Prime vs Random Sets

Sebastien Plaasch
Maxime Rubio

31 mai 2020

Table des matières

1 Introduction

Le travail qui est présenté dans ce rapport porte sur les nombres premiers et plus particulièrement sur leur distribution. L'objectif est d'évaluer la place de la distribution des nombres premiers dans des conjectures sur ceux-ci. Nous désignerons par $\mathbb{P}$ l'ensemble des nombres premiers et par π la fonction définie comme suit : $\pi : [0, \infty[\to \mathbb{R}, x \mapsto \#\{p \in \mathbb{P} \mid p \leqslant x\}$.

Dans un premier temps, nous avons créer des ensembles aléatoires qui partagent la même distribution que l'ensemble des nombres premiers. Pour créer les ensembles et vérifier leur fiabilité à la distribution des nombres premiers, nous avons utilisé le théorème des nombres premiers :

Théorème 1 (Théorème des Nombres Premiers)**.** *Quand x tend vers l'infini :*

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

Nous allons aussi utiliser la fonction Li : $[2, \infty[\to \mathbb{R}, x \mapsto \int_2^x \frac{dt}{\log(t)}$ et la relation $\pi(x) \sim \mathrm{Li}(x)$ quand x tend vers l'infini. Par deux approches différentes, nous avons créé 400 ensembles aléatoires sur lesquels nous avons testé deux conjectures.

2 Création des ensembles aléatoires

2.1 Approche analytique

Notre première 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 2) issu de l'article [2] .

Théorème 2. *L'hypothèse de Riemann est équivalente à l'assertion*

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

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 positifs. Pour $n > 11$, voici la méthode de sélection de l'élément $q_n \in Q$:

- on pose $a_n = \max\left\{q_{n-1}, \lceil ali(n) - \frac{1}{\pi}\sqrt{n}\log^{5/2}(n)\rceil\right\}$ (où $\lceil x\rceil$ désigne la partie entière supérieure de x) ;
- on pose $b_n = \lfloor ali(n) + \frac{1}{\pi}\sqrt{n}\log^{5/2}(n)\rfloor$ (où $\lfloor x\rfloor$ désigne la partie entière inférieure de x) ;
- q_n est choisi aléatoirement dans $\{m \in \mathbb{N} \,|\, a_n \leqslant m \leqslant b_n\}$ avec une distribution uniforme ; chaque élément de l'ensemble a la même probabilité d'être sélectionné.

En procédant de la sorte, le théorème 2 sera toujours vrai pour tout ensemble aléatoire Q.

Par cette méthode, nous avons créé 200 ensembles de nombres inférieurs 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. Nous testerons nos conjectures sur ces ensembles.

2.1.2 Définition de la fonction σ_Q

On peut désormais définir $\sigma_Q : [0, \infty[\to \mathbb{R}$, $x \mapsto \#\{n \in Q : n < x\}$. Nous parlerons systématiquement de la fonction σ_Q 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, grâce à la construction des ensembles Q, nous pouvons supposer que toutes les fonctions σ_Q sont semblables, comme l'illustre le graphe 17 en annexe.

Afin de visualiser $\sigma_Q(x)$ en la comparant à $\pi(x)$ et $\frac{x}{\log(x)}$, voici leur graphe respectif (où σ_Q représente un de nos ensembles) :

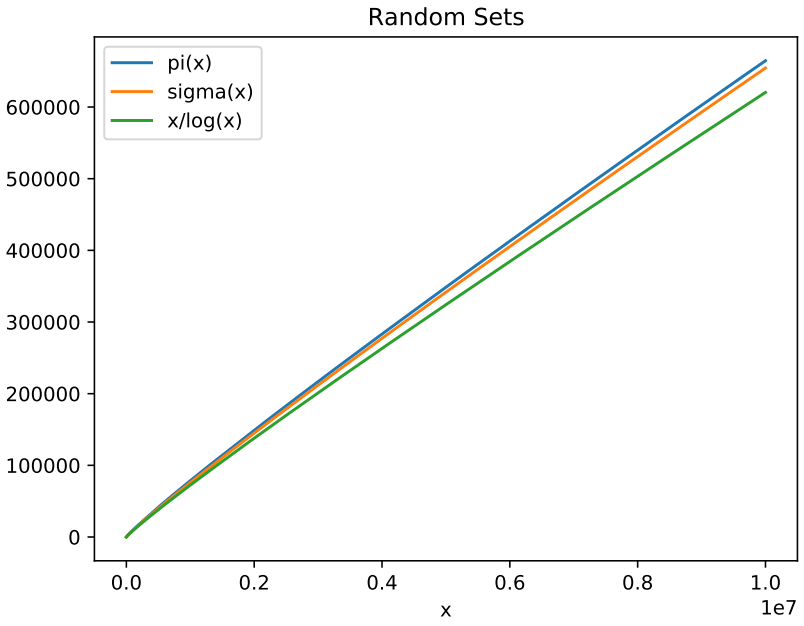


FIGURE 1 – Graphe de σ_Q, π et $\frac{x}{\log(x)}$

2.1.3 Écart entre $\pi(x)$ et $\sigma_Q(x)$

La fonction σ_Q semble suivre la même allure que π. Cependant, lors de nos expérimentations, nous avons dessiné des graphes (figures 18, 19 et 20) pour des valeurs de x inférieures à celles de la figure 1. Pour des petites valeurs de x, la courbe de $\sigma_Q(x)$ était presque confondue avec celle de $\frac{x}{\log(x)}$. Lorsque les valeurs de x sont de plus en plus grandes, $\sigma_Q(x)$ tend vers $\pi(x)$. Pour analyser l'écart entre $\pi(x)$ et $\sigma_Q(x)$, nous avons tracé le graphe de la fonction $x \mapsto \frac{\pi(x)}{\sigma_Q(x)}$:

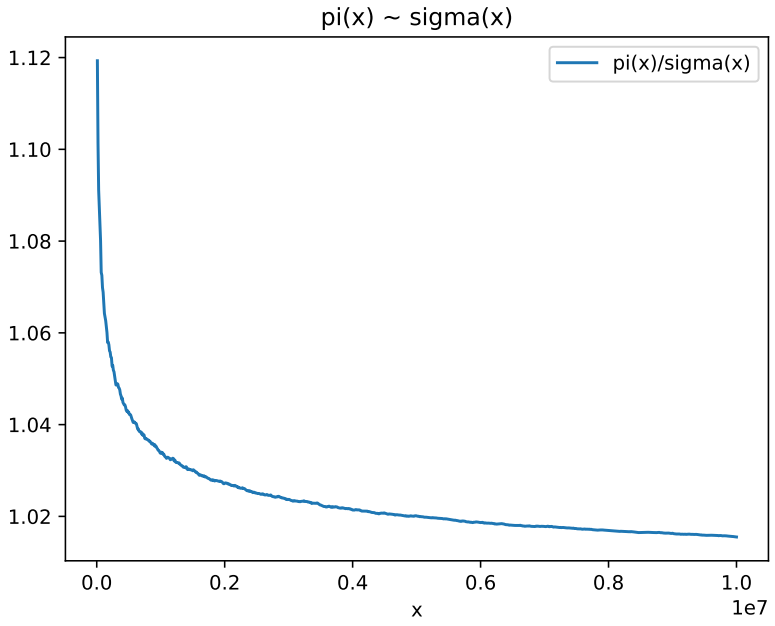


FIGURE 2 – Graphe de $\frac{\pi}{\sigma_Q}$

On peut supposer que $\pi(x) \sim \sigma_Q(x)$.

