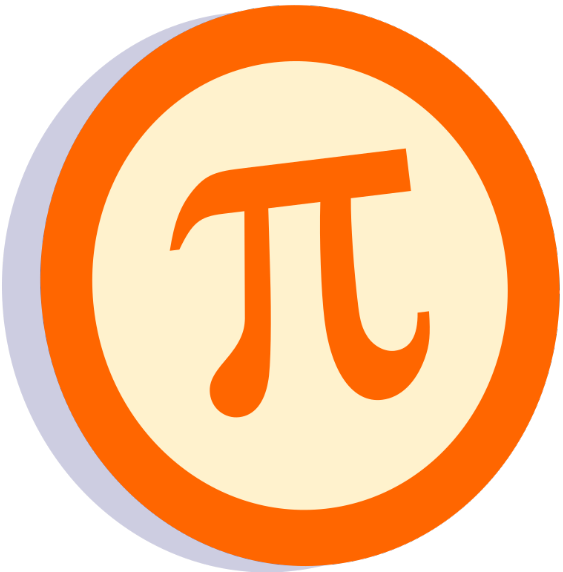


Pi randomness

Rapport de projet de simulation

Année académique 2016-2017

Auteurs :
SALEMI Marco
LECOCQ Alexis

Directeurs :
BUYS Alain
-

29 mai 2017

Table des matières

1 Introduction

Dans le cadre du cours de simulation, nous avons été amenés à réaliser un projet afin de mettre en pratique la théorie vue au cours.

Les objectifs du projet sont :

1. analyser le caractère aléatoire des décimales de pi par des tests vus au cours ;
2. utiliser ces décimales pour construire un générateur de loi uniforme dans l'intervalle [0, 1[;
3. comparer le générateur du point 2 avec celui utilisé par défaut dans Python.

Pour ce faire, un fichier nous est fourni. Celui-ci contient les 1 000 000 premières décimales du nombre pi.

Le projet doit être réalisé en python et nous avons opté pour la version 3.

Nous avons utilisé deux librairies externes :
- **scipy** : pour l'accès à la table des valeurs de Kolmogorov-Smirnov ;
- **plotly** : pour réaliser les graphiques.

L'installation de ces libraires est très simple à l'aide du gestionnaire de paquets pip3 :

```
$ pip3 install scipy plotly
```

Sur certains systèmes, les droits administrateurs sont requis pour installer ces librairies.

2 Les décimales de π

2.1 Test de χ^2

Le premier test consiste à étudier le nombre d'apparitions de chaque décimale. Si la séquence suit une loi uniforme, l'ensemble des décimales apparaissent exactement le même nombre de fois.

Résultats du test :

Décimales	Valeur attendue	Valeur observée
0	100000	99959
1	100000	99758
2	100000	100026
3	100000	100229
4	100000	100230
5	100000	100359
6	100000	99548
7	100000	99800
8	100000	99985
9	100000	100106

FIGURE 1 – Tableau des décimales

Graphique :

