons maintenant comparer les groupes d'ensembles aléatoires entre eux. Pour cela, nous allons mesurer, pour chaque ensemble, le rapport  $\sigma_k(n)/\pi(n)$  en fonction de  $n \in \mathbb{N}$ , où  $\sigma_k(n)$  mesure le nombre d'éléments inférieurs à  $n$ .

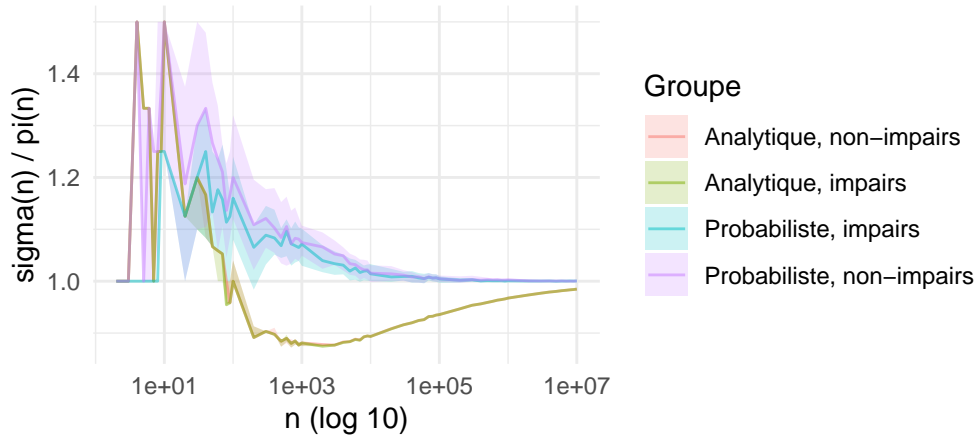


FIGURE 8 – Rapport entre  $\sigma(x)$  et  $\pi(x)$ . La courbe représente la médiane de chaque groupe d'ensemble. La surface autour de la courbe montre l'écart entre les premiers et troisièmes quartiles.

La figure 8 montre bien que tous ces ensembles suivent la distribution de  $\pi(x)$ , et convergent à partir  $10^3$ . Il est à noter que les ensembles  $Q$  (analytiques) impairs et non-impairs sont superposés. Nous voyons que le nombre d'éléments dans ces ensembles est inférieur à  $\pi(x)$ , tandis que les ensembles  $R$  (probabilistes) possèdent globalement plus d'éléments. On observe aussi que le cardinal des ensembles  $R$  tend rapidement vers  $\pi(x)$ , et montre une plus grande dispersion.

Nous allons maintenant mesurer à quel point ces ensembles sont différents des nombres premiers. Calculons alors, pour chaque ensemble, la pourcentage de nombres premiers dans chaque ensemble, en fonction de  $n$ .

La figure 9 montre cette proportion pour chaque collection d'ensembles.

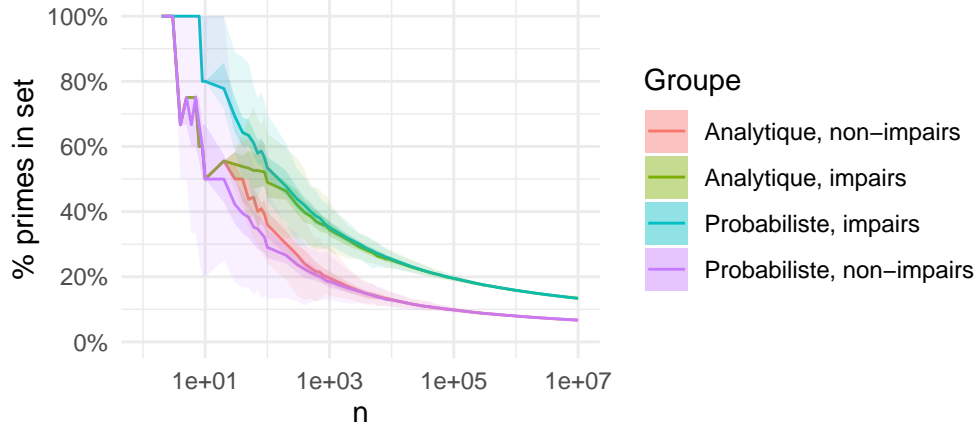


FIGURE 9 – Proportion de nombres premiers dans les ensembles aléatoires en fonction de  $n$ , en pourcentage. La courbe représente la médiane, la surface autour de la courbe, les premiers et troisièmes quartiles.

À partir de  $10^3$ , moins de la moitié des éléments des ensembles sont des nombres premiers. Dès  $10^5$ , les nombres premiers ne représentent plus que 10% des ensembles impairs, et 20% des ensembles non-impairs. De plus, il n’y a pas de différence significative entre deux ensembles d’un même groupe dès lors que  $n$  est assez grand.

### 3 Ensembles aléatoires et conjectures

Dans cette section, nous allons vérifier si des conjectures sur les nombres premiers peuvent s’appliquer aux ensembles aléatoires créés à la section 2. Ainsi, on sera en mesure d’estimer si la véracité d’une conjecture est susceptible de tenir grâce à la répartition des nombres premiers plutôt qu’à leur propriété d’être premier.