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 en vérifiant si le théorème des nombres premiers (voir 1) est vrai quand on remplace $\pi(x)$ par $\sigma_Q(x)$. Pour démontrer le Théorème des Nombres Premiers, il a été démontré que, quand x tend vers l'infini, $\mathrm{Li}(x) \sim \frac{x}{\log(x)}$. Nous pouvons montrer, graphiquement, que lorsque x tend vers l'infini, $\sigma_Q(x) \sim \mathrm{Li}(x)$ (voir figure 3), ce qui implique que $\sigma_Q(x) \sim \frac{x}{\log(x)}$.

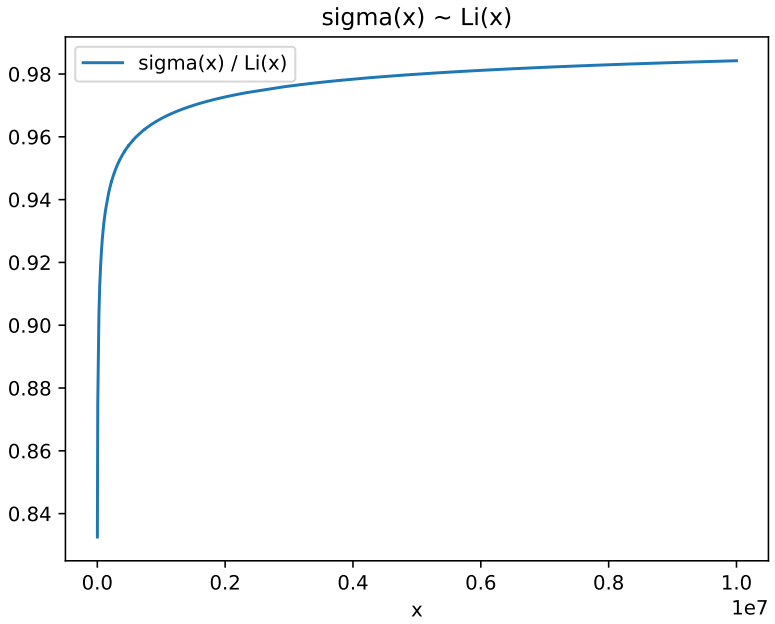


FIGURE 3 – Graphe de $\frac{\sigma_q}{Li}$

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 nombres de premiers inférieurs à n.

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

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

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

La fonction $x \mapsto \frac{1}{\log x}$ est une fonction continue, décroissante et positive sur l'intervalle $[2, \infty[$. L'erreur entre $\text{Li}(x)$ 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 $R_k \subset \mathbb{N}, k \in \{1, ..., 100\}$, 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 $R_k \subset \{1, ..., 10^7\}$.

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 $\mathrm{Li}(x)$, et donc à $\pi(x)$ (voir figures 4 et 5).

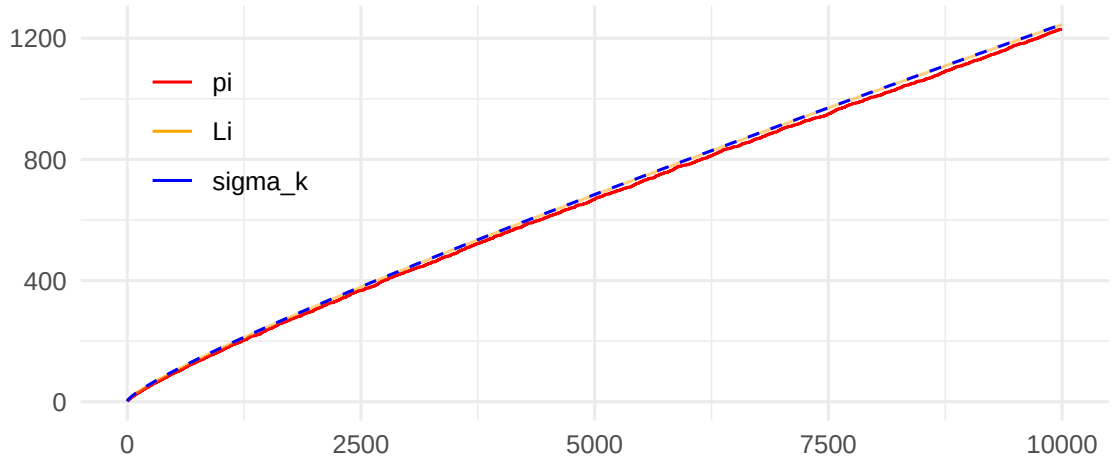


FIGURE 4 – Graphes des fonctions π, Li and σ_R. Li et σ_R sont superposées.

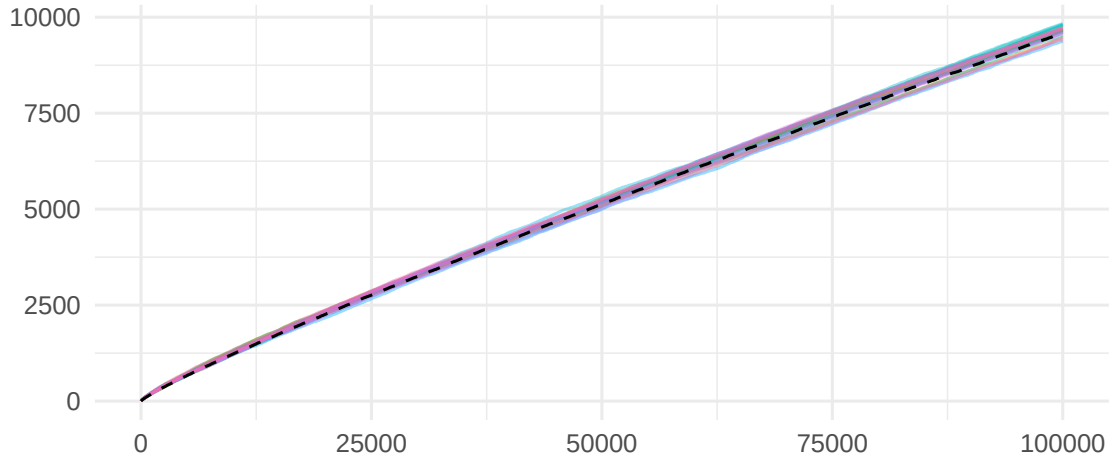


FIGURE 5 – graphes des fonctions σ_{R_k} pour $k \leq 25$ (i.e. les 25 premiers ensembles) et π (en pointillé)

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 : $(2, 3, 5, 7, ...)$, mais contiennent ensuite des éléments choisis aléatoirement à partir du 5$^{\text{e}}$ 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 $\sigma_{R'_k}(n)$.

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

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

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

La première inégalité résulte de l'inégalité triangulaire et de $Li(9) > 4$. La seconde 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 $\mathrm{Li}(x)$. 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).

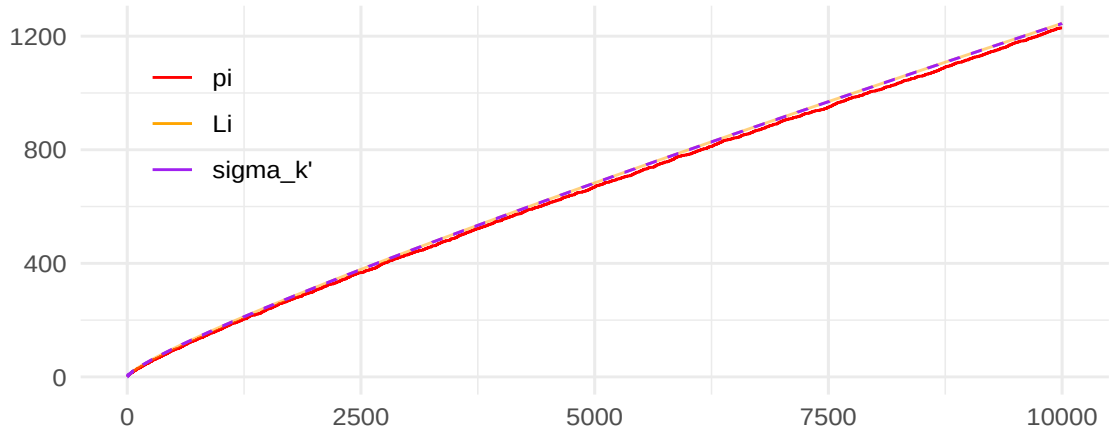


FIGURE 6 – Graphes des fonctions π et σ'_R.