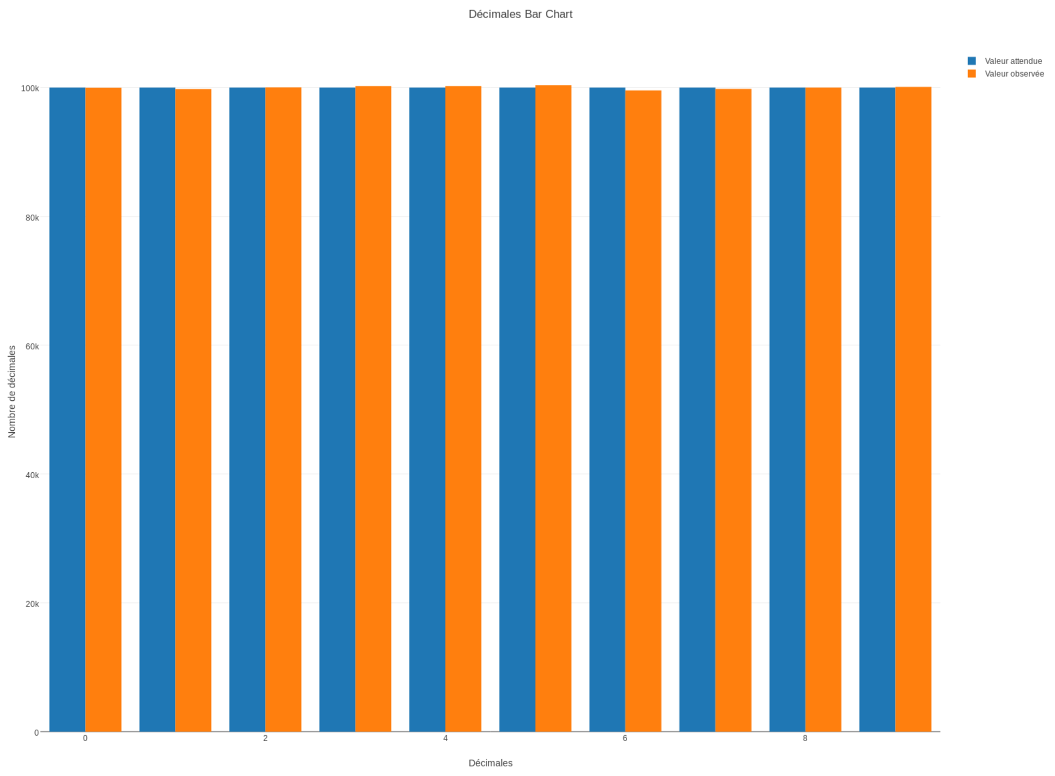


FIGURE 2 – Graphique des décimales

Test de χ^2 :

α	Valeur	Limite	Résultat
0.001	5.509	27.877	réussi
0.01	5.509	21.666	réussi
0.05	5.509	16.919	réussi
0.1	5.509	14.684	réussi

Figure 3 – Tableau du χ^2

Comme nous pouvons le voir dans le tableau, le test est réussi pour tous les α choisis.

2.2 Test du poker

Le test du poker consiste à considérer une suite de décimales comme une suite de paquets de l décimales. Dans chaque paquet, on compte le nombre de décimales différentes qui le composent. On regroupe ensuite les paquets par le nombre de décimales différentes qui les composent.

Dans le cas où la séquence de décimales suivrait une loi uniforme, la probabilité d'avoir r décimales différentes dans une séquence de longueur l est de :

$$\frac{\left\{ \begin{array}{c} l \\ r \end{array} \right\} \prod_{i=10-r+1}^{10} i}{10^l}$$

où $\left\{ \begin{array}{c} l \\ r \end{array} \right\}$ est le nombre de Stirling.

Pour obtenir la valeur théorique, on multiple la probabilité par le nombre total de paquets. On a opté dans ce test pour des paquets de 5 décimales.

Résultats du test :

# Décimales $\neq$	Valeur attendue	Valeur observée
1	20	13
2	2700	2644
3	36000	36172
4	100800	100670
5	60480	60501

FIGURE 4 – Tableau du Poker

Graphique :

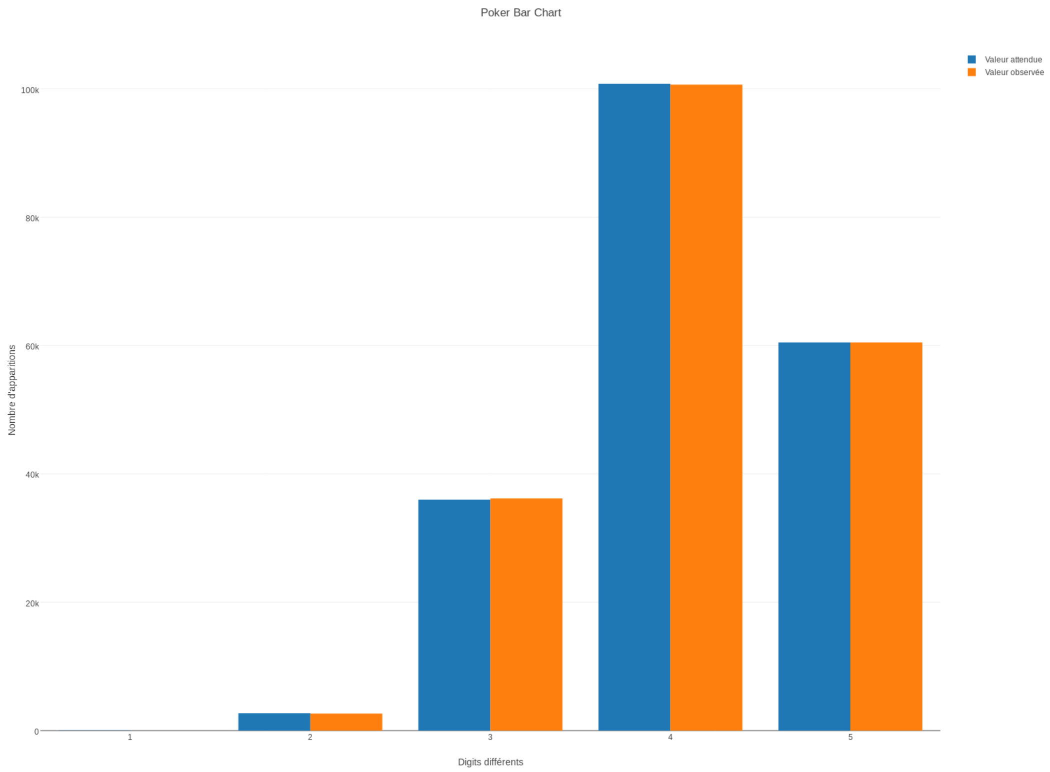


FIGURE 5 – Graphique du Poker

Test de χ^2 :

α	Valeur	Limite	Résultat
0.001	4.608	18.467	réussi
0.01	4.608	13.277	réussi
0.05	4.608	9.488	réussi
0.1	4.608	7.779	réussi

FIGURE 6 – Tableau du χ^2

Comme nous pouvons le voir dans le tableau, le test est réussi pour tous les α choisis.

