Pi randomness

Rapport de projet de simulation

Année académique 2016-2017

Auteurs :
SALEMI Marco
LECOCQ Alexis

Directeurs :
BUYS Alain
-

27 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.

2 Les décimales de pi

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.

Décimales	Valeur attendue	Valeur observée
0	100000.0	99959
1	100000.0	99758
2	100000.0	100026
3	100000.0	100229
4	100000.0	100230
5	100000.0	100359
6	100000.0	99548
7	100000.0	99800
8	100000.0	99985
9	100000.0	100106

FIGURE 1 – Tableau des décimales

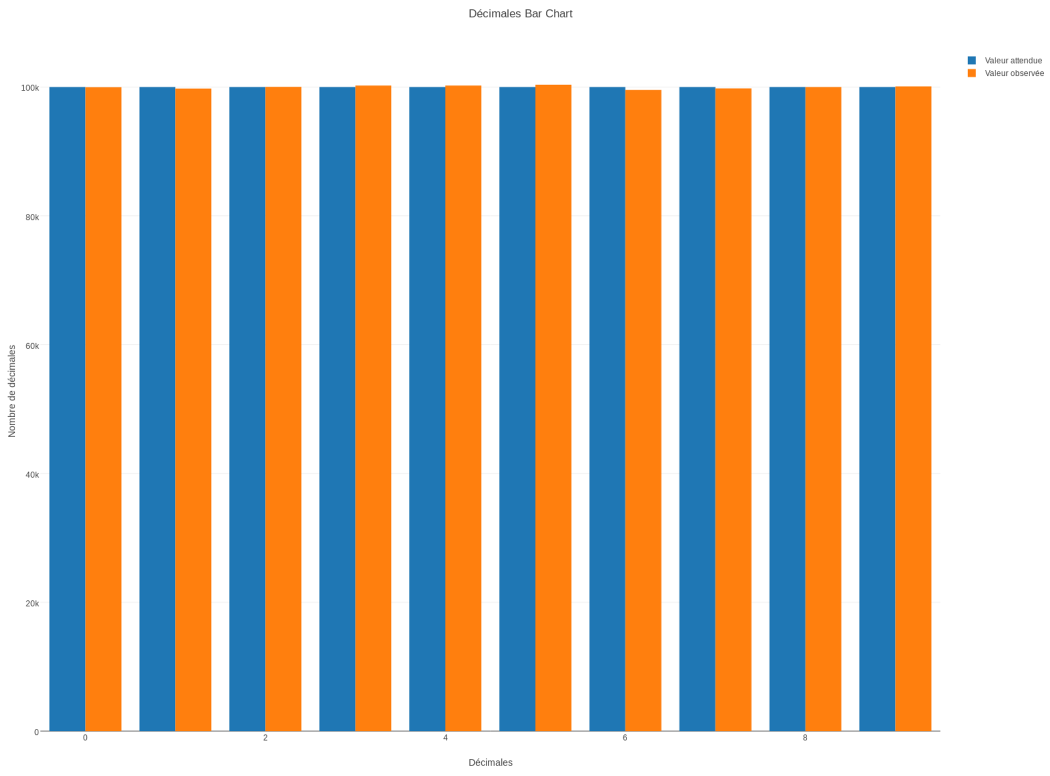

FIGURE 2 – Graphique des décimales

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

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

2.2 Test du poker

Le test du poker consiste à prendre une suite de décimales (ici 5) et calculer le nombre de décimales différentes qui composent cette suite. Si la séquence suit une loi uniforme, la probabilité d'avoir r chiffres différents dans une séquence de longueur l est :

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

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

Poker	Valeur attendue	Valeur observée
1	20	13
2	2700	2644
3	36000	36172
4	100800	100670
5	60480	60501