#### 3.1 Les nombres premiers jumeaux

La première conjecture que nous allons analyser, et sans doute la plus célèbre, est la conjecture des nombres premiers jumeaux.

**Définition 1.** Soient  $a, b \in \mathbb{N}$ ,  $a < b$ , on dit que  $a$  et  $b$  sont jumeaux si  $a + 2 = b$ .

**Conjecture 1.** *Il y a une infinité de nombres premiers jumeaux.*

Ces dernières années, il y’a eu de grosses avancées dans la démonstration de la conjecture. Ainsi, pour tout  $m \geq 1$ , soit  $H_m := \liminf_{n \rightarrow \infty} (p_{n+m} - p_n)$ , où  $p_n$  dénote le  $n$ -ième nombre premier. La conjecture des nombres premiers jumeaux est donc équivalente à  $H_1 = 2$ . En 2013, le mathématicien chinois Zhang Yitang est le premier à trouver une borne supérieure finie pour  $H_1$ , il a démontré que  $H_1 \leq 70\,000\,000$ . Suite à la publication de Zhang Yitang, de nombreux mathématiciens se sont mis en quête de réduire la borne supérieure de  $H_1$ . En optimisant les résultats de Zhang Yitang et grâce à d’autres méthodes ils ont pu montrer que  $H_1 \leq 246$ .

Afin de tester la conjecture sur les ensembles aléatoires, nous avons tracé un graphe (figure 10), où pour chaque ensembles aléatoires et l’ensemble des nombres premiers (jusqu’à  $10^7$ ) on trace une fonction qui compte le nombre de jumeaux. Nous pouvons faire les observations suivantes :

- Pour chaque type d’ensemble (selon l’approche utilisée), on constate, logiquement, que le nombre de jumeaux est plus ou moins le double pour les ensembles impairs.
- Il y’a plus de jumeaux dans les ensembles  $R$  que dans les ensembles  $Q$  ce qui peut s’expliquer par le fait qu’ils ont plus d’éléments que les ensembles  $Q$ .

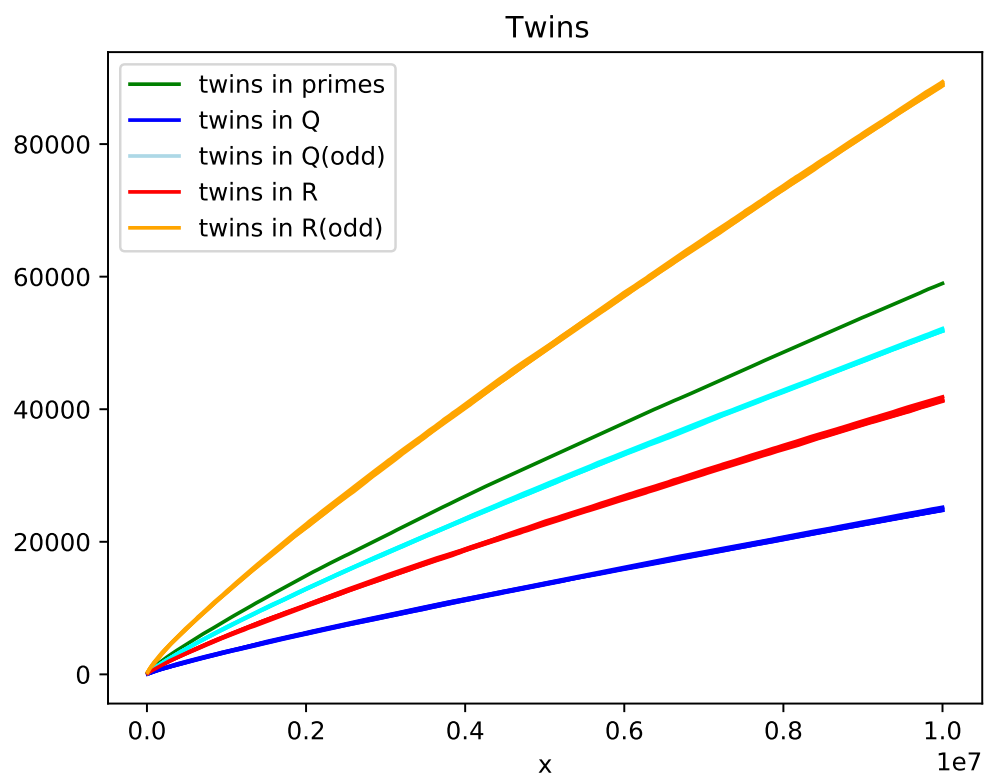


FIGURE 10 – Nombre de jumeaux dans chaque ensemble

- De manière générale, toutes les courbes sont croissantes, ce qui indiquerait, autant pour les nombres premiers que pour les ensembles aléatoires, que le nombre de jumeaux tend vers l'infini.

Par ailleurs, ce graphe éveille une idée intéressante. Si on désigne par  $f$  la fonction qui compte le nombre de jumeaux dans l'ensemble des nombres premiers et par  $g$  celle qui compte les jumeaux dans un ensemble aléatoire et qu'on parvient à montrer que  $f$  et  $g$  sont semblables (sous-entendu qu'elles le sont), alors  $\lim_{x \rightarrow \infty} g(x) = \infty \Rightarrow \lim_{x \rightarrow \infty} f(x) = \infty$ . Ce qui démontrerait la conjecture des nombres premiers jumeaux. Pour ce faire une idée d'une éventuelle équivalence entre  $f$  et  $g$ , voici le graphe (figure 11) de  $\frac{f(x)}{g(x)}$ , où  $g$  est appliquée à un ensemble impair  $Q$  choisi arbitrairement.