2.3 Test du collectionneur de coupons

Le collectionneur de coupons est un test permettant de vérifier si nos différentes décimales de Pi suivent une loi uniforme.

Le fonctionnement de ce test que nous avons adapté à notre problème se déroule comme ceci :
- Nous parcourons les différentes décimales de Pi dans l'ordre, en calculant le nombres de digits visités.
- Lorsque nous avons rencontrés tous les différents digits (de 0 à 9) dans une séquence donnée de taille r, nous gardons la valeur r en mémoire.
- Nous recommençons ensuite le procédé avec les décimales qui suivent

Nous obtenons ainsi les différentes occurrences de séquences de tailles r_i contenant tout les différents digits.

Nous comparons ensuite cette valeur à la valeur théorique suivant une loi uniforme, et ceci à l'aide d'un χ^2.

Nous calculons la valeur théorique à l'aide de la probabilité S_r ci-dessous. Celle-ci représente la probabilité de rencontrer r digits avant d'avoir rencontré tout les différents digits possibles. r est donc la longueur de la séquence contenants les digits.

$$S_r = \frac{d!}{d^r}\left(\left\{ \begin{matrix} r \\ d \end{matrix} \right\} - d \left\{ \begin{matrix} r-1 \\ d \end{matrix} \right\}\right) = \frac{d!}{d^r}\left\{ \begin{matrix} r-1 \\ d-1 \end{matrix} \right\}$$

FIGURE 7 – Graphique du Poker

où d est le nombre de différents digits possibles (il vaut 10 dans notre cas)

Remarque : S_r sera égal à 0 lorsque r<d, donc lorsque r<10.

Ceci s'explique par le fait qu'il est impossible de rencontrer tout les différents digits, puisque la séquence est plus petite que le nombre des digits (qui vaut ici 10).

Nous obtenons donc le tableau de valeurs suivant pour les différentes longueurs de séquences , ainsi que le graphique correspondant, et le tableau représentant nos tests de χ^2. .

Collectionneur de coupons	Valeur attendue	Valeur observée
0-9	0	0
10	12.39307776	12
11	55.76884992	62
12	143.140048128	154
13	276.055807104	265
14	445.533624088	496
15	636.400653977	645
16	831.928596481	869
17	1017.20430344	1008
18	1180.91435762	1150
19	1315.83993579	1341
20	1418.52980752	1354
...	...	...
45	314.649867464	324
46	284.820911145	280
47	257.651373497	266
48	232.938892283	219
49	210.488959104	212
50	190.116510903	185
51	171.646916633	197
52	154.916499868	142
53	139.772710642	148
54	126.074036915	115
55	113.689727286	113
...	...	...
90	2.88994755473	0
91	2.60102568515	1
92	2.34098142576	4
93	2.10692993077	4
94	1.89627425595	0
95	1.7066766851	1
96	1.53603290049	1
97	1.38244871762	4
98	1.24421913165	0
99	1.11980944715	2
100	1.0078382854	0
101	0.907062283239	1