FIGURE 4 – Tableau du Poker

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

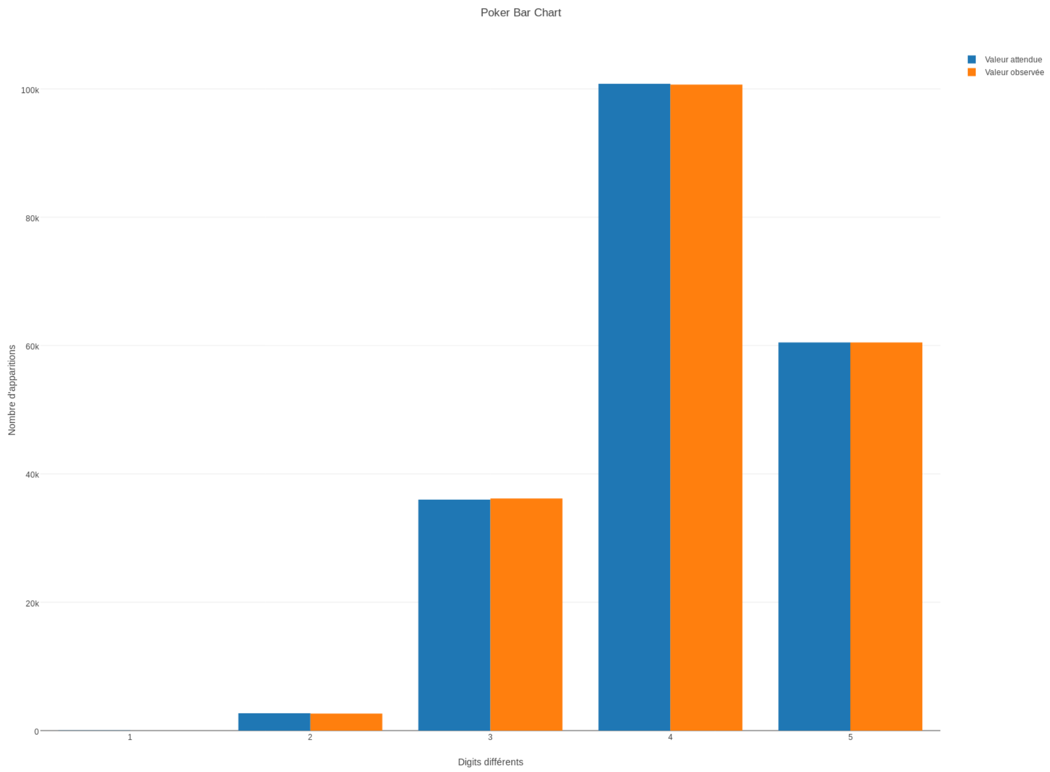

FIGURE 5 – Graphique du Poker

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

2.3 Interprétation des tests

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

3 Générateur de loi uniforme

Notre générateur est très simple, il suit ces étapes :

1. lecture des 15 premiers chiffres à l'emplacement actuel comme un nombre ;
2. division de ce nombre par 10^{15} afin d'obtenir un nombre dans l'intervalle [0, 1[.

Quand nous arrivons à la fin du fichier, nous revenons au début pour que 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.

La période de ce générateur est de 200 000. En effet, si le premier nombre est extrait du début du fichier, après 66 667 ($\lceil \frac{1000000}{15} \rceil$) mouvements de 15 caractères, nous nous retrouvons à 5 caractères après le début du fichier. Après 66 667 mouvements, nous nous retrouvons 10 caractères après le début du fichier. Si après 66 667 mouvements, nous aurions été 15 après le début du fichier, après 66 666 mouvements, nous revenons au début du fichier. Nous avons donc lu 3*66 666 +2 nombres aléatoires avant de relire le premier.

En tant que bons programmeurs, nous avons paramétré le nombre de digits lus pour générer un nombre aléatoire. Ainsi, le programmeur peut décider lui-même la balance entre une grande précision pour les nombres générés ou une grande période. En effet, si moins de digits sont lus, la précision d'un nombre sera plus petite mais la période aura tendance à s'agrandir.

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

Pour effectuer ce test, nous générons 100 000 nombres avec notre générateur ainsi que 100 000 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. Nous comparons la proportion de chaque générateur avec la proportion attendue d'une loi uniforme et nous en prenons le plus grand écart (D_n).

À l'aide de la table de Kolmogorov-Smirnov, nous pouvons accepter ou refuser l'hypothèse disant que les générateurs suivent une loi uniforme.

Nous pouvons également aller plus loin et comparer les D_n obtenus pour chaque générateur Il est évident que le générateur ayant le plus petit D_n sera celui se rapprochant d'avantage de la loi uniforme.

X	Valeurs attendues	Valeurs de Pi	Valeurs de Python
0.01	0.01	0.01008	0.01019
0.02	0.02	0.02001	0.02041
0.03	0.03	0.03026	0.03055
0.04	0.04	0.03993	0.04118
0.05	0.05	0.04986	0.05164
0.06	0.06	0.05944	0.06145
0.07	0.07	0.06959	0.07172
0.08	0.08	0.0794	0.08185
0.09	0.09	0.08891	0.09194
0.1	0.1	0.09909	0.10154
...	...	...	...
0.45	0.45	0.44995	0.45082
0.46	0.46	0.4601	0.46057
0.47	0.47	0.47026	0.47057
0.48	0.48	0.48063	0.48064
0.49	0.49	0.49032	0.49075
0.5	0.5	0.50059	0.50101
0.51	0.51	0.51073	0.51043
0.52	0.52	0.52031	0.52009
0.53	0.53	0.53065	0.53072
0.54	0.54	0.54064	0.54095
0.55	0.55	0.55076	0.55057
...	...	...	...
0.9	0.9	0.8998	0.90023
0.91	0.91	0.90979	0.91049
0.92	0.92	0.91907	0.92049
0.93	0.93	0.92897	0.93025
0.94	0.94	0.93927	0.94029
0.95	0.95	0.94987	0.95097
0.96	0.96	0.95941	0.96029
0.97	0.97	0.96985	0.96935
0.98	0.98	0.9798	0.97962
0.99	0.99	0.98989	0.98957