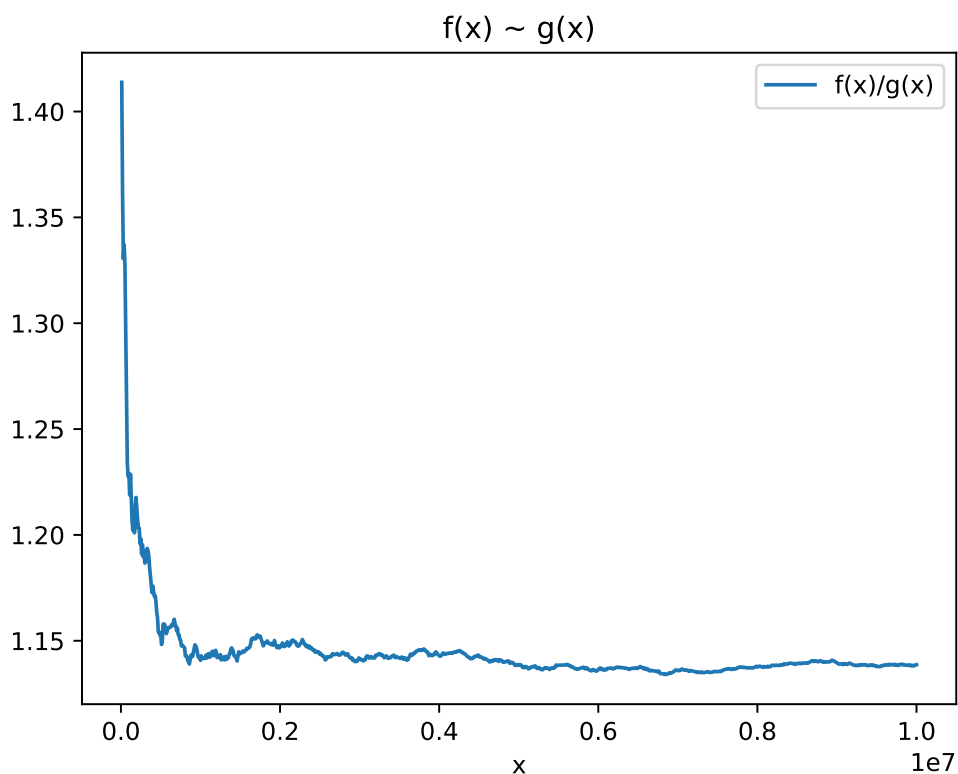


FIGURE 11 –  $\frac{f(x)}{g(x)}$

Pour les ensembles aléatoires  $R$  (créés par l'approche probabiliste), on peut avoir une bonne approximation de la fonction  $g$ , qui compte le nombre de jumeaux. Soit  $R$  un tel ensemble, on définit  $g : [0, \infty[ \rightarrow \mathbb{R}, x \mapsto \sum_{i=2}^{\lfloor x \rfloor} \frac{1}{\log(i) \log(i+2)}$ . "Elle donne une bonne approximation car c'est l'espérance mathématique." À titre comparatif, voici le graphe de  $g(x)$  (figure 12), d'une fonction qui compte les jumeaux dans l'ensemble des nombres premiers et une qui les compte dans  $R$ .

On peut montrer que  $\lim_{x \rightarrow \infty} g(x) = +\infty$ . Donc  $f \geq g \Rightarrow \lim_{x \rightarrow \infty} f(x) = +\infty$ .

Pour conclure, la répartition des nombres premiers nous a permis de créer des ensembles aléatoires qui ont une infinité de nombres jumeaux.