Figure 8 – Tableau du Collectionneur De Coupons 2

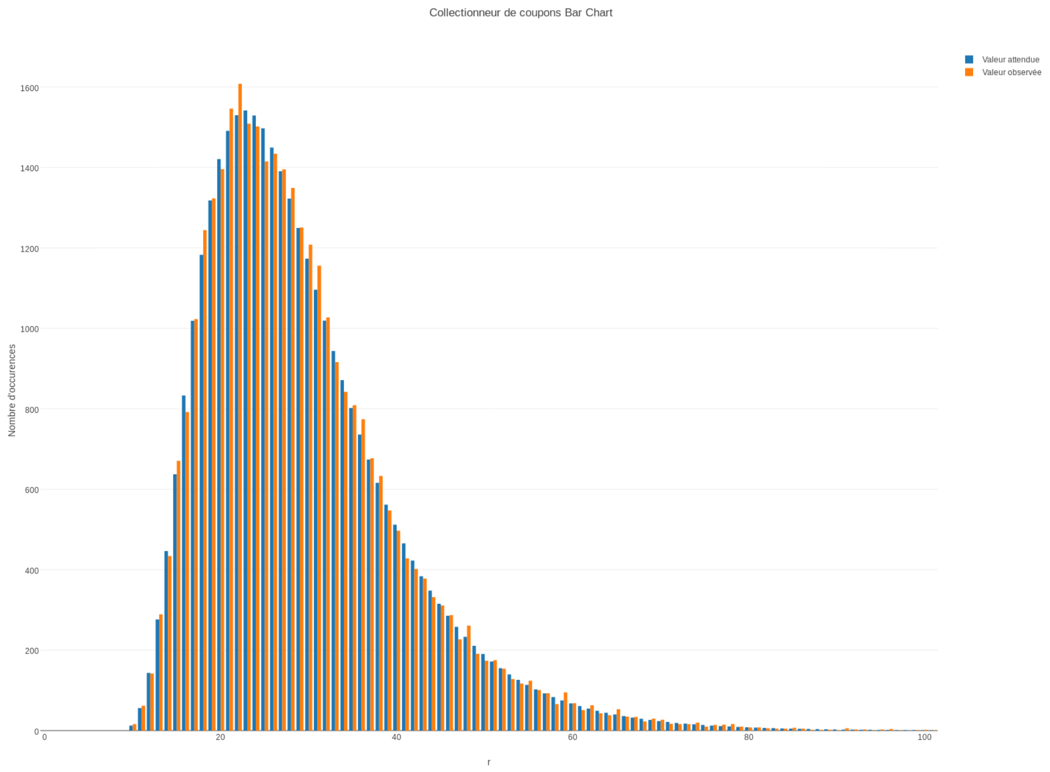


FIGURE 9 – Graphique du Collectionneur De Coupons

α	AValeur	Limite	Résultat
0.001	79.837	150.667055668	réussi
0.01	79.837	136.971003847	réussi
0.05	79.837	125.458419408	réussi
0.1	79.837	119.588667243	réussi

FIGURE 10 – Tableau de χ^2

Nous remarquons que les valeurs du tableau sont proches des valeurs théoriques. Ainsi que notre graphique suit la forme d'une gaussienne.

Ce test confirme donc que les décimales de Pi suivent bien une loi uniforme, car les différents tests de χ^2 sont respectés.

2.4 Interprétation des résultats

D'après les tests effectués ci-dessus, les décimales de pi suivent une loi uniforme.

3 Générateur de loi uniforme à partir des décimales de π

Notre générateur est très simple, il suit ces étapes :

1. lecture des n premiers chiffres à l'emplacement actuel comme un nombre ;
2. division de ce nombre par 10^n afin d'obtenir un nombre dans l'intervalle $[0, 1[$.

Les caractères éventuellement manquants à la fin du fichiers sont lus en début de fichier. Ainsi, le générateur ne s'arrête jamais.

Afin que le générateur ne commence pas toujours la séquence au même emplacement, nous choisissons l'emplacement de départ en fonction d'un timestamp. Ce timestamp représente le nombre de millisecondes écoulées depuis le 1er janvier 1970 UTC.

Notre générateur possédant un paramètre n, sa période est variable.

3.1 Choix du paramètre

Paramétrer notre générateur soulève une question : quel paramètre utiliser par défaut et pourquoi ?

Choisir un paramètre trop petit réduirait la précision du générateur. Par exemple si un seul digit est lu, alors il est évident que notre générateur ne générera que 10 nombres différents. Afin d'obtenir la meilleure précision possible, nous nous sommes fixé un minimum de 15 digits. Cela correspond à la précision utilisée par Python pour représenter u nombre flottant. Augmenter n au-delà de 15 n'améliorerait donc pas la précision.

Par ailleurs, choisir un paramètre qui divise 1000000 réduirait la période. Par exemple, si nous choisissons un paramètre de 100, alors il est évident qu'après 10000 générations, nous nous retrouverions au début du fichier. Cela réduirait la période à 10000. En fait, il convient d'utiliser un nombre qui est premier par rapport à 1000000 pour obtenir la période maximale de 1000000.

Notre premier choix s'est donc porté sur le nombre $n = 17$, qui est le premier nombre premier après 15. Nous avons également réalisé différents tests pratiques et ce choix du paramètre s'est avéré le plus judicieux. Afin de ne pas perturber le lecteur, nous n'avons pas inclus les tests des autres paramètres dans ce rapport. Cependant, ces derniers sont facilement réalisables à l'aide du code fourni en annexe.

4 Comparaison avec le générateur par défaut de Python

Nous allons maintenant comparer le caractère aléatoire de notre générateur avec celui utilisé par défaut dans Python (Mersenne Twister). Nous n'allons pas réutiliser ni le test du χ^2 ni le test du poker. Bien qu'il existe des méthodes de correspondance, d'autres tests sont plus appropriés pour tester des fonctions continues. Dans ces tests se trouve notamment le test de Kolmogorov-Smirnov.

4.1 Test de Kolmogorov-Smirnov

Le test de Kolmogorov-Smirnov est un test vérifiant si les valeurs générées pseudo-aléatoirement sont réparties uniformément entre 0 et 1.

