Simulation sur ordinateur
Première session : Rapport
Etude du caractère pseudo-aléatoire de π

Maazouz Mehdi, Caredda Giuliano
BA3 Info

28 Mai 2018

Sommaire

1 Introduction

Dans le cadre du cours de *Simulation sur ordinateur* dispensé et corrigé par *Monsieur Alain BYUS*, tous les étudiants sont amenés à réaliser un travail.

Ce dernier consiste à :

- Etudier le caractère pseudo-aléatoire des décimales de π en utilisant les techniques vues au cours afin de déterminer si ces dernières suivent une loi uniforme.

-Implémenter un générateur de loi uniforme dans l'intervalle [0,1[à l'aide des décimales de π.

-Comparer ce dernier au générateur par défaut de Python.

Nous avons scindé le projet en 2 parties, la première concerne l'étude du caractère pseudo-aléatoire des décimales de π à l'aide de 3 tests différents.
La deuxième partie concerne non seulement l'explication et l'implémentation de notre générateur aléatoire, mais également sa comparaison avec le générateur par défaut de Python.

Remarque : un fichier contenant le premier million des décimales de π nous a été fourni.

1.1 Logiciels utilisés

-Le langage de programmation **Python** dans sa version **3.5** a été utilisé dans la réalisation de nos différents scripts.

-Le logiciel **Gnuplot 5.0** nous a servis dans la réalisation des différents histogrammes.

-Le logiciel libre **TexStudio 5.5.1** nous a permis de rédiger ce rapport.

-Notons enfin l'utilisation de modules de **Python** tel que **Math**, **Typing**, **Scipy**, **logging** ou encore **Matplotlib**.

Certains de ces modules sont issus de librairies externes et peuvent nécessiter une installation. Nous conseillons de les installer via le gestionnaire de paquets *pip3*. Attention, il se peut que l'installation requiert les droits administrateurs.

2 Etude de π

2.1 Approche Naïve

Pour notre première approche, nous allons tracer un histogramme contenant les occurrences de chacun des digits présents dans notre fichier fourni. Ensuite, nous comparerons ces occurrences observées avec les occurrences théoriques,attendues si nos données suivaient bien une loi uniforme.

Le nombre d'occurrence théorique de chaque digit peut être calculé assez facilement. En effet, nous avons 10 digits (de 0 à 9), chacun ayant une probabilité valant 1/10 d'apparaître. Comme notre fichier contient 1 000 000 de décimales, la valeur attendue pour chaque digit est de 100 000.

Regardons maintenant notre histogramme et débattons ensuite des premières constatations.