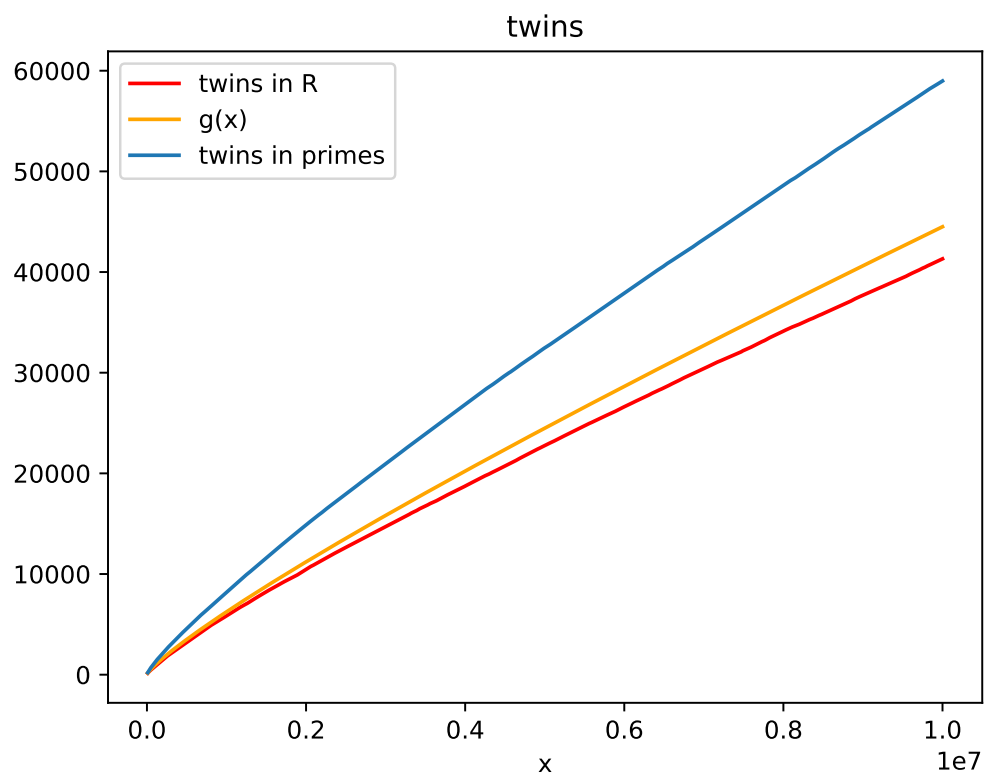


FIGURE 12 – Comparaison de  $g(x)$



## 3.2 Seconde conjecture

### 3.2.1 Introduction

La conjecture que nous allons tester ici est la suivante :

**Conjecture 2.** *Pour tout  $n = 6, 7, \dots$ , il existe un nombre premier  $p$  tel que  $6n - p$  et  $6n + p$  sont tous les deux premiers.*<sup>1</sup>

Nous allons d'abord vérifier que cette conjecture tient pour les nombres premiers, puis vérifier si celle-ci tient aussi pour les ensembles aléatoires suivant leur distribution. La procédure pour analyser cette conjecture est donc la suivante : pour chaque  $n \in \mathbb{N}, 6 \leq n \leq 10000$ , nous vérifierons pour chaque  $p \in R_k$  si  $(6n - p, 6n + p) \in R_k^2$ . Nous collecterons alors tout  $n$  tel que  $\neg P(n)$  dans des tableaux de données<sup>2</sup> afin d'analyser, pour chaque ensemble ou chaque groupe d'ensemble, le nombre et la distribution des erreurs.

### 3.2.2 Analyse

Soit  $P(n)$  l'assertion "pour un certain  $n \in \mathbb{N}, n \geq 6$ , Il existe  $p \in R_k, p \leq n$  tel que  $6n - p \in R_k$  et  $6n + p \in R_k$ ".

En utilisant un algorithme<sup>3</sup>, nous avons pu observer que, pour les nombres premiers, l'assertion est vraie pour tout  $6 < n \leq 10^6$ . Le même algorithme confirme que la conjecture n'est pas vérifiée, du moins pour tout  $n \geq 6$ , en ce qui concerne les ensembles aléatoires. Cependant, il semble que certains de ces ensembles possèdent des propriétés similaires si l'on choisit un  $n$  plus grand.

Nous nous intéressons au nombre d'erreurs (c'est à dire le nombre d'entiers  $n$  pour laquelle l'assertion n'est pas vérifiée), ainsi que le plus grand entier pour lequel l'assertion est fausse. Cette dernière information est intéressante car si ce plus grand entier est petit, alors la conjecture est vérifiée pour tout  $n$  plus grand.

Le graphique 13 ci-dessous représente en abscisses le nombre total d'erreur (c'est à dire  $\#\{n | \neg P(n)\}$ ) pour  $n \in \{6, \dots, 10^5\}$ , et en ordonnées le plus grand entier pour lequel l'assertion est fausse (ou bien  $\max\{n | \neg P(n)\}$ ). Chaque point représente un ensemble. Les ensembles ayant les meilleurs "performances" sont alors situés en bas à gauche : ceux-ci ont alors un faible nombre d'erreurs, et vérifie l'assertion pour tout  $n$  plus grand.