Pour effectuer ce test, nous générons 1 000 000 de nombres avec notre générateur ainsi que 1 000 000 de nombres avec le générateur de Python. Ensuite, nous divisons l'intervalle [0, 1] en 100 valeurs réparties uniformément. Pour chaque valeur, nous analysons la proportion de nombres de chaque générateur se trouvant en dessous de cette valeur. Cette proportion correspond à la fonction de répartition et est notée $F_n(x)$. Dans le cas où le générateur suit une loi uniforme, la fonction de répartition serait :

$$F(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x > 1 \\ x & \text{sinon} \end{cases}$$

Nous comparons ensuite les fonctions de répartition pratiques $F_n(x)$ et théoriques $F(x)$ en notant le plus grand écart D_n. À l'aide de la table de Kolmogorov-Smirnov, nous pouvons lire la valeur critique D_α correspondant à notre nombre d'échantillons n et notre valeur critique α (analogue à celle du χ^2). Si $D_n > D_\alpha$, il convient de rejeter l'hypothèse selon laquelle ce générateur suit une loi uniforme.

Pour un nombre d'échantillons $n > 50$, il existe une formule pour calculer cette valeur critique :

$$D_\alpha = \sqrt{\frac{-\frac{1}{2} * \ln \frac{\alpha}{2}}{n}}$$

Nous pouvons également aller plus loin et comparer les D_n obtenus pour chaque générateur. Plus D_n est petit, plus le générateur se rapproche de la loi uniforme et meilleur il est.

Résultats du test :

x	$F(x)$	$F_n(x)$ Pi	$F_n(x)$ Python
0.01	0.01	0.009938	0.010077
0.02	0.02	0.019829	0.020109
0.03	0.03	0.029877	0.030099
0.04	0.04	0.039915	0.040026
0.05	0.05	0.049863	0.050101
0.06	0.06	0.059905	0.060362
0.07	0.07	0.069801	0.070405
0.08	0.08	0.079752	0.080280
0.09	0.09	0.089925	0.090444
0.10	0.10	0.099959	0.100334
...	...	...	...
0.45	0.45	0.449946	0.450557
0.46	0.46	0.459981	0.460541
0.47	0.47	0.470174	0.470473
0.48	0.48	0.480217	0.480434
0.49	0.49	0.490131	0.490429
0.50	0.50	0.500202	0.500360
0.51	0.51	0.510268	0.510548
0.52	0.52	0.520059	0.520405
0.53	0.53	0.530114	0.530331
0.54	0.54	0.540164	0.540338
0.55	0.55	0.550358	0.550515
...	...	...	...
0.90	0.90	0.899894	0.900212
0.91	0.91	0.909808	0.910198
0.92	0.92	0.919816	0.920301
0.93	0.93	0.929801	0.930276
0.94	0.94	0.939854	0.940290
0.95	0.95	0.950093	0.950184
0.96	0.96	0.960108	0.960128
0.97	0.97	0.969959	0.969998
0.98	0.98	0.979862	0.980039
0.99	0.99	0.989916	0.989981

FIGURE 11 – Tableau de Kolmogorov-Smirnov

Graphique :

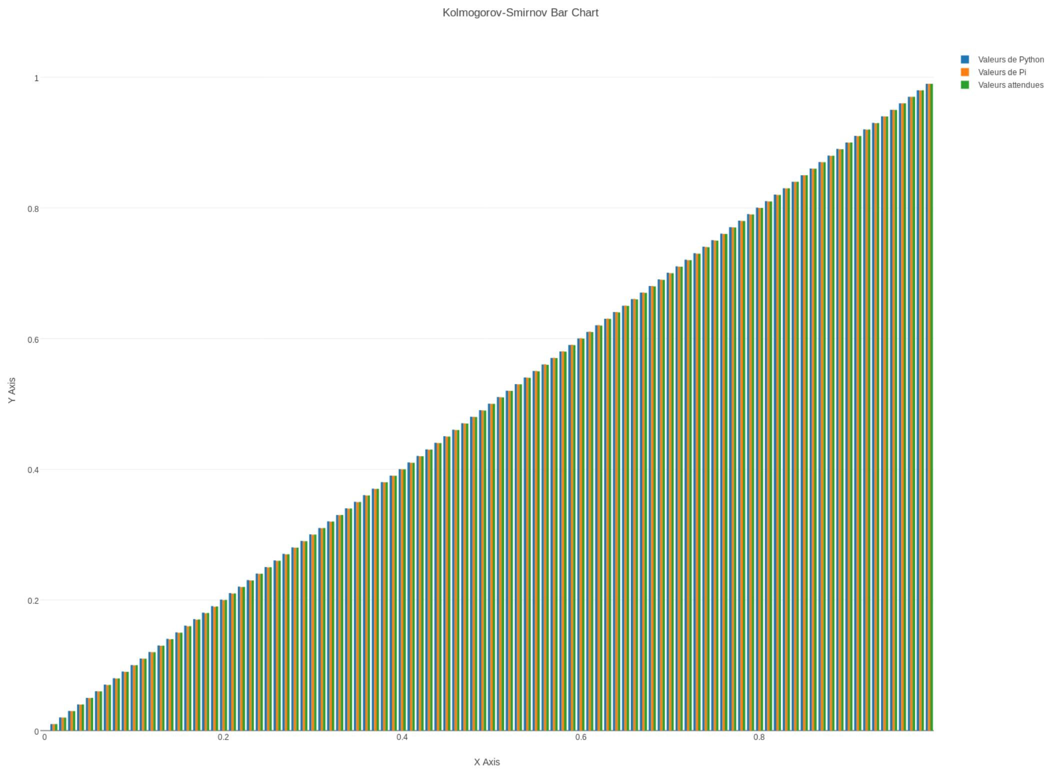


FIGURE 12 – Graphique de Kolmogorov-Smirnov

Écarts et valeurs critiques :