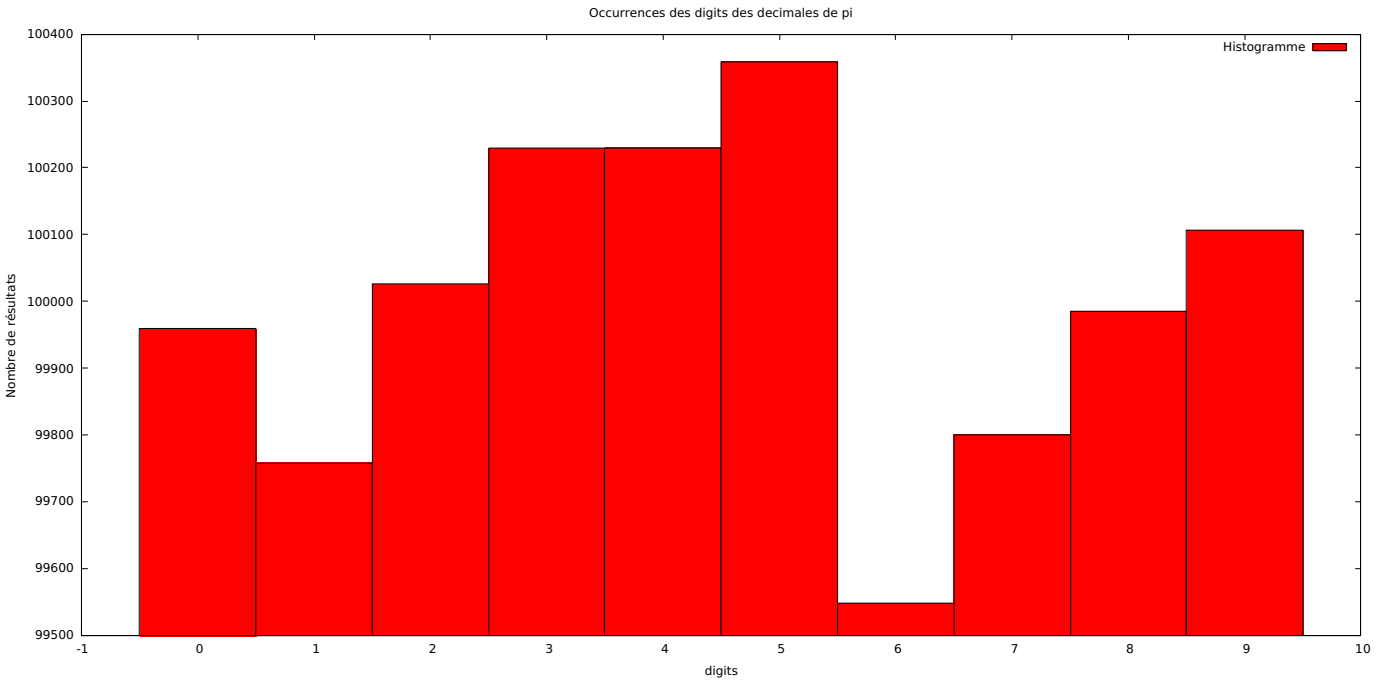


FIGURE 1 –

Mettons désormais les données dans un tableau afin de comparer les occurrences obtenues et théoriques.

	Résultats		
Digit	**Observé**	**Attendu**	**Erreur en %**
0	99 959	100 000	0.041
1	99 758	100 000	0.24
2	100 026	100 000	0.026
3	100 229	100 000	0.229
4	100 230	100 000	0.230
5	100 359	100 000	0.359
6	99 548	100 000	0.452
7	99 800	100 000	0.2
8	99 985	100 000	0.015
9	100 106	100 000	0.106

On peut constater qu'entre les résultats observés et attendus, il y a à chaque fois une erreur inférieur à 0,5%
Ce qui nous encourage à penser que nos données suivent bien une loi uniforme même si cela ne constitue pas une preuve.
Passons maintenant à des tests plus complets.

2.2 Test de χ^2

Le premier test que nous allons effectuer est le test du χ^2. Ce dernier est un test statistique qui va permettre de vérifier si une série de données est susceptible de suivre une loi de probabilité.

2.2.1 Rappel Théorique

Afin de vérifier si notre échantillon suit une loi, prenons cette loi comme hypothèse nulle H_0

$$K_n = \sum_{i=1}^{r} (\frac{n_i - Np_i}{\sqrt{Np_i}})^2 \; et \; on \; prend \; \chi^2_{r-1,1-\alpha}$$

-r correspond au nombre de classes.
-Np_i correspond aux nombres d'occurrences théoriques.
-n_i correspond aux nombres d'occurrences observées.
-Le paramètre α représente l'erreur de première espèce, autrement dit, la probabilité de rejeter l'hypothèse nulle H_0 alors que cette dernière est vraie.

-$\chi^2_{r-1,\alpha-1}$ le degré de liberté.

Si $K_n < \chi^2$, alors on accepte notre hypothèse nulle.
Si $K_n > \chi^2$, on la rejette.

2.2.2 Mise en pratique

Partons désormais de notre hypothèse nulle :

H_0 = nos décimales suivent une loi uniforme.