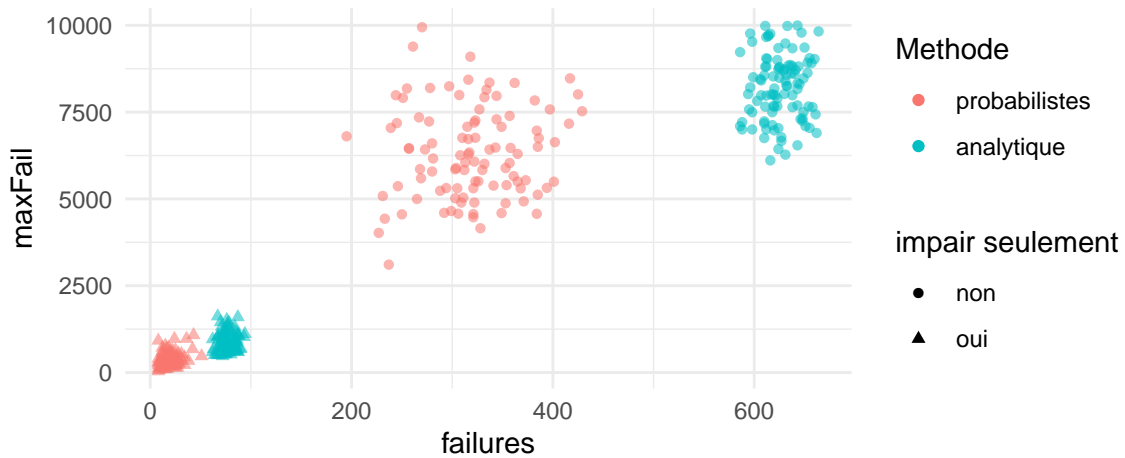


FIGURE 13 –

1. Conjecture 2.3 de [maths.nju.edu.cn/~zwsun/](https://maths.nju.edu.cn/~zwsun/)

2. le code pour ces tableaux de données est donné dans l'appendice A.2. Ceux-ci sont sauvegardés dans le dossier `/data/conjecture_2_3/` du github

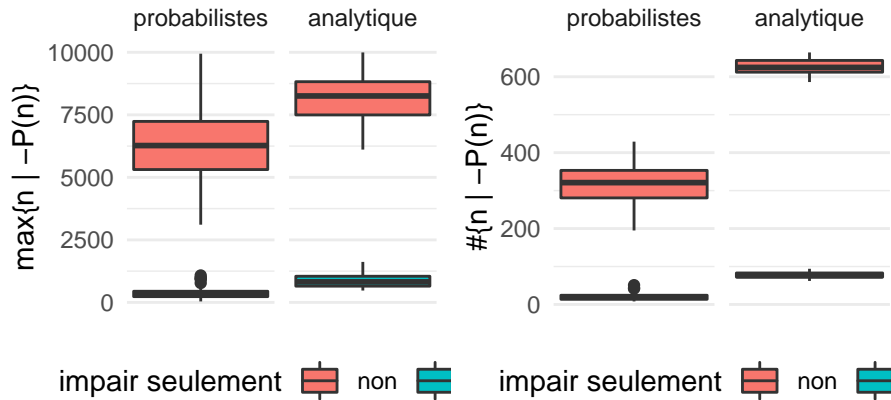
3. voir appendice : A.3.1 - "py\_code/test\_conj\_2\_3.py"

Nous debuterons par remarquer que l'existence d'un entier  $p$  répondant aux critères de la conjecture n'est pas rare. En effet, pour  $n \in \{6, \dots, 10000\}$ , l'assertion est vérifiée par près de 99% des entiers  $n$  testés pour chaque ensemble (moins de 650 erreurs).

Les ensembles non restreints aux nombres impairs génèrent tout de même un nombre assez élevé d'erreurs : plus de 200 erreurs pour la quasi-totalité de ces ensembles. De plus, ces erreurs persistent assez tardivement : pour la majorité de ces ensembles, il existe (au moins) un entier  $n > 5000$  pour lequel l'assertion est fausse.

Les ensembles composés de nombres impairs ont cependant de bien meilleures performances . Ceux-ci ont un faible nombre d'erreurs (moins de 50 pour les ensembles générés par l'algorithme probabiliste, moins de 100 pour les ensembles générés par les algorithmes analytiques, voir figure 15). De plus, le plus grand entier pour lequel l'assertion n'est pas vérifiée est relativement faible : cela signifie que pour tout entier  $n > 1500$ , l'assertion est vérifiée. Pour plus de trois quarts des ensembles aléatoires générés par l'algorithme probabiliste, on a même l'assertion vérifiée pour tout  $n > 500$ . Finalement, on remarque aussi que les ensembles créés par l'algorithme probabiliste ont des performances sensiblement meilleures. Cela est sûrement du au fait qu'ils possèdent sensiblement plus d'éléments que les autres ensembles (voir figure 8), ou alors parce que leur distribution est plus proche de celles des nombres premiers.

Les diagrammes en boîtes 14 et 15 ci-dessus offrent un aperçu de la distribution des erreurs. La seconde figure se concentre sur les ensembles impairs afin de faciliter la lecture du graphique.



(a) Plus grand  $n$  ne vérifiant pas l'assertion

(b) Nombre d'erreurs

FIGURE 14 – Diagrammes en boîte

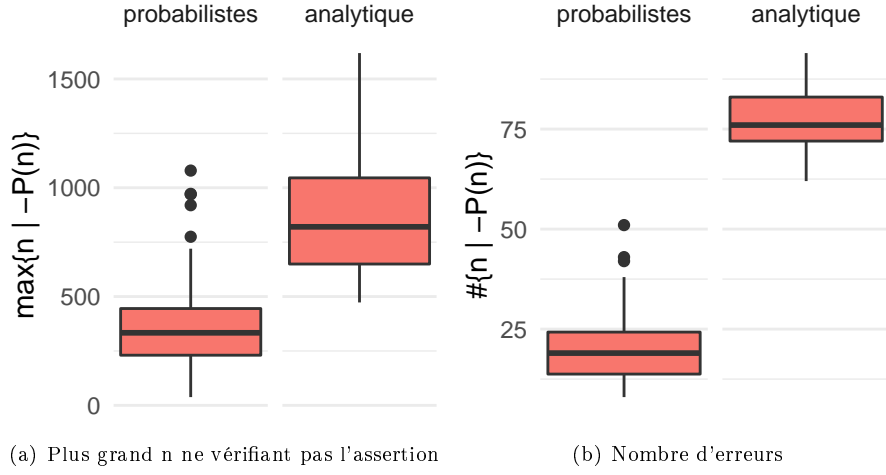


FIGURE 15 – Diagrammes en boîte - ensembles impairs.