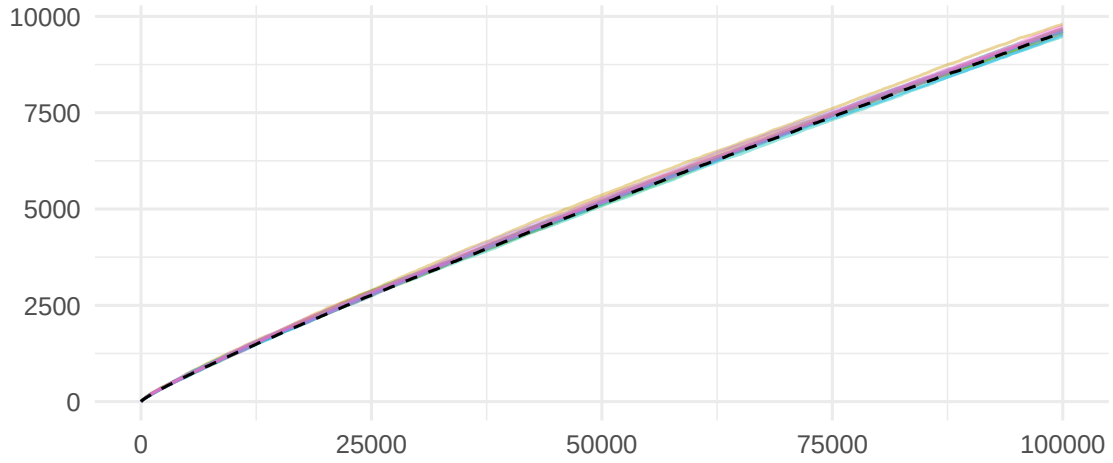


FIGURE 7 – graphes des fonctions $\sigma_{R'_k}$ pour $k \leq 25$ (25 premiers ensembles impairs) et π

2.3 Comparaison 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.

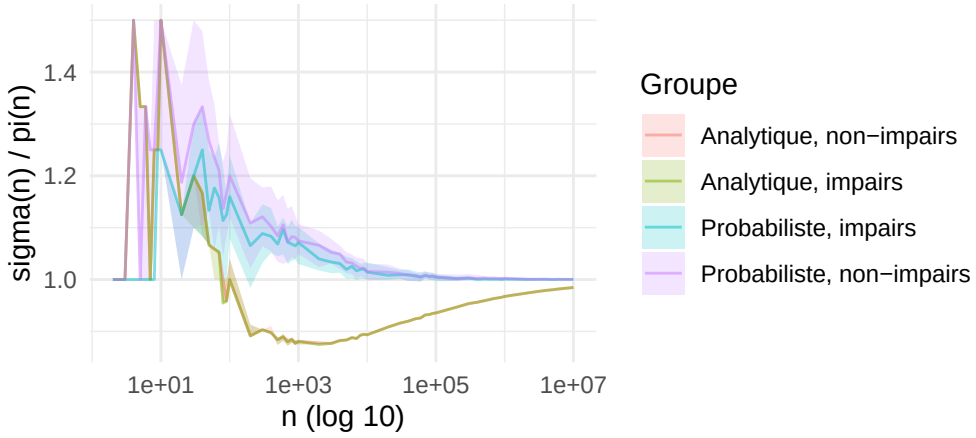


FIGURE 8 – Rapport entre $\sigma(x)$ et $\pi(x)$. La courbe représente la médiane de chaque groupe d'ensemble. La surface autour de la courbe montre l'écart entre les premiers et troisièmes quartiles.

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.

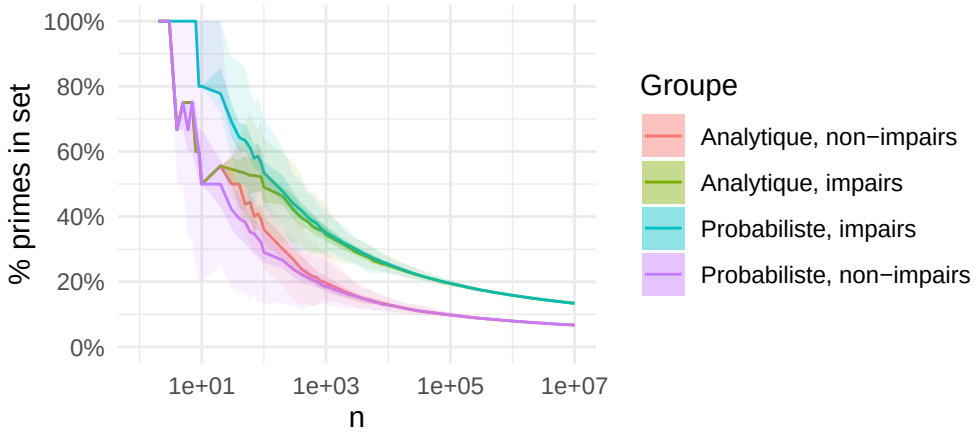


FIGURE 9 – Proportion de nombres premiers dans les ensembles aléatoires en fonction de n, en pourcentage. La courbe représente la médiane, la surface autour de la courbe, les premiers et troisièmes quartiles.

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

3.1.1 Introduction

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.*

Ainsi, pour tout $m \geqslant 1$, soit $H_m := \liminf_{n\to\infty}(p_{n+m} - p_n)$, 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 $H_1 \leqslant 70\,000\,000$. 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 $H_1 \leqslant 246$ (voir l'article [1]).

3.1.2 Analyse

Définissons pour un ensemble quelconque S la fonction $f_S : [2, \infty[\to \mathbb{N}, x \mapsto f_S(x) = \#\{n \in S \mid n+2 \in S, n+2 \leqslant x\}$. Afin de tester la conjecture sur les ensembles aléatoires, nous avons tracé le graphe (figure 10) de $f_{\mathbb{P}}$, f_Q (pour 20 ensembles dont 10 sont composés d'éléments impairs) et f_R (pour 20 ensembles, comme pour f_Q).