De plus :

-r sera fixé à 10, car nous considérerons une classe par digit.
-Np_i sera fixé à 100 000.
-Quant à n_i, ses différentes valeurs seront issues du nombre d'occurrence observés dans l'histogramme présenté précédemment.
-$\chi^2_{9,\alpha-1}$ où 9 est le degré de liberté calculé en faisant r-1 (nombre de classes -1).

Voici les différents résultats obtenus en fonction de différentes valeurs pour α.

	Résultats		
α	K_n	$\chi^2_{9,\alpha-1}$	**Resultat**
0.001	5.50908	27.877	Reussite
0.025	5.50908	19.023	Reussite
0.1	5.50908	14.684	Reussite

Nous pouvons constater que tous les tests ont réussi, nous sommes donc confortés dans notre supposition qui était que les décimales de π suivent une loi uniforme.

2.3 Test du Poker

Une fois le test du χ^2 effectué, passons maintenant à un autre test qui va s'appuyer sur ce dernier, le test du Poker.

Premièrement, nous allons diviser notre million de décimales en 200 000 blocs contenant chacun 5 décimales. Nous dirons que chaque bloc contient une séquence.

Une fois les blocs obtenus, nous allons les classer en fonction du nombre de digit différents par bloc. Ce nombre est appelé r et est compris entre 1 et 5.
En effet, une séquence peut ne contenir qu'un digit différent (par exemple :

11111) mais également 5 digits différents (12345).

Ensuite, nous passerons au calcul du χ^2 afin de comparer les occurrences obtenues pour chaque r avec les occurrences théoriques.

Afin d'obtenir les occurrences théorique, nous allons devoir calculer P_r, autrement dit, la probabilité d'avoir r.

$$P_r = \frac{\left\{ \begin{array}{c} k \\ r \end{array} \right\} d(d-1)...(d-r+1)}{d^k} \qquad (r \leq d)$$

Avec :

-K la longueur de la séquence.
-**d** le nombre de digits possibles.
-**r** le nombre de digits différents.

Remarque : nous pouvons constater que P_r se base sur le nombre de *Stirling*.
Ce dernier se définit comme étant le *nombre de manières possibles de constituer r paquets avec k nombres.*

$$\left\{ \begin{array}{c} k \\ r \end{array} \right\}$$

Les valeurs suivantes ont pu facilement être déterminées par les explications ci-dessus :

-$K = 5$.
-**d** $= 10$.

Une fois P_r calculé pour chaque r, multiplions ces dernières par le nombre de paquet total (200 000) pour avoir les occurrences théoriques.

	Résultats	
r	Occurrence observée	Occurrence théorique
1	13	20
2	2644	2700
3	36172	36000
4	100670	100800
5	60501	60480