α	D_n Pi	D_n Python	D_α	Meilleur
0.001	0.000708	0.000849	0.001949	Pi
0.010	0.000708	0.000849	0.001628	Pi
0.050	0.000708	0.000849	0.001358	Pi
0.100	0.000708	0.000849	0.001224	Pi

FIGURE 13 – Tableau de Kolmogorov-Smirnov

Nous remarquons que les deux générateurs passent ce test avec tous les α choisis. Nous remarquons également que notre générateur fait mieux que le générateur de Python.

Attention cependant à l'interprétation hâtive des résultats. En effet, tant que nous n'avons pas étudié **tous** les nombres générés par les deux générateurs (l'entièreté de leur période), de nouveaux nombres pourraient apparaître lors d'une prochaine exécution. Nous pourrions donc obtenir un résultat différent.

4.2 Test du gap

Le test du gap est un test vérifiant la grandeur des trous (gap en anglais) séparant des nombres faisant partie d'un même intervalle.

Ce test se fait en différentes étapes :
- Nous générons n nombres à travers nos générateurs de nombres aléatoires.
- Nous choisissons un intervalle $[a, b] \in [0, 1]$ (nous avons ici choisis a=0 et b =1/2 pour un temps de calcul optimal).
- Nous marquons les nombres se trouvant dans cet intervalle
- Nous calculons les distances entre chaque nombres marqués (nous notons r_i les différentes distances.

Nous obtenons ainsi les différentes occurrences r_i que nous appelons les gaps (trous en anglais), et pouvons les comparer à l'aide d'un χ^2 avec les valeurs théoriques attendues :

$$r_i = Np(1-p)^i$$
$$r_{>i} = N(1-p)^{i+1}$$

où N est le nombres total de gaps observés

p est la probabilité d'être dans l'intervalle $|b - a|$

Nous avons donc effectuer ce test sur notre générateur pseudo-aléatoire et le générateur de python.

4.2.1 Notre générateur

Ci-dessous nous avons illustrer les occurrences pour les différents gaps obtenus dans un tableau et à l'aide d'un graphique. Nous avons fixé la limite à 15 afin d'éviter les classes vides, ces dernières étant néfastes au test de χ^2.

APi Gap	Valeur attendue	Valeur observée
0	250101.000	250067
1	125050.500	125248
2	62525.250	62273
3	31262.625	31471
4	15631.312	15616
5	7815.656	7860
6	3907.828	3826
7	1953.914	1898
8	976.957	954
9	488.479	471
10	244.239	252
11	122.120	135
12	61.060	66
13	30.530	30
14	15.265	19
> 15	15.265	16
Total	500202	500202