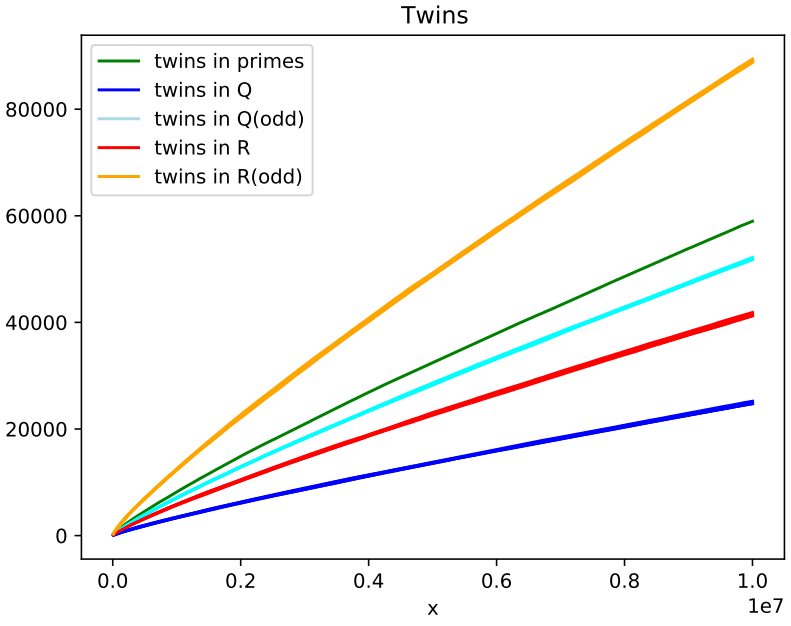


FIGURE 10 – Nombre de jumeaux dans chaque ensemble

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 est probablement dû à la construction des ensembles ;
— 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 deux idées intéressantes qui prouveraient la conjecture :
— Si $f_{\mathbb{P}}(x) \sim g(x)$ quand x tend vers l'infini ($g = f_Q$ ou $g = f_R$) et $\lim\limits_{x\to+\infty} g(x) = \infty$ alors $\lim\limits_{x\to+\infty} f_{\mathbb{P}}(x) = \infty$. Pour ce faire une idée d'une éventuelle similarité entre $f_{\mathbb{P}}$ et g quand x tend vers l'infini, nous avons tracé le graphe de $\frac{f_{\mathbb{P}}}{g}$, où $g = f_Q$ pour un ensemble Q quelconque (figure 11) ;
— Si $f_{\mathbb{P}} \geqslant g$ et $\lim\limits_{x\to+\infty} g(x) = \infty$ alors $\lim\limits_{x\to+\infty} f_{\mathbb{P}}(x) = \infty$.

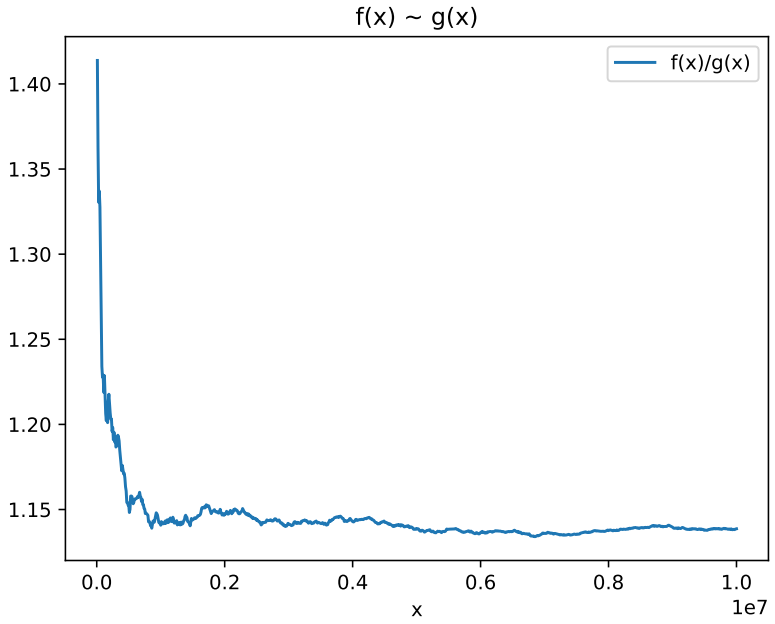


FIGURE 11 – $\frac{f_{\mathbb{P}}}{g}$

De plus, pour les ensembles aléatoires R créés selon l'approche probabiliste, on peut avoir une bonne approximation de la fonction f_R. Soit R un tel ensemble, on définit $h : [0, \infty[\to \mathbb{R}, x \mapsto \sum_{i=2}^{\lfloor x \rfloor} \frac{1}{\log(i)\log(i+2)}$. Par construction, pour $i \geqslant 3$, la probabilité que $i \in R$ vaut $P(i \in R) = \frac{1}{\log(i)}$. Donc, la probabilité que i et $i+2 \in R$ vaut $P(i \in R \text{ et } i+2 \in R) = \frac{1}{\log(i)\log(i+2)}$ parce que les événements sont indépendants. La formule de l'espérance mathématique nous est donnée par $E(x) = \sum_i x_i p_i$ où $p_i = P(x = x_i)$. Or $\forall\ i\ x_i = 1$, donc en additionnant les probabilités, on obtient l'espérance mathématique. À titre comparatif, voici le graphe de h, $f_{\mathbb{P}}$ et f_R (figure 12).

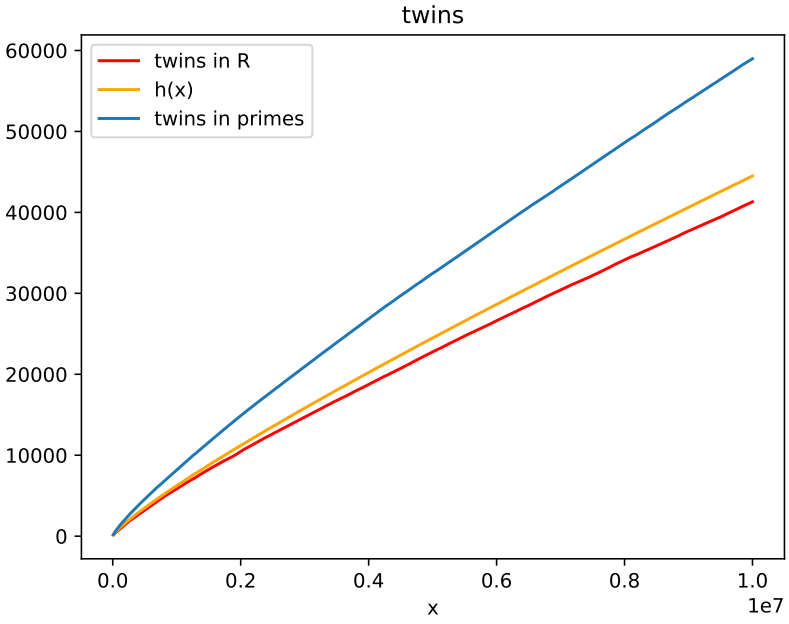


FIGURE 12 – Comparaison de h avec f_R et $f_{\mathbb{P}}$

On peut montrer que $\lim_{x\to\infty} h(x) = +\infty$ et ainsi prouver l'infinité des jumeaux dans un ensemble R, en supposant l'ensemble infini.

3.1.3 Conclusion de l'analyse

Pour conclure, nos méthodes de constructions d'ensembles aléatoires semblent, comme pour les nombres premiers, avoir une infinité de nombres jumeaux tout en ayant des résultats très différents entre les différents types d'ensembles. Ceci est probablement dû à la construction des ensembles. En effet, par l'approche analytique, pour le choix du n-ième élément on impose une distribution uniforme parmi un nombre fixe de possibilités. Tandis que pour l'approche probabiliste, le choix de l'élément n ne dépend que de la probabilité $\frac{1}{\log(n)}$, ce qui nous permet d'avoir une bonne approximation de la fonction g, la fonction h.

3.2 Seconde conjecture

3.2.1 Introduction

La conjecture que nous allons tester ici est la suivante :

Conjecture 2. *Pour tout $n = 6, 7, ...,$ il existe un nombre premier p tel que $6n - p$ et $6n + p$ sont tous les deux premiers.* [1]

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 [2] 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 [3], 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 lesquelle 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 $n \in \{6, ..., 10^5\}$, et en ordonnés 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.