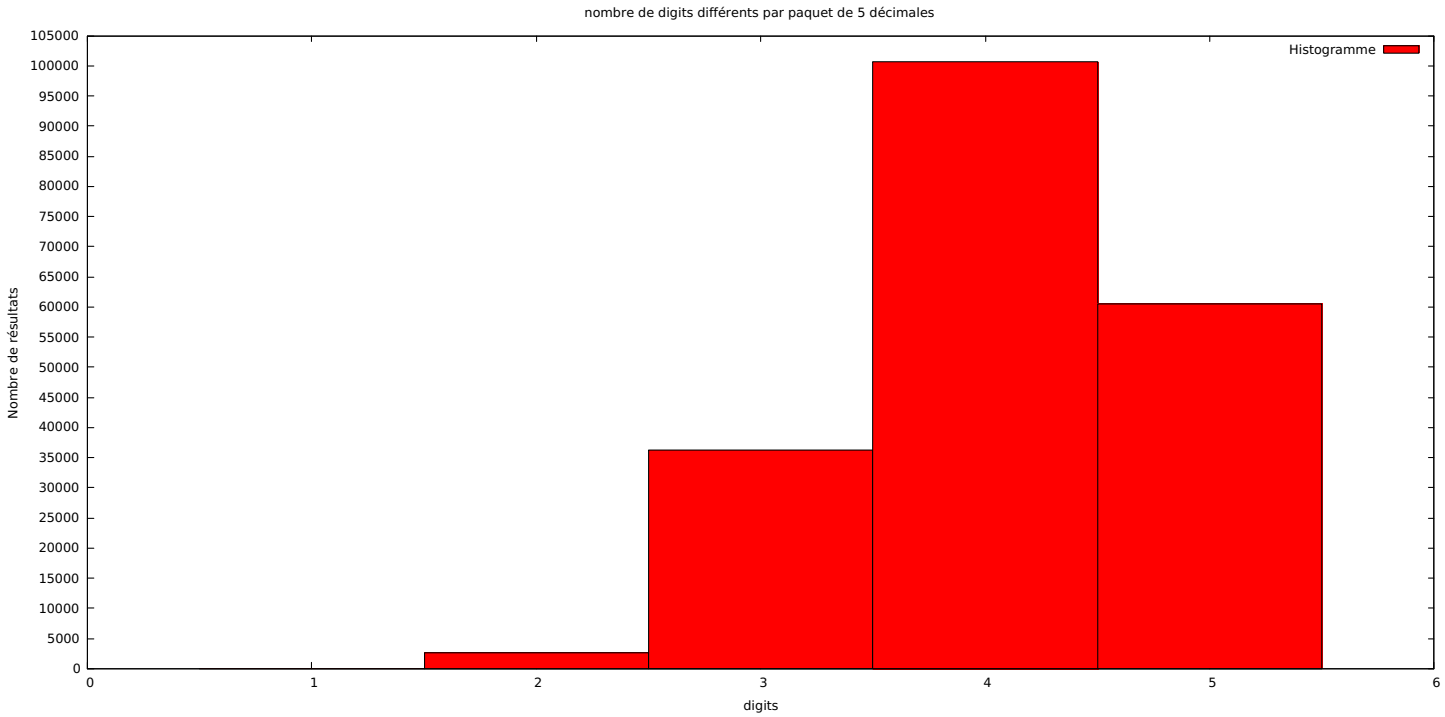


FIGURE 2 –

Après avoir adapté les différents paramètres, passons désormais au test de χ^2

	Résultats		
α	K_n	$\chi^2_{4,\alpha-1}$	**Resultat**
0.001	4.608	18.467	Reussite
0.025	4.608	11.143	Reussite
0.1	4.608	7.779	Reussite

Encore une fois, tous les tests ont réussis.

2.4 Test du collectionneur de coupons

Le dernier test que nous allons employer repose sur le principe suivant.

Tout d'abord, nous allons parcourir les décimales de π avec pour objectif de construire des séquences. Une séquence étant construite dès que l'on a parcouru une fois tous les digits possible (de 0 à 9 compris). Il suffit ensuite de recommencer la procédure avec les décimales suivantes.

On peut facilement comprendre que nous obtiendrons des séquences avec des tailles différentes.
Une fois fait, nous comparerons les occurrences observées pour chaque taille de séquence avec celles théorique grâce au test de χ^2.

Afin d'obtenir les occurrences théoriques, nous devons tout d'abord calculé la probabilité S_r, autrement dit, la probabilité d'avoir une séquence de taille r ayant tous nos digits est donnée par la formule suivante :

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

Avec :
-d représente le nombre de digits possibles, ici, il sera égal à 10.
-r la taille de la séquence.

Afin d'obtenir la valeur théorique pour chaque taille de séquence r_i, il suffit de multiplier la probabilité S_{r_i} par le nombre total de séquences obtenues.