FIGURE 14 – Tableau de Pi Gap

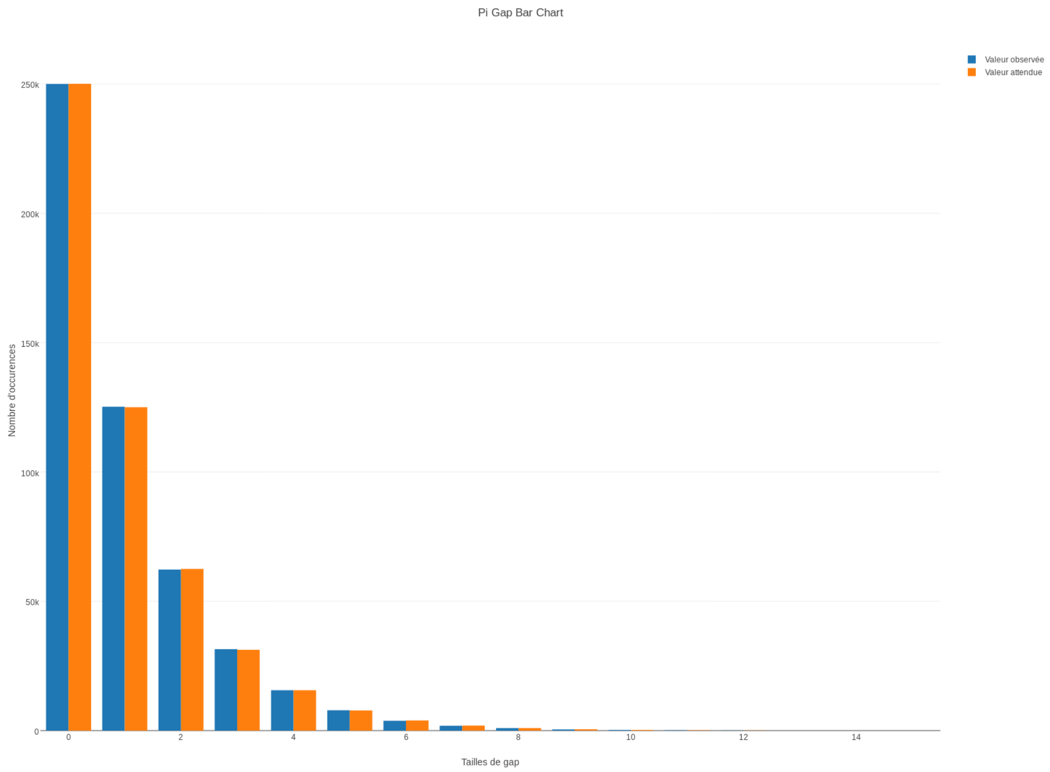


FIGURE 15 – Graphique de Gap

Résultats du test de χ^2 :

α	AValeur	Limite	Résultat
0.001	10.431	37.697	réussi
0.010	10.431	30.578	réussi
0.050	10.431	24.996	réussi
0.100	10.431	22.307	réussi

FIGURE 16 – Tableau de χ^2

Nous constatons donc que les différentes valeurs observées sont proches des valeurs théoriques. Et que le test de χ^2 réussit bien. Notre générateur passe donc ce test avec succès.

4.2.2 Le générateur de python

En ce qui concerne ce générateur, nous avons procédé de la même façon que ci-dessus pour notre générateur.

APython Gap	Valeur attendue	Valeur observée
0	249672.000	249152
1	124836.000	125037
2	62418.000	62606
3	31209.000	31237
4	15604.500	15686
5	7802.250	7839
6	3901.125	3834
7	1950.562	1969
8	975.281	942
9	487.641	522
10	243.820	254
11	121.910	122
12	60.955	78
13	30.478	31
14	15.239	19
> 15	15.239	16
Total	499344	499344

Figure 17 – Tableau de Python Gap

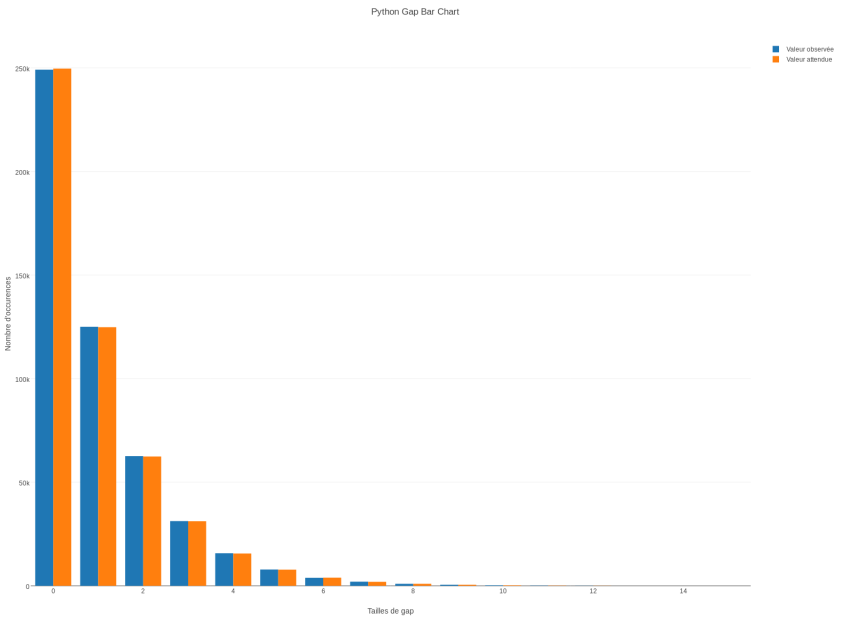


FIGURE 18 – Graphique de Gap

α	AValeur	Limite	Résultat
0.001	13.649	37.697	réussi
0.010	13.649	30.578	réussi
0.050	13.649	24.996	réussi
0.100	13.649	22.307	réussi

FIGURE 19 – Tableau de χ^2

Nous observons, ici aussi, que les valeurs sont proches des valeurs théoriques et que le test de χ^2 est réussi.

4.3 Test du collectionneur de coupons

Tout comme nous l'avons précédemment effectué sur les décimales de Pi, nous allons ici effectuer le même test sur le générateur de python.

Afin d'effectuer ce test comme précédemment, nous avons du avoir recourt à une petite adaptation.