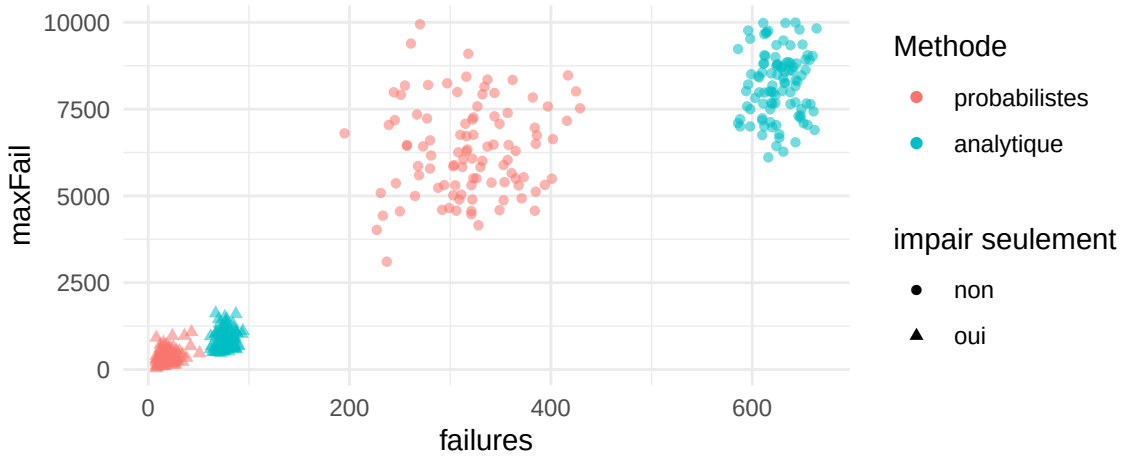


FIGURE 13 –

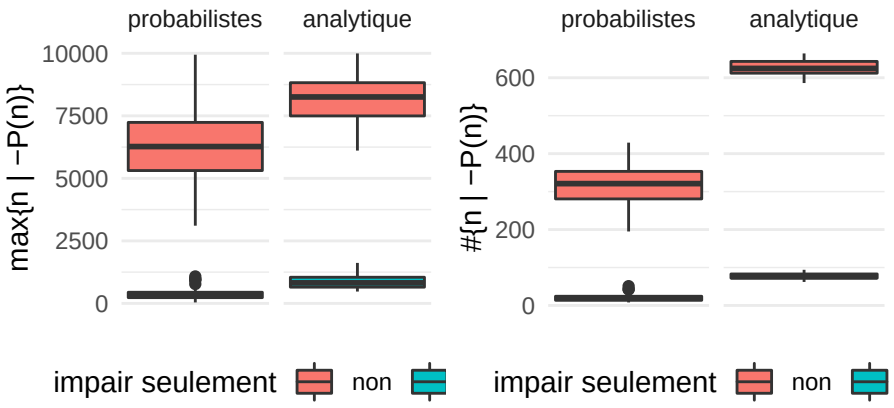
1. Conjecture 2.3 de maths.nju.edu.cn/z̃wsun/
2. le code pour ces tableaux de donnés est donné dans l'appendice A.2. Ceux-ci sont sauvegardés dans le dossier /data/conjecture_2_3/ du github
3. voir appendice : A.3.1 - "py_code/test_conj_2_3.py"

Nous débuterons par remarquer que l'existence d'un entier p répondant aux critères de la conjecture n'est pas rare. En effet, pour $n \in \{6, ..., 10000\}$, 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.

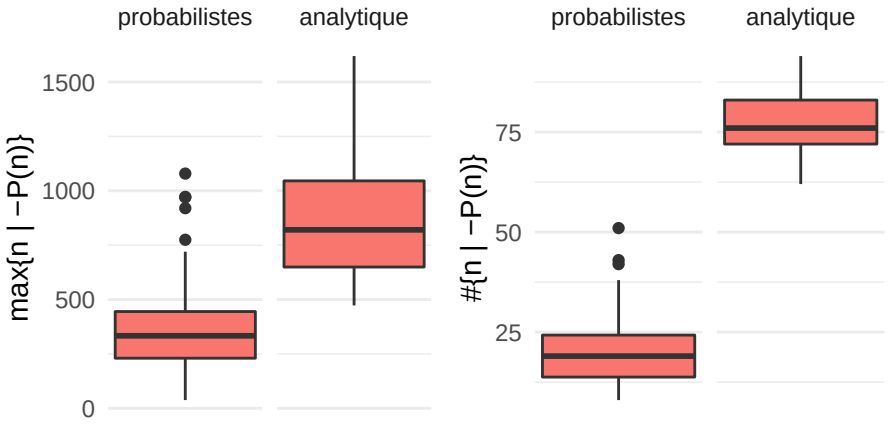
Les ensembles composés de nombres impairs ont cependant de bien meilleurs performances . Ceux-ci ont un faible nombre d'erreurs (moins de 50 pour les ensembles générés par l'algorithme probabiliste, moins de 100 pour les ensembles générés par les algorithmes analytiques, voir figure 15). De plus, le 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 crées par l'algorithme probabiliste ont des performances sensiblement meilleures. Cela est surement du au fait qu'ils possèdent sensiblement plus d'éléments que les autre ensembles (voir figure 8), ou alors parce que leur distribution est plus proche de celles des nombres premiers.

Les diagrammes en boites 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 boite



(a) Plus grand n ne vérifiant pas l'assertion (b) Nombre d'erreurs

FIGURE 15 – Diagrammes en boite - 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 apparait 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 intervalle $[6+10n, 6+11n[$.

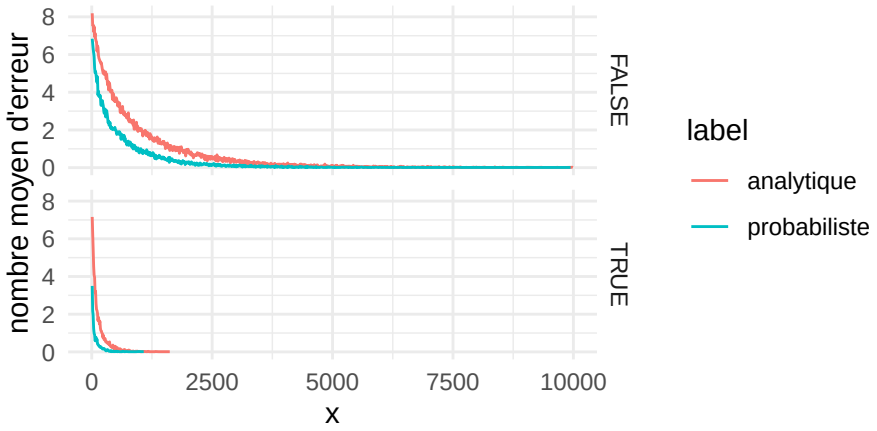


FIGURE 16 – Diagramme représentant le nombre moyen d'erreur sur l'intervale $]6 + 10n, 6 + 11n]$

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 suffisement 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, notemment 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 élement $p < n$ 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 suffisement petit. Lorsque n est suffisement grand, la probabilité d'une erreur est alors si faible qu'elle devient négligeable.

4 Conclusion

En conclusion de ce travail, on peut constater que l'approche probabiliste semble mieux fonctionner, ou du moins pour des ensembles plus petits. La distribution des ensembles R est directement très proche de celle des nombres premiers tandis que pour l'approche analytique, il faut prendre des ensembles plus grands avant d'avoir des résultats probants. Néanmoins, les deux approches illustrent bien la place que prend la distribution des nombres premiers et tout ce qu'on peut faire avec grâce à tout ce qu'on sait sur la fonction π et ses excellentes approximations. En se rapprochant davantage des nombres premiers en construisant des ensembles aléatoires composés de nombres impairs, on parvient à avoir d'excellents résultats pour les deux conjectures testées.