Nos longueurs de séquences formeront nos classes pour le test de χ^2. Nous avons 100 classes, cependant, les séquences ayant une taille allant de 1 à 9 ont une valeur théorique et observée de 0. Elles sont donc inutiles à prendre en considération pour le test de χ^2.
Le degré de liberté sera donc égal à 91-1($\chi^2_{90,\alpha-1}$).

Tailles de séquences	Valeur théorique	Valeur observée
10	12.39706944	12
11	55.78681248	62
12	143.186152032	154
13	276.144721776	265
14	445.677125782	496
15	636.605631934	645
16	832.196551932	869
17	1017.53193425	1008
18	1181.29471772	1150
19	1316.26375399	1341
20	1418.98670105	1354
21	1489.0470501	1482
22	1528.21745078	1576
23	1539.67053158	1515
24	1527.32748565	1543
25	1495.36642995	1456
26	1447.88033297	1470
27	1388.65981367	1345
28	1321.07229861	1317
29	1248.01086229	1224
30	1171.89036844	1145
31	1094.67343335	1105
32	1017.91330149	1018
33	942.804564011	968
34	870.235668768	883
35	800.839427633	817
36	735.039345563	772
37	673.090709825	680
38	615.116112395	640
39	561.135537135	522
40	511.091408294	506
41	464.869130432	456
42	422.313697469	406
43	383.242942754	379
44	347.457965104	351
45	314.751212876	324
46	284.912648965	280
47	257.734360295	266
48	233.013919451	219
49	210.556755384	212
50	190.177745431	185

FIGURE 3 – Tableau de Test du CDC : Les décimales de Pi 1

Tailles de séquences	Valeur théorique	Valeur observée
51	171.702202299	197
52	154.966396843	142
53	139.817729962	148
54	126.114644036	115
55	113.726345551	113
56	102.532395165	87
57	92.4222090177	95
58	83.294505053	77
59	75.0567200775	80
60	67.624416886	69
61	60.9206957291	66
62	54.8756204189	62
63	49.4256662748	59
64	44.5131947077	44
65	40.0859574053	39
66	36.0966316837	34
67	32.5023875297	32
68	29.2644860878	22
69	26.3479087976	22
70	23.7210160015	27
71	21.3552335886	25
72	19.2247660837	27
73	17.3063345114	17
74	15.5789373362	8
75	14.0236327962	12
76	12.6233409926	12
77	11.3626641589	5
78	10.2277236148	13
79	9.20601199223	9
80	8.28625941323	12
81	7.45831238842	8
82	6.71302429704	7
83	6.04215639528	8
84	5.43828838508	4

FIGURE 4 – Tableau de Test du CDC : Les décimales de Pi 3

Tailles de séquences	Valeur théorique	Valeur observée
85	4.89473765491932	4
86	4.40548637952699	5
87	3.96511573604793	6
88	3.56874655969782	3
89	3.21198582270483	1
90	2.89087837643692	0
91	2.60186344817007	1
92	2.34173543125690	4
93	2.10760855073593	4
94	1.89688502594326	0
95	1.70722638771169	1
96	1.53652764052733	1
97	1.38289398981167	4
98	1.244619881547491	0
99	1.12017012599947	2

FIGURE 5 – Tableau de Test du CDC : Les décimales de Pi 3

Nous pouvons constater que le tableau affiche les résultats à partir des séquences de longueur 10. En effet, il est inutile de prendre en considération les séquences de longueur inférieure à 10 car il est impossible de trouver une séquence contenant les 10 digits différents (0 à 9) si la séquence a une longueur inférieure à 10.

Voici les résultats obtenues une fois le test de χ^2 effectué :

	Résultats		
α	K_n	$\chi^2_{9,\alpha-1}$	**Resultat**
0.001	78.792	137.21	Reussite
0.025	78.792	118.14	Reussite
0.1	78.792	107.57	Reussite

2.5 Conclusion

Tous les tests qui ont été effectués et qui avaient pour but de déterminer si les décimales de π suivaient bien une loi uniforme ont été concluants. Et ce, malgré les différentes marges d'erreurs que nous avions définis avec le paramètre α.