Il existe trois ensembles vérifiant l'assertion pour tout  $n$ ,  $100 \leq n \leq 10000$  :

- l'ensemble probabiliste impair 032 vérifiant l'assertion pour tout  $n > 38$
- l'ensemble probabiliste impair 091 vérifiant l'assertion pour tout  $n > 74$
- l'ensemble probabiliste impair 033 vérifiant l'assertion pour tout  $n > 97$

Pour ces ensembles, l'assertion tient, comme pour les nombres premiers, au moins jusqu'à  $n = 10^6$ . De plus, au delà de 100, il existe 50 ensembles ayant moins de 5 erreurs. Il apparaît alors que la probabilité d'une erreur diminue, et tend vers 0, lorsque  $n$  devient grand, comme le fait remarquer le graphique ci-dessous, représentant le nombre moyen d'erreurs sur un interval  $[6 + 10n, 6 + 11n]$ .

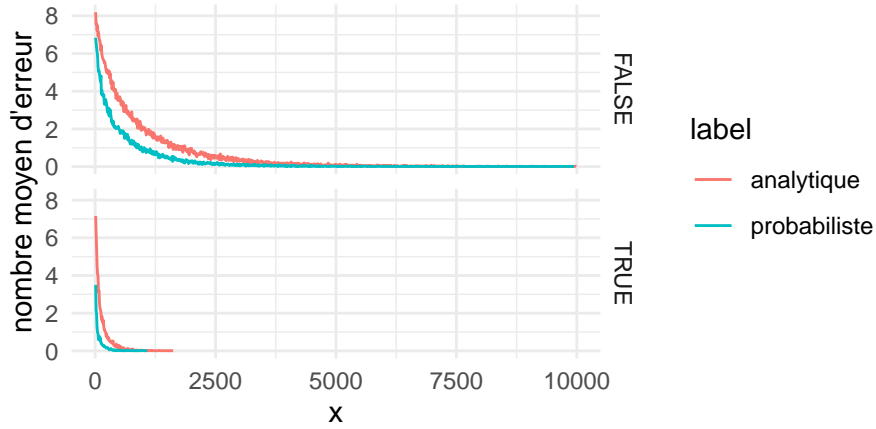


FIGURE 16 – Diagramme représentant le nombre moyen d'erreur sur l'intervalle  $]6 + 10n, 6 + 11n]$

On observe ici que pour les ensembles aléatoires impairs, le nombre moyen d'erreurs tend très rapidement vers 0 jusqu'à qu'aucune erreur n'apparaisse aux alentours de 1200. Cette même statistique diminue moins vite pour les autres ensembles, mais tend elle aussi vers 0. Il semble alors que pour un  $n$  suffisamment grand, la probabilité que l'assertion soit fausse diminue, voire devient négligeable pour les ensembles impairs.

### 3.2.3 Conclusion de l'analyse

Certains ensembles aléatoires, notamment les ensembles impairs, semblent vérifier la conjecture pour  $n$  plus grand que 6. Pour certains d'entre eux, il semble qu'il existe un entier  $m \in \mathbb{N}$  à partir duquel, pour tout  $n > m$ , il existe toujours un élément  $p < n$  dans l'ensemble tel que  $p, 6n - p$  et  $6n + p$  sont tous dans l'ensemble. Nous nous demandons alors si cette conjecture tient pour les nombres premiers, comme pour certains ensembles aléatoires démontrant de bonnes "performances", qu'à cause d'une distribution "heureuse" au départ, lorsque  $n$  est suffisamment petit. Lorsque  $n$  est suffisamment grand, la probabilité d'une erreur est alors si faible qu'elle devient négligeable.

## 4 Conclusion

## A Appendice

### A.1 Code : Création d'ensembles aléatoires

Les codes des appendices A.1.1 et A.1.1 sont ceux utilisés pour générer les ensembles aléatoires par algorithme probabiliste. Ceux-ci possèdent des complications inutiles, dus au fait que nous enregistrons au départ les ensembles sous forme de tableau de données. Nous montrons ces algorithmes tels quels, car ce sont ceux qui ont été utilisés pour générer les ensembles. Nous proposons cependant en appendice A.1.2 une version simplifiée de code.

#### A.1.1 Création d'ensembles probabiliste

Le code suivant genere 100 ensembles aléatoires non-impairs.