Références

[1] D.H.J Polymath. The "bounded gaps between primes" polymath project - a retrospective analysis. *EMS Newsletter*, December 2014.

[2] Juan ARIAS DE REYNA and Jérémy de TOULISSE. The n-th prime asymptotically. *Journal de Théorie des Nombres de Bordeaux*, 25, 2013.

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, dues au fait que nous enregistrions 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 génère 100 ensembles aléatoires non-impairs.

../r_code/generate_samples2.R

```r
# import base
source("r_code/pack_func.R")
library(stringr)

# bound of the samples
bound <- 10^7
x <- seq(2,bound)
p <- "data/prob_sets/"
dir.create(p)

# Sel generates a random variable between 0 and 1, returns true if
    variable <= 1/log(x), false otherwise
Sel <- function(x) {runif(1) <= 1/log(x)}

df <- data.frame(x)
# This function generates a single random set. Takes all integers
    between 2 and x and apply Sel function logic
k <- 1

rsamp <- function(){
  d <- data.frame(x,v= sapply(x, Sel))
  df <<- as.data.frame(cbind(df,R = d$v))
  sample <- d$x[which(d$v == TRUE)]
  write.table(sample, paste(p,"prob_set_",str_pad(k,3,pad="0"),".txt",
      sep=""),row.names = F,col.names = F)
  # only selected integers are returned in the list.
  k <<- k + 1
  d
}

# generates a list of samples
rand_sets <- replicate(100,rsamp())
```

A.1.2 Création d'ensembles probabiliste impairs

../r_code/odd_prob_sets.R

```r
# This script will generate multiple randoms sets following the distribution of pi(x).
# With the exception of 2, all elements of the sets are odd integers, which is a property of prime numbers.

# import base functions and
source("r_code/pack_func.R")
library(stringr)

# bound of the samples
bound <- 10^7
x <- c(2,seq(3,bound,2))
p <- "data/odd_prob_sets/" #Setup path to record sets
dir.create(p) #Create directory accordingly to path

# Sel generates a random variable between 0 and 1, returns true if variable <= 1/log(x), false otherwise
Sel <- function(x) {runif(1) <= 2/log(x)}

df <- data.frame(x)
# This function generates a single random set. Takes all integers between 2 and x and apply Sel function logic
k <- 1

rsamp <- function(){
  d <- data.frame(x,v= sapply(x, Sel))
  #df <<- as.data.frame(cbind(df,R = d$v))
  sample <- d$x[which(d$v == TRUE)]
  write.table(sample, paste(p,"odd_prob_set_",str_pad(k,3,pad="0"),".txt",sep=""),row.names = F,col.names = F)
  # only selected integers are returned in the list.
  k <<- k + 1
  sample
}

# generates a list of samples
rand_sets <- replicate(100,rsamp())
```

A.1.3 Création d'ensembles probabiliste impairs

../r_code/simplified_generator.R

```r
library(stringr)
library(magrittr)

gen_sample <- function(n) {
  i <<- i + 1 # i is used to name file within the folder
  # (2:n) generates the integer interval [2,n]
  # The part within the square bracket generates a boolean vector of
      same length with probability = 1/m, m in [2,n]
  (2:n)[sapply(2:n,FUN=function(m) runif(1) <= 1/log(m))] %>%
  write.table(
    paste(p,"prob_set_",str_pad(i,3,pad="0"),".txt",sep=""),
    col.names = F,
    row.names = F)
  }

gen_odd_sample <- function(n){
  i <<- i + 1
  # same function as before but interval and probability are updated.
  c(2,seq(3,n,2))[sapply(c(2,seq(3,n,2)),FUN=function(m) runif(1) <= 2
      /log(m))] %>%
    write.table(
      paste(p,"odd_prob_set_",str_pad(i,3,pad="0"),".txt",sep=""),
      col.names = F,
      row.names = F)
}

i =0
p<- "data/prob_sets/" # folder where sets will be created
replicate(100,gen_sample(10^7))

i =0
p<- "data/odd_prob_sets/"
replicate(100,gen_odd_sample(10^7))
```

A.1.4 Création d'ensembles par l'approche analytique

Le code qui suit comprend toutes les fonctions utilisées pour créer les ensembles aléatoires via l'approche analytique, les manipuler et tracer tous les graphes de la section 2.1.

../py_code/approche_analytique.py

```python
import matplotlib.pyplot as plt
from math import log, fabs, sqrt, pi, ceil, floor
from mpmath import li, findroot
from random import choice


def gen_primes(N):
    """Assume n is an integer, generate prime numbers up to N."""
    prime = set()
    for n in range(2, N + 1, 1):
```

```
11          if all(n % p > 0 for p in prime):
12              prime.add(n)
13              yield n
14
15
16  def g(n):
17      """Part of code often used in other functions."""
18      return (1 / pi) * sqrt(n) * (log(n))**(5/2)
19
20
21  def Q(N):
22      """Assume N is an integer, bigger than 26, generate Q(x) a random
            set up to N"""
23      q = [2, 3, 4, 5, 8, 10, 13, 16, 19, 23, 26]
24      for x in q:
25          yield x
26      qn0 = 26
27      for i in range(11, N, 1):
28          a = ceil(ali(i) - g(i))
29          if a < qn0:
30              a = qn0 + 1
31          b = floor(ali(i) + g(i))
32          qn1 = choice([k for k in range(a, b+1, 1)])
33          if qn1 <= N:
34              qn0 = qn1
35              yield qn1
36          else:
37              break
38
39
40  def get_set(N, file_name):
41      """Renvoit, sous forme de liste, l'ensemble aleatoire file_name,
            jusqu'a N."""
42      ensemble = []
43      of = open(file_name, 'r')
44      line = of.readline()
45      while line:
46          nombre = int(line[:-1])
47          if nombre <= N:
48              ensemble.append(nombre)
49          else:
50              break
51          line = of.readline()
52      return ensemble
53
54
55  def record_set(N, file_name):
56      """Enregistre un ensemble aleatoire, jusqu'a N, dans file_name."""
57      of = open(file_name, "w")
58      for number in Q(N):
59          of.write(str(number) + "\n")
60      of.close()
61
62
```

```
63  def sigma(x):
64      a, b = 0, len(q) - 1
65      if q[b] <= x:
66          return b+1
67      while b-a > 1:
68          m = (a+b) // 2
69          if q[m] == x:
70              return m+1
71          if q[a] <= x <= q[m]:
72              b = m
73          elif q[m] <= x <= q[b]:
74              a = m
75      else:
76          return a+1
77
78
79  def Pi(x):
80      """Alternative a Pi(x)."""
81      a, b = 0, len(p) - 1
82      if p[b] <= x:
83          return b+1
84      while b-a > 1:
85          m = (a+b) // 2
86          if p[m] == x:
87              return m+1
88          if p[a] <= x <= p[m]:
89              b = m
90          elif p[m] <= x <= p[b]:
91              a = m
92      else:
93          return a+1
94
95
96  def Li(x):
97      """Assume x is a real number, return li(x) - li(2)."""
98      return li(x) - li(2)
99
100
101 def ali(x):
102     """Return ali(x) such that li(ali(x)) = x."""
103     return findroot(lambda y: li(y) - x, - 7).real
104
105
106 def test1():
107     """Test whether sigma(x) ~ x/log(x) by plotting the graph of sigma
        (x) / (x/(log(x))."""
108     x = [i for i in range(100, 10**5+1, 100)]
109     y = [sigma(i)/(i/log(i)) for i in x]
110     g, = plt.plot(x, y)
111     plt.legend([g], ['sigma(x) / (x/(log(x))'])
112     plt.title("sigma(x) ~ x/log(x)")
113     plt.xlabel('x')
114     plt.savefig('images/test1.pdf')
115
```

```
116 def test2():
117     """Test whether sigma(x) ~ Li(x) by plotting the graph of sigma(x)
            / Li(x)."""
118     x = [i for i in range(100, 10**5+1, 100)]
119     y = [sigma(i)/Li(i) for i in x]
120     g, = plt.plot(x, y)
121     plt.legend([g], ['sigma(x) / Li(x)'])
122     plt.title("sigma(x) ~ Li(x)")
123     plt.xlabel('x')
124     plt.savefig('images/test2.pdf')
125
126 def graphe():
127     """Plot the graph of Li(x), Pi(x), sigma(x), x/log(x)."""
128     x = [i for i in range(10**4, 10**7 + 1, 10**4)]
129     y1 = [Li(i) for i in x]
130     y2 = [Pi(i) for i in x]
131     y3 = [sigma(i) for i in x]
132     y4 = [i/log(i) for i in x]
133     g1, = plt.plot(x, y1)
134     g2, = plt.plot(x, y2)
135     g3, = plt.plot(x, y3)
136     g4, = plt.plot(x, y4)
137     plt.legend([g2, g3, g4], ['pi(x)', 'sigma(x)', 'x/log(x)'])
138     plt.title("Random Sets")
139     plt.xlabel('x')
140     plt.savefig('images/analytic_approach_sets.pdf')
141
142 def ecarts():
143     """Plot the graph of Pi(x)/sigma(x)."""
144     x = [i for i in range(10**4, (10**7)+1, 10**4)]
145     y = [Pi(i)/sigma(i) for i in x]
146     g, = plt.plot(x, y)
147     plt.legend([g], "pi(x)/sigma(x)")
148     plt.title("pi(x) ~ sigma(x)")
149     plt.xlabel('x')
150     plt.savefig('images/ecarts.pdf')
```