Nous pouvons donc affirmer, que d'après ces tests, les décimales de π suivent une loi uniforme.

3 Générateur de nombre pseudo-aléatoire

3.1 Enonce

La deuxième partie de notre projet consiste à implémenter un générateur de nombre pseudo-aléatoire et ensuite, de le comparer avec le générateur par défaut de Python (Mersenne Twister). Plus concrètement, nous devons implémenter un générateur qui devra, comme son nom l'indique, générer des nombres compris entre 0 et 1 de manière uniforme.

3.2 Implémentation

Nous savons que l'ordinateur est une machine déterministe. Mais étant donné que les premières décimales de π nous sont fournies, nous allons pouvoir nous en servir pour l'implémentation de notre générateur.
Tout d'abord, nous découpons les décimales de π fournie en plus petit nombre de 8 chiffres que nous stockons dans un tableau. Nous avons ensuite besoin d'une *seed* qui nous permettra de retomber sur les mêmes nombres lorsque la *seed* est la même. Soit nous donnons cette *seed* en paramètre lors de la création de l'objet *PseudoRandomGenerator*, soit elle est générée à partir des secondes écoulée depuis l'Epoch Unix.
Cette *seed* est ensuite coupée en deux et chaque morceau va servir d'index pour trouver une partie du nombre pseudo-aléatoire dans le tableau où nous avons stocké les fragment de décimale de π. Nous continuons la génération en concaténant un *0.* et les deux morceaux trouvé dans le tableau. Grâce à la concaténation du *0.* nous avons bien un nombre compris entre 0 et 1.

4 Comparaison des générateurs

4.1 Approche Naïve

Nous allons tout d'abord regarder naïvement un histogramme qui nous permettra de nous faire une idée de la répartitions des valeurs des nombres généré. Pour ce faire, nous allons générer 1 000 000 nombres via notre générateur et via celui de python. Ensuite, les valeurs seront trié par écart de 0.1. Les figures 6 et 7 sont ces histogrammes et pour plus de lisibilité les résultats sont dans les table 6 et 7.
Le nombre d'occurrence de chaque écart doit être d'environ 100 000 car nous avons 1 000 000 nombres réparti en 10 écart égaux.

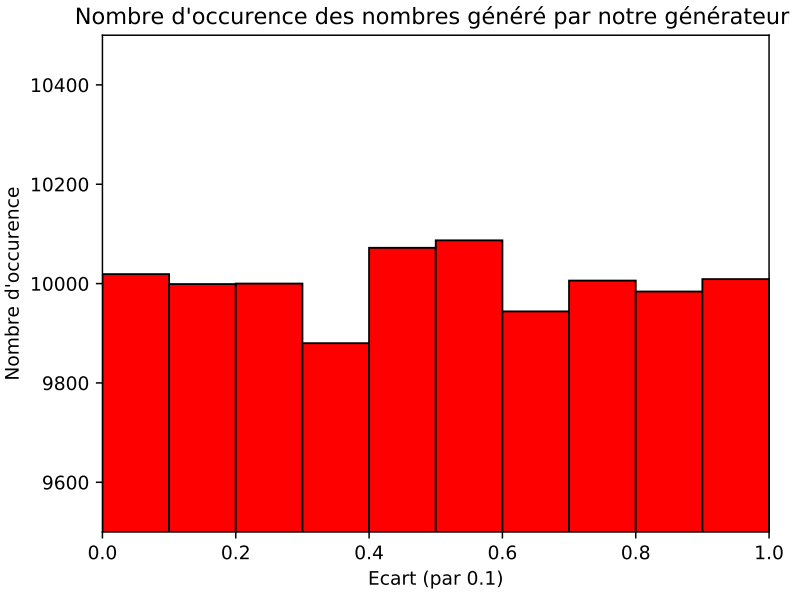
Nous remarquons que les erreurs sont globalement moindre pour notre générateur. L'erreur maximale atteinte par celui-ci est de 0.12%. Celle de notre générateur est de 0.197%. Passons maintenant à des tests plus complets.