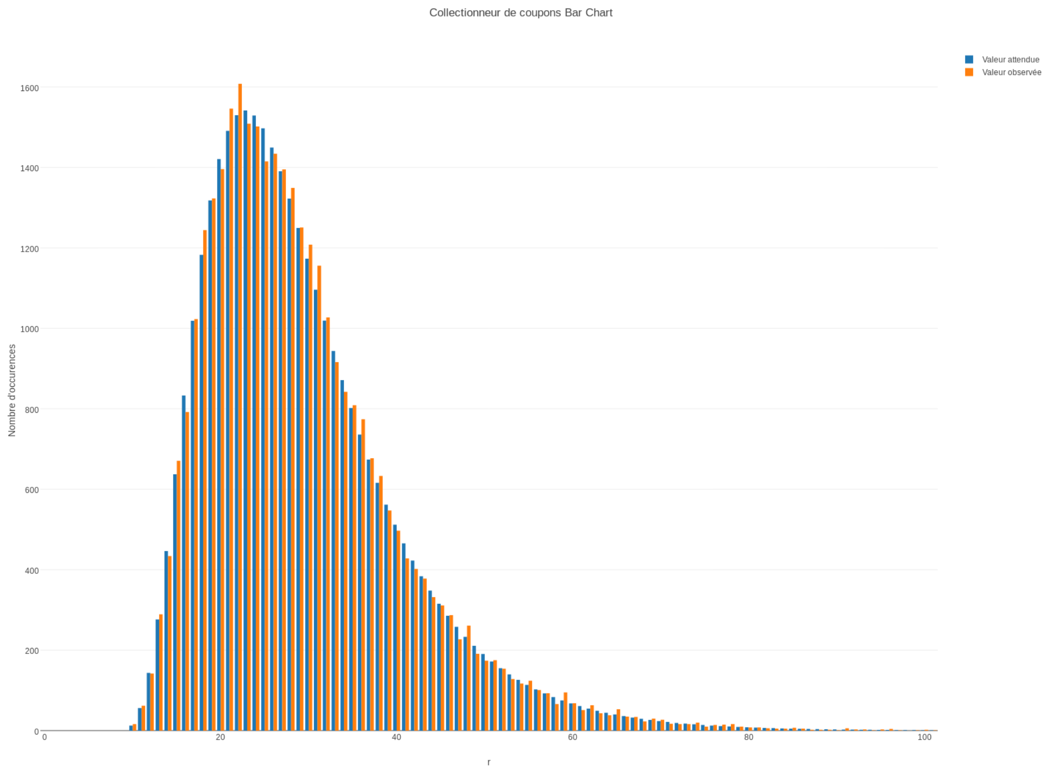
Nous avons donc discrétiser 1 million de nombres générés aléatoirement par python entre 0 et 1. Nous avons choisis ce nombre en rapport avec notre notre nombre de Pi qui possède 1 million de décimales.

Nous obtenons donc le tableau de valeurs, le graphique associé et les test de χ^2 suivants :

Collectionneur de coupons	Valeur attendue	Valeur observée
0-9	0	0
10	12.40614144	7
11	55.82763648	59
12	143.290933632	151
13	276.346800576	268
14	446.003265996	462
15	637.071490928	643
16	832.805541594	810
17	1018.27654972	1016
18	1182.15917248	1166
19	1317.22697718	1336
20	1420.02509544	1436
...	...	...
45	314.98154336	300
46	285.12114401	307
47	257.922966653	258
48	233.18443574	219
49	210.710837838	198
50	190.316914815	194
51	171.827851541	157
52	155.07979906	163
53	139.920046598	127
54	126.206932948	127
55	113.809568882	121
...	...	...
...	...	...
90	2.89299388033	3
91	2.60376745503	2
92	2.34344908011	3
93	2.10915086885	1
94	1.89827313956	2
95	1.70847571183	2
96	1.53765204971	1
97	1.38390597207	3
98	1.24553067677	0
99	1.12098985065	1
100	1.00890065885	0
101	0.9080184276	0

FIGURE 20 – Tableau de Collectionneur de coupons 2

Nous pouvons donc conclure, par les tests réussis, que le générateur suit lui aussi une loi uniforme.



α	AValeur	Limite	Résultat
0.001	70.467	150.667055668	réussi
0.01	70.467	136.971003847	réussi
0.05	70.467	125.458419408	réussi
0.1	70.467	119.588667243	réussi

FIGURE 21 – Tableau de χ^2

4.4 Interprétation des résultats

Malgré la simplicité de notre générateur, celui-ci donne de très bons résultats. Cela peut s'expliquer par le fait que nous n'avons effectué nos tests que sur un nombre limité de nombres générés. En effet, la période de notre générateur est de 200 000 alors que la période du générateur de Python (Mersenne Twister) est de 2^{19937}.

D'après les tests effectués ci-dessus, ...

5 Conclusion

Nous avons bien réalisé les objectifs fixés dans l'introduction, à savoir analyser le caractère aléatoire des décimales de pi, construire un générateur uniforme et le comparer au générateur par défaut de Python.

Nous avons ainsi eu l'occasion de mettre en pratique et d'approfondir les concepts vus au cours théorique notamment les test de χ^2, le test du poker, le test de Kolmogorov-Smirnov, ...

Nous tenons à remercier le titulaire BUYS Alain pour le dévouement dont il a fait preuve cette année.