Figure 7 – Tableau de Kolmogorov-Smirnov

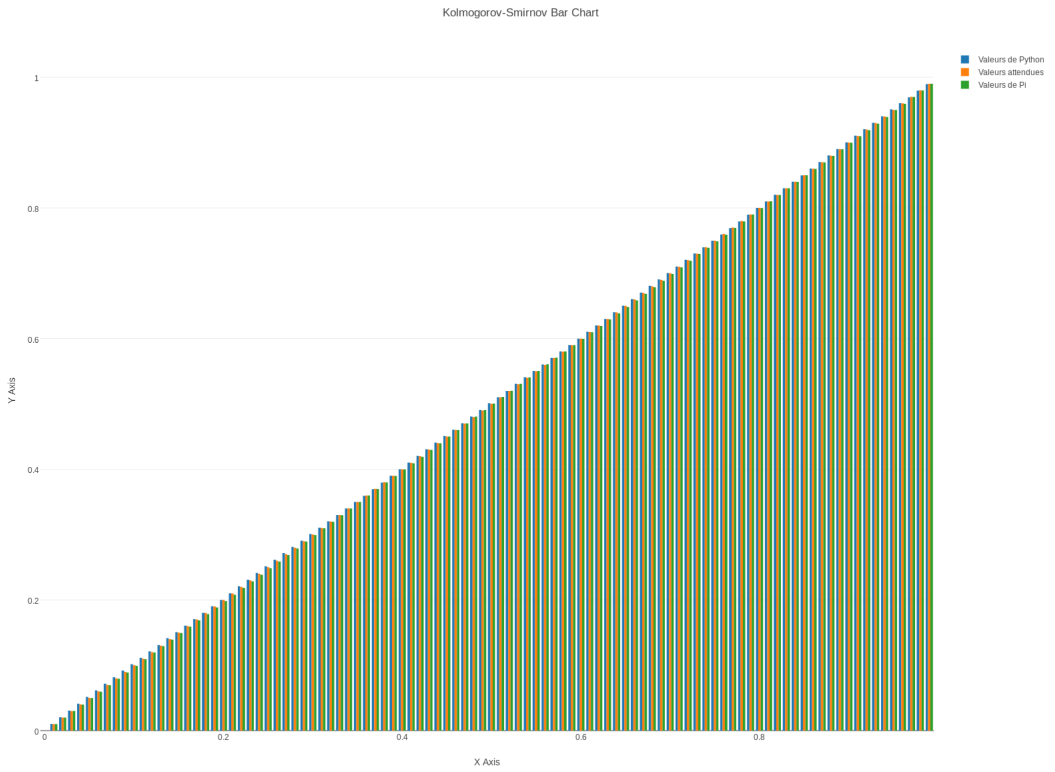

Figure 8 – Graphique de Kolmogorov-Smirnov

α	Valeur Pi	Valeur Python	Limite	Meilleur
0.001	0.00212	0.00194	0.006165	Python
0.01	0.00212	0.00194	0.005147	Python
0.05	0.00212	0.00194	0.004295	Python
0.1	0.00212	0.00194	0.00387	Python

Figure 9 – Tableau des D_α

Nous remarquons que le générateur de Python est meilleur dans ce test. Cependant, si on répète ce test, il est possible d'obtenir des résultats différents. En effet, tant que le nombre de nombres générés sera inférieur au plus petit commun multiple entre la période des deux générateurs, nous obtiendrons des nombres et donc des résultats différents.