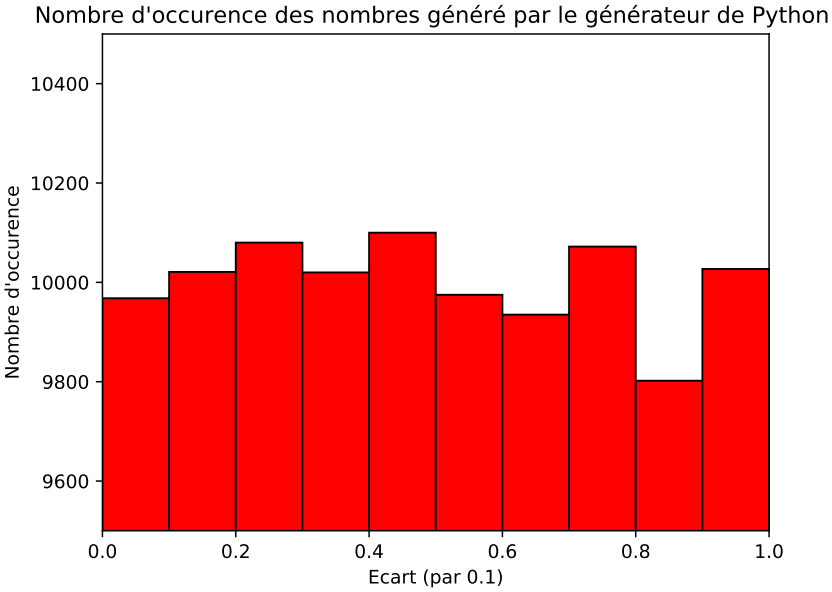
Table 6 – Résultat de notre générateur

	Résultats		
écart	Observé	Attendu	Erreur en %
0-0.1	10019	10000	0.019
0.1-0.2	9999	10000	0.001
0.2-0.3	10000	10000	0.0
0.3-0.4	9880	10000	0.12
0.4-0.5	10072	10000	0.072
0.5-0.6	10087	10000	0.087
0.6-0.7	9944	10000	0.056
0.7-0.8	10006	10000	0.006
0.8-0.9	9984	10000	0.016
0.9-1	10009	10000	0.009

Table 7 – Résultat du générateur Python

	Résultats		
écart	Observé	Attendu	Erreur en %
0-0.1	9964	10 000	0.036
0.1-0.2	10025	10 000	0.025
0.2-0.3	10078	10 000	0.078
0.3-0.4	10021	10 000	0.021
0.4-0.5	10100	10 000	0.1
0.5-0.6	9976	10 000	0.024
0.6-0.7	9935	10 000	0.065
0.7-0.8	10072	10 000	0.072
0.8-0.9	9803	10 000	0.197
0.9-1	10026	10 000	0.026





4.2 Test de χ^2

Comme pour l'étude de π, nous utilisons en premier le test de χ^2. Pour le rappel théorique voir 2.2

4.2.1 Mise en pratique

Partons désormais de notre hypothèse nulle :

H_0 = nos nombres généré suivent une loi uniforme.

De plus :

-r sera fixé à 10, car nous prenons un classe tout les 0.1 d'écart.
-Np_i sera fixé à 10 000.
-Quant à n_i, ses différentes valeurs seront issues du nombre d'occurrence observés dans l'histogramme présenté précédemment.
-$\chi^2_{9,\alpha-1}$ où 9 est le degré de liberté calculé en faisant r-1 (nombre de classes -1).