---

```
../r_code/generate_samples2.R
1 # import base
2 source("r_code/pack_func.R")
3 library(stringr)
4
5 # bound of the samples
6 bound <- 10^7
7 x <- seq(2,bound)
8 p <- "data/prob_sets/"
9 dir.create(p)
10
11 # Sel generates a random variable between 0 and 1, returns true if
    variable <= 1/log(x), false otherwise
12 Sel <- function(x) {runif(1) <= 1/log(x)}
13
14 df <- data.frame(x)
15 # This function generates a single random set. Takes all integers
    between 2 and x and apply Sel function logic
16 k <- 1
17
18 rsamp <- function() {
19   d <- data.frame(x,v= sapply(x, Sel))
20   df <- as.data.frame(cbind(df,R = d$v))
21   sample <- d$x[ which(d$v == TRUE)]
22   write.table(sample, paste(p,"prob_set_",str_pad(k,3,pad="0"),".txt",
    sep=""),row.names = F,col.names = F)
23   # only selected integers are returned in the list.
24   k <- k + 1
25   d
26 }
27
28 # generates a list of samples
29 rand_sets <- replicate(100,rsamp())
```

---

### A.1.2 Création d'ensembles probabiliste impairs

---

```
../r_code/odd_prob_sets.R
1 # This script will generate multiple randoms sets following the
  distribution of  $\pi(x)$ .
2 # With the exception of 2, all elements of the sets are odd integers,
  which is a property of prime numbers.
3
4 # import base functions and
5 source("r_code/pack_func.R")
6 library(stringr)
7
8 # bound of the samples
9 bound <- 10^7
10 x <- c(2, seq(3, bound, 2))
11 p <- "data/odd_prob_sets/" #Setup path to record sets
12 dir.create(p) #Create directory accordingly to path
13
14 # Sel generates a random variable between 0 and 1, returns true if
  variable <= 1/log(x), false otherwise
15 Sel <- function(x) {runif(1) <= 2/log(x)}
16
17 df <- data.frame(x)
18 # This function generates a single random set. Takes all integers
  between 2 and x and apply Sel function logic
19 k <- 1
20
21 rsamp <- function() {
22   d <- data.frame(x, v= sapply(x, Sel))
23   #df <- as.data.frame(cbind(df, R = d$v))
24   sample <- d$x[ which(d$v == TRUE)]
25   write.table(sample, paste(p, "odd_prob_set_", str_pad(k, 3, pad="0"), ".
    txt", sep=""), row.names = F, col.names = F)
26   # only selected integers are returned in the list.
27   k <- k + 1
28   sample
29 }
30
31 # generates a list of samples
32 rand_sets <- replicate(100, rsamp())
```

---

### A.1.3 Création d'ensembles probabiliste impairs

../r\_code/simplified\_generator.R

---

```
1 library(stringr)
2 library(magrittr)
3
4 gen_sample <- function(n) {
5   i <- i + 1 # i is used to name file within the folder
6   # (2:n) generates the integer interval [2,n]
7   # The part within the square bracket generates a boolean vector of
8     same length with probability = 1/m, m in [2,n]
9   (2:n)[apply(2:n,FUN=function(m) runif(1) <= 1/log(m))] %>%
10  write.table(
11    paste(p,"prob_set_",str_pad(i,3,pad="0"),".txt",sep=""),
12    col.names = F,
13    row.names = F)
14  }
15 gen_odd_sample <- function(n){
16   i <- i + 1
17   # same function as before but interval and probability are updated.
18   c(2,seq(3,n,2))[apply(c(2,seq(3,n,2)),FUN=function(m) runif(1) <= 2
19     /log(m))] %>%
20   write.table(
21     paste(p,"odd_prob_set_",str_pad(i,3,pad="0"),".txt",sep=""),
22     col.names = F,
23     row.names = F)
24 }
25 i =0
26 p<- "data/prob_sets/" # folder where sets will be created
27 replicate(100,gen_sample(10^7))
28
29 i =0
30 p<- "data/odd_prob_sets/"
31 replicate(100,gen_odd_sample(10^7))
```

---

## A.2 Seconde conjecture

### A.2.1 Rapport de conjecture

Le code suivant génère un tableau de donnée pour chaque ensemble, contenant le nombre d'entiers vérifiant la conjecture, le nombre d'échec et le plus grand entier ne vérifiant pas la conjecture.

../py\_code/conjecture\_2\_3.py

---

```
1 import pandas as pd
2
3 primes = set()
4 for i in range(2, 100000):
5     if all(i % p > 0 for p in primes):
6         primes.add(i)
7
8
9 def conjecture(n, set):
10     """
11     Verify conjecture for a given integer n. \n
12     Loops through all elements in set se, returns a tuple (n,p) where
13     : \n
14     \t - n is the inputed number
15     \t - p is the minimum element of the set for which both (6n-p) and
16         (6n+p) are elements of the set
17     """
18     #  $n*6-p > 0 \Rightarrow n*6 > p$ 
19     s = {p for p in set if p < n}
20     for p in s:
21         if (n*6 - p) in set and (n*6 + p) in set:
22             return (n, p)
23
24 def tryForSet(set, bound, setname):
25     """
26     Try conjecture for a given set up to a certain bound. \n
27     setname is a string which will be used for the output report. \n
28     Output is a dataframe with setname, number of successes and
29     failures, maximum element for which conjecture was not verified
30     and success rate for the set
31     """
32     success, failure, maxFail = 0,0,0
33     for x in range(6,bound):
34         if conjecture(x, set):
35             success += 1 # Increment success count if output is non
36                           null
37         else:
38             failure += 1 # Increment failure count if output is nul
39             maxFail = x # then sets maxFail to last unverified
40                           element.
41     print("Set: ", setname, \
42           ", success: ", success, \
43           ", failure:", failure, \
44           ", maxFail:", maxFail, \
45           ", success rate: ", round(success/(failure+success)*100,2), "%",
46           sep = " ")
```



```

41     return pd.DataFrame([[setname, success, failure, maxFail]], columns
42                           =["set", "succes", "failures", "maxFail"])
43
44 import os
45 bound = 10000
46 files = [f for f in os.listdir('./data/odd_prob_sets') if f.endswith('.txt')]
47
48 def record_output(folder, bound, output):
49     """
50     Takes a folder containing sets as an input, generates a csv report
51     up to a given bound
52     """
53     df = pd.DataFrame(columns=["set", "succes", "failures", "maxFail"])
54     df = df.append(tryForSet(primes, bound, "primes"))
55     files = [f for f in os.listdir(folder) if f.endswith('.txt')]
56     for f in files:
57         s = {int(line.strip()) for line in open(folder + f)}
58         df = df.append(tryForSet(s, bound, f))
59         df.to_csv('./data/' + output + ".csv")
60
61 record_output('./data/odd_prob_sets/', 10000, 'conjecture_2_3_odd_10k')
62 record_output('./data/prob_sets/', 10000, 'conjecture_2_3_10k_V3')
63 record_output('./data/odd_analytic_sets/', 10000, 'conjecture_2_3_odd_analytique_10k')
64 record_output('./data/analytic_sets/', 10000, 'conjecture_2_3_analytique_10k')