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 tupple (n,p) where
           : \n
13     \t - n is the inputed number
14     \t - p is the minimum element of the set for which both (6n-p) and
           (6n+p) are elements of the set
15     """
16     # n*6-p > 0 => n*6 > p
17     s = {p for p in set if p < n}
18     for p in s:
19         if (n*6 - p) in set and (n*6 + p) in set:
20             return (n, p)
21
22 def tryForSet(set, bound, setname):
23     """
24     Try conjecture for a given set up to a certain bound. \n
25     setname is a string which will be used for the output report. \n
26     Output is a dataframe with setname, number of successes and
           failures, maximum element for which conjecture was not verified
           and success rate for the set
27     """
28     success, failure, maxFail = 0,0,0
29     for x in range(6,bound):
30         if conjecture(x, set):
31             success += 1    # Increment success count if output is non
                   null
32         else:
33             failure += 1    # Increment failure count if output is nul
34             maxFail = x     # then sets maxFail to last unverified
                   element.
35     print("Set: ", setname, \
36         ", success: ", success, \
37         ", failure:", failure, \
38         ", maxFail:", maxFail, \
39         ", success rate: ", round(success/(failure+success)*100,2),"%"
               , \
```

```
40          sep = "")
41      return pd.DataFrame([[setname, success, failure, maxFail]],columns
            =["set","succes","failures","maxFail"])
42
43  import os
44  bound = 10000
45
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
            up to a given bound
51      """
52      df = pd.DataFrame(columns=["set","succes","failures","maxFail"])
53      df = df.append(tryForSet(primes,bound,"primes"))
54      files = [f for f in os.listdir(folder) if f.endswith('.txt')]
55      for f in files:
56          s = {int(line.strip()) for line in open(folder + f)}
57          df = df.append(tryForSet(s,bound,f))
58          df.to_csv('./data/' + output + ".csv")
59
60  record_output('./data/odd_prob_sets/',10000,'conjecture_2_3_odd_10k')
61  record_output('./data/prob_sets/',10000,'conjecture_2_3_10k_V3')
62  record_output('./data/odd_analytic_sets/',10000,'conjecture_2_3_odd_
        analytique_10k')
63  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 => 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 tupple (n,p) where
           : \n
7      \t - n is the inputed number
8      \t - p is the minimum element of the set for which both (6n-p) and
           (6n+p) are elements of the set
9      """
10     # n*6-p > 0 => n*6 > p
11     s = {p for p in set if p < n and n*6 > p in set}
12     for p in s:
13         if (n*6 - p) in set and (n*6 + p) in set:
14             return p
15
16 def tryForSet(set, bound, setname):
17     """
18     Try conjecture for a given set up to a certain bound. \n
19     setname is a string which will be used for the output report. \n
20     Output is a dataframe with setname, number of successes and
           failures, maximum element for which conjecture was not verified
           and success rate for the set
21     """
22     global df
23     print("Trying set: ", setname)
24     for x in range(6,bound):
25         p = conjecture(x, set)
26         if p:
27             if x*6-p not in set or x*6+p not in set:
28                 print("Error for", x, p)
29                 exit
30         else:
31             # print("Set", setname, "failed for n=",x)
32             df = df.append(pd.DataFrame([[setname,x]], columns=["set",
                   "failedInt"]))
33
34
35 folder = './data/'
36 import os
37 df = pd.DataFrame(columns=["set","failedInt"])
38
39 # for subfolder in [subfolder for subfolder in os.listdir(folder) if "
       set" in subfolder]:
40 #     for f in [f for f in os.listdir(folder + "/" + subfolder) if f.
       endswith('.txt')]:
41 #         set = {int(line.strip()) for line in open(folder + "/" +
       subfolder + "/" + f)}
42 #         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')
```

A.4 Code pour le test de la conjecture 1

../py_code/conjecture.py

```
1  import matplotlib.pyplot as plt
2  import approche_analytique as analytic
3
4  def count_twins(N, ens):
5      """Fonction qui compte le nombre de nombres jumeaux inferieurs a x
           dans set."""
6      k = 0
7      for i in range(len(ens)-1):
8          if ens[i+1] < N:
9              if ens[i]+2 == ens[i+1]:
10                 k += 1
11         else:
12             break
13     return k
14
15 def twin_primes(N):
16     """Fonction qui renvoit un graphe de fonctions qui comptent le
           nombre
17     de nombres jumeaux inferieurs a x."""
18     x = [i for i in range(10, N+1, 10)]
19     path_q_odd = "~/analytique_odd_set"
20     path_q = "~/analytique_set_"
21     path_v_odd = "~/odd_prob_set_"
22     path_v = "~/prob_set_"
23     p = analytic.get_set(N, "~/set_primes.txt")
24     y1 = [count_twins(i, p) for i in x]
25     g1, = plt.plot(x, y1)
26     for j in range(1, 6, 1):
27         q = analytic.get_set(N, path_q + str(j) + ".txt")
28         q_odd = analytic.get_set(N, path_q_odd + str(j) + ".txt")
29         v = analytic.get_set(N, path_v + str(0) + str(0) + str(j) + ".
               txt")
30         v_odd = analytic.get_set(N, path_v_odd + str(0) + str(0) + str
               (j) + ".txt")
31         y2 = [count_twins(i, q) for i in x]
32         y2_odd = [count_twins(i, q_odd) for i in x]
33         y3 = [count_twins(i, v) for i in x]
34         y3_odd = [count_twins(i, v_odd) for i in x]
```