Les résultats suivant ont été obtenu pour un α valant 0.05. Pour notre générateur, nous obtenons $K_n = 3.102$ et $\chi^2_{9,\alpha-1} = 16.9189$ ce qui nous permet de constater que le test est réussi.
Pour le générateur de python, nous obtenons $K_n = 6.7916$ et $\chi^2_{9,\alpha-1} = 16.9189$ ce qui nous permet de constater qu'ici aussi, le test est réussi.

Pour compléter ce test, nous l'avons effectuer 100 fois sur les deux générateurs pour voir si il est toujours positif ou si il y a parfois des négatifs : voir 8 page 16

TABLE 8 – Pourcentage de réussite du test de χ^2

	réussi	échec
notre générateur	77	23
générateur de python	95	5

Nous remarquons que malgré des résultats plus intéressant lors de notre premier test, notre générateur a un taux d'échec fortement supérieur a celui de python.

4.3 Test du gap

4.3.1 Rappel théorique

Le test du gap permet de vérifier l'espace entre deux nombres du même intervalle.
Tout d'abord, nous génèrerons des nombres avec les deux générateurs. Nous utiliserons les séries de nombres générée pour l'approche naïve. Il nous faut ensuite une intervalle $[a, b] \in [0, 1]$ et nous marquons les nombres se trouvant dans celui-ci.
Il faut ensuite calculer les distances entre les nombres marqués(noté r_i). grâce à ces distances (gap en anglais), nous pouvons effectuer un test de χ^2 avec comme valeur attendues :

$$r_i = Np(1-p)^i$$

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

avec N, le nombre de gap p, la probabilité d'être dans l'intervalle $[a, b]$.

4.3.2 Mise en pratique

Les résultats avec les deux séries de nombre généré pour l'approche naïve et un $\alpha = 0.05$ sont : pour notre générateur $K_n = 5.372$ et $\chi^2_{9,\alpha-1} = 16.9189$. Le test est donc une réussite. Pour le générateur de python, nous obtenons $K_n = 10.42723$ et $\chi^2_{9,\alpha-1} = 16.9189$ ce qui nous permet de constater qu'ici aussi, le test est réussi.
Ici aussi, nous allons effectuer le test cent fois avec d'autres séries de nombres.

Table 9 – Pourcentage de réussite du test du gap

	réussi	échec
notre générateur	93	7
générateur de python	93	7

Avec ce test du gap, nous remarquons que les deux générateurs sont similaire dans les résultats.

4.4 Test du collectionneur de coupons

Comme pour l'étude de π, nous utilisons le test du collectionneur de coupons. Pour le rappel théorique voir 2.4

4.4.1 Mise en pratique

Les résultats avec les deux séries de nombre généré pour l'approche naïve et un $\alpha = 0.05$ sont : pour notre générateur $K_n = 40.0503$ et $\chi^2_{9,\alpha-1} = 48.6023$. Le test est donc une réussite. Pour le générateur de python, nous obtenons $K_n = 52.0544$ et $\chi^2_{9,\alpha-1} = 48.6023$. Le test est donc ici un échec.

Table 10 – Pourcentage de réussite du test du collectionneur de coupon

	réussi	échec
notre générateur	99	1
générateur de python	83	17

Nous remarquons que notre générateur a de meilleur résultat pour le test du collectionneur.

4.5 Interprétation

D'après les résultats des trois tests, nous remarquons que les deux générateurs respectent bien une loi uniforme.

De plus, nous remarquons que sur le test de χ^2, le générateur de python fait mieux alors qu'à contrario, notre générateur fait mieux sur le test du collectionneur de coupon.

5 Conclusion

Comme indiqué dans l'introduction, nous avons prouvé le caractère pseudo-aléatoire des décimales de π et montré qu'ils suivent une loi uniforme.
De plus, nous avons implémenté un générateur de loi uniforme et nous l'avons comparé avec le générateur par défaut de python.
Grâce a ce projet, nous avons pu mettre en pratique différents concepts vu au cours tel que le test de χ^2, le test du poker, du collectionneur de coupons et du gap.