```

---

### A.3 Test conjecture

Le code suivant vérifie l'assertion pour tout entier, après avoir donné fourni les limites supérieures et inférieures. Aucun rapport n'est généré, mais le programme échoue dès qu'un entier ne satisfait pas les conditions de l'assertion

../py\_code/test\_conj\_2\_3.py

---

```
1 import pandas as pd
2 def conjecture(n, set):
3     """
4     Verify conjecture for a given integer n. \n
5     Loops through all elements in set se, returns the first element of
        the set p for which both  $6*n-p$  and  $6*n+p$  are elements of the
        set.
6     """
7     #  $n*6-p > 0 \Rightarrow n*6 > p$ 
8     s = {p for p in set if p < n}
9     for p in s:
10         if (n*6 - p) in set and (n*6 + p) in set:
11             return (p)
12
13 minbound = int(input("Please enter lower bound: ")) # initial value
        for which assertion will by test
14 maxbound = int(input("Please enter higher bound: ")) # last value for
        which assertion will by tested
15
16 # Select a set here
17 s = {int(line.strip()) for line in open('./data/odd_prob_sets/odd_
        prob_set_032.txt')}
18
19 # Loop through all integers between minbound and maxbound
20 # fails as soon as the assertion is not verified.
21 for n in range(minbound, maxbound):
22     p = conjecture(n, s)
23     if p:
24         print(n, p)
25     else:
26         print("Failed at rank", n)
27         break
28 print("Finished")
```

---

### A.3.1 Error mapping

../py\_code/conjecture2\_3\_error\_mapping.py

---

```
1 import pandas as pd
2
3 def conjecture(n, set):
4     """
5     Verify conjecture for a given integer n. \n
6     Loops through all elements in set se, returns a tuple (n,p) where
7     : \n
8     \t - n is the inputed number
9     \t - p is the minimum element of the set for which both (6n-p) and
10         (6n+p) are elements of the set
11     """
12     # n*6-p > 0 => n*6 > p
13     s = {p for p in set if p < n and n*6 > p in set}
14     for p in s:
15         if (n*6 - p) in set and (n*6 + p) in set:
16             return p
17
18 def tryForSet(set, bound, setname):
19     """
20     Try conjecture for a given set up to a certain bound. \n
21     setname is a string which will be used for the output report. \n
22     Output is a dataframe with setname, number of successes and
23         failures, maximum element for which conjecture was not verified
24         and success rate for the set
25     """
26     global df
27     print("Trying set: ", setname)
28     for x in range(6, bound):
29         p = conjecture(x, set)
30         if p:
31             if x*6-p not in set or x*6+p not in set:
32                 print("Error for", x, p)
33                 exit
34             else:
35                 # print("Set", setname, "failed for n=", x)
36                 df = df.append(pd.DataFrame([[setname, x]], columns=["set",
37                     "failedInt"]))
38
39 folder = './data/'
40 import os
41 df = pd.DataFrame(columns=["set", "failedInt"])
42
43 # for subfolder in [subfolder for subfolder in os.listdir(folder) if "
44     set" in subfolder]:
45     # for f in [f for f in os.listdir(folder + "/" + subfolder) if f.
46         endswith('.txt')]:
47         # set = {int(line.strip()) for line in open(folder + "/" +
48             subfolder + "/" + f)}
49         # tryForSet(set, 1000, f)
```

```

43 #         df.to_csv(' ./data/failure_mapping.csv ')
44
45 for subfolder in [subfolder for subfolder in os.listdir(folder) if "
    set" in subfolder]:
46     for f in [f for f in os.listdir(folder + "/" + subfolder) if f.
        endswith('.txt')]:
47         set = {int(line.strip()) for line in open(folder + "/" +
            subfolder + "/" + f)}
48         tryForSet(set, 10000, f)
49         df.to_csv(' ./data/failure_mapping_10K_V2.csv ')

```

---