```
35          if j == 1:
36              g2, = plt.plot(x, y2, "blue")
37              g2_odd, = plt.plot(x, y2_odd, "lightblue")
38              g3, = plt.plot(x, y3, "red")
39              g3_odd, = plt.plot(x, y3_odd, "orange")
40          else:
41              plt.plot(x, y2, "blue")
42              plt.plot(x, y2_odd, "cyan")
43              plt.plot(x, y3, "red")
44              plt.plot(x, y3_odd, "orange")
45      plt.legend([g1, g2, g2_odd, g3, g3_odd], ['twins in primes', '
            twins in Q', 'twins in Q(odd)', 'twins in R', 'twins in R(odd)'
            ])
46      plt.title("Twins")
47      plt.xlabel('x')
48      plt.savefig('images/twins.pdf')
```

A.5 Graphes supplémentaires

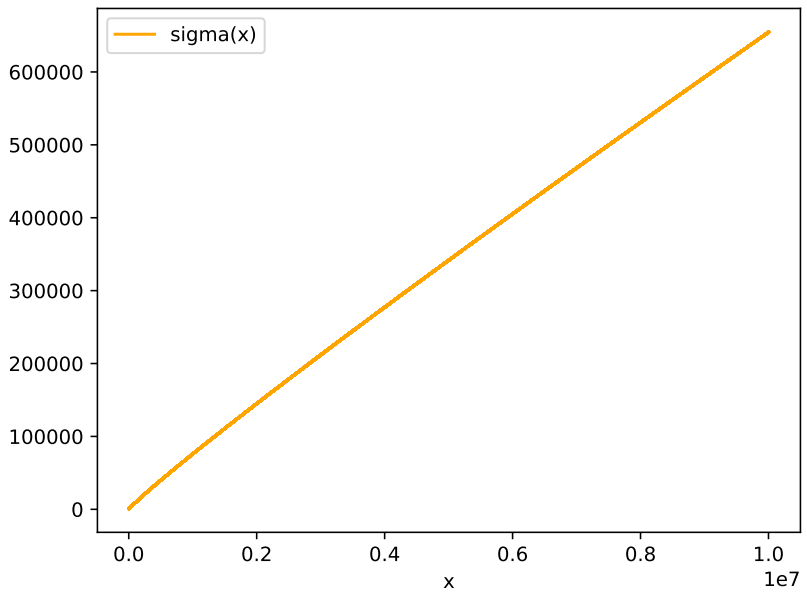


FIGURE 17 – Graphe de σ_Q pour tous les ensembles Q

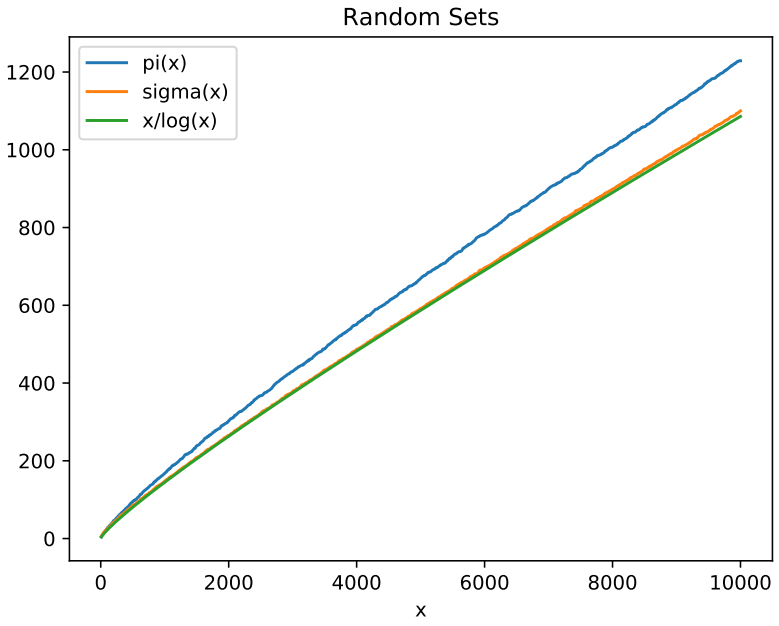


FIGURE 18 – Figure 1 pour x jusqu'à 10^4

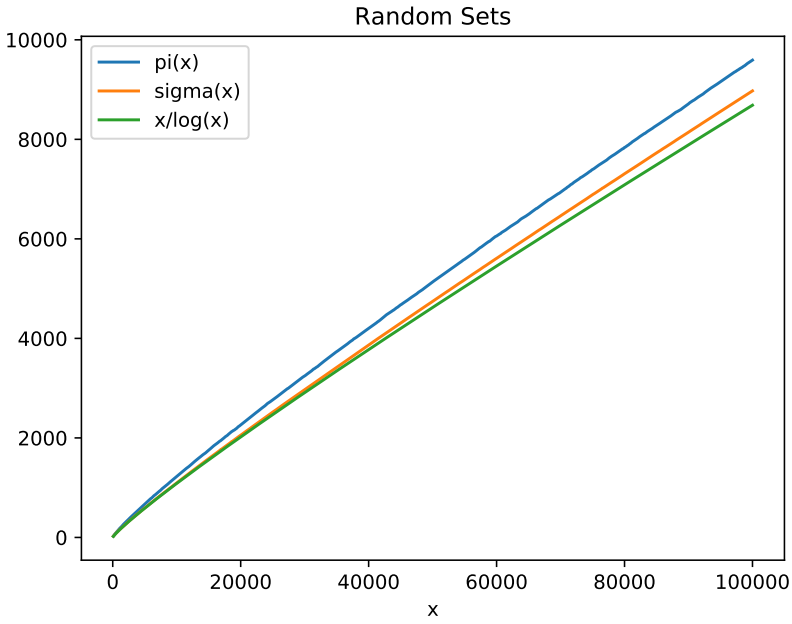


FIGURE 19 – Figure 1 pour x jusqu'à 10^5

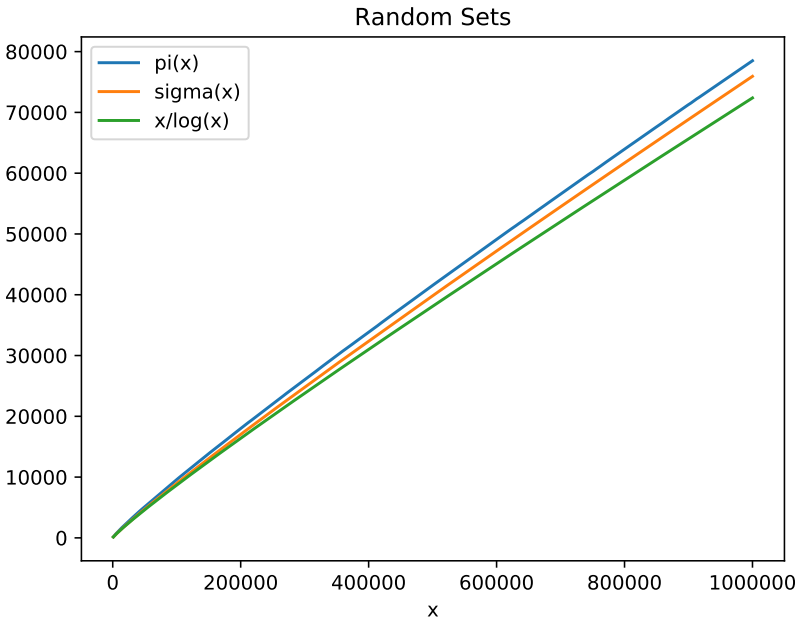


FIGURE 20 – Figure 1 pour x jusqu'à 10^6