4.2 Test de gap

Le test de gap est un test vérifiant si les valeurs générés pseudo-aléatoirement sont réparties uniformément entre 0 et 1. 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 :

$$r_i = N.(p.(1-p)^i)$$

où N est le nombres total de gaps observés

p est la probabilité d'être dans l'intervalle qui vaut b-a

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

4.2.1 Notre générateur sur Pi

Ci-dessous nous avons illustrer les occurrences pour les différents gaps obtenus dans un tableau et à l'aide d'un graphique. Pour tout les r_i plus grands que 21, les résultats sont proches ou égal à 0, ils ne sont donc pas significatifs et il n'est pas important de les mettre dans le tableau.

r_i	Valeur attendue	Valeur observée
0	250101.0	250067
1	125050.5	125248
2	62525.25	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	122.12	135
12	61.06	66
13	30.53	30
14	15.265	19
15	7.632	7
16	3.816	4
17	1.908	1
18	0.954	2
19	0.477	1
20	0.239	1
21	0.119	0

FIGURE 10 – Tableau de Gap

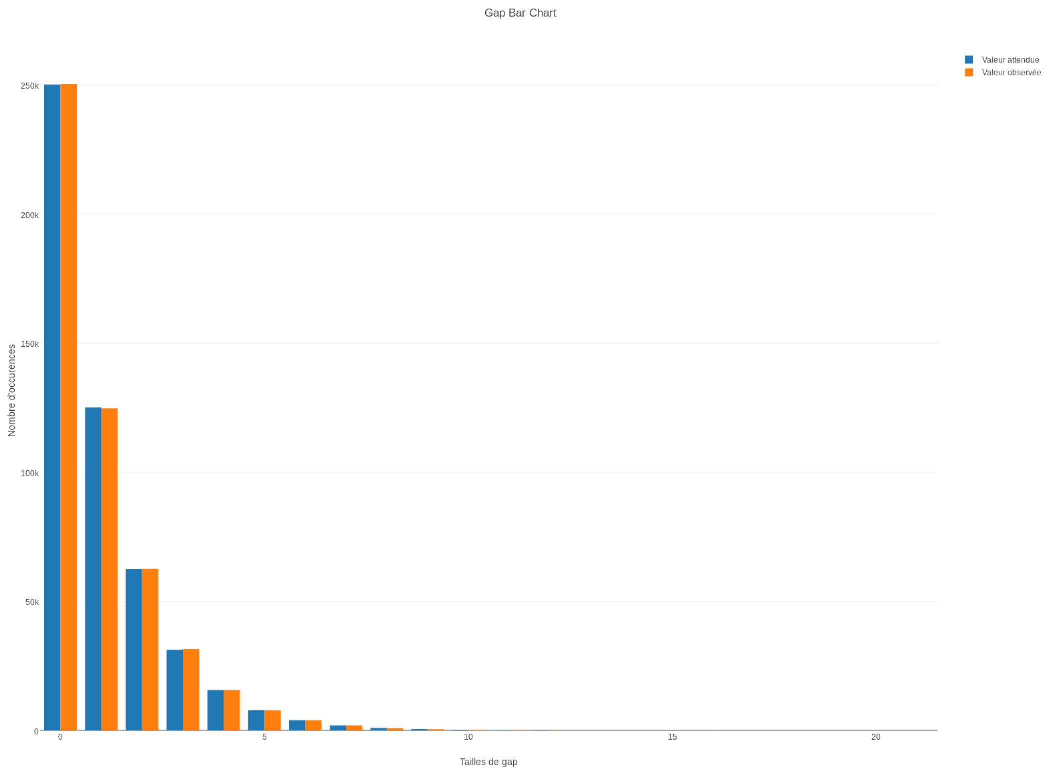

FIGURE 11 – Graphique de Gap

Ci-dessous nous avons notre test de χ^2 :

α	AValeur	Limite	Résultat
0.001	15.16	46.7970380416	réussi
0.01	15.16	38.9321726835	réussi
0.05	15.16	32.6705733409	réussi
0.1	15.16	29.6150894362	réussi

FIGURE 12 – 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 est donc bien répartit uniformément selon ce test.

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 sur Pi. Nous avons aussi, pour les mêmes raisons que nous avons cité précédemment, éviter d'afficher les valeurs supérieur à 21.

Notre tableau correspondant se trouve à la figure 13, le graphique associé à la figure 14, et notre test de χ^2 se trouve à la figure 15.

r_i	Valeur attendue	Valeur observée
0	249770.5	249636
1	124885.25	124785
2	62442.625	62609
3	31221.312	31076
4	15610.656	15690
5	7805.328	7790
6	3902.664	3987
7	1951.332	2030
8	975.666	982
9	487.833	478
10	243.917	229
11	121.958	135
12	60.979	73
13	30.49	17
14	15.245	10
15	7.622	9
16	3.811	5
17	1.906	0
18	0.953	0
19	0.476	0
20	0.238	0
21	0.119	0

FIGURE 13 – Tableau de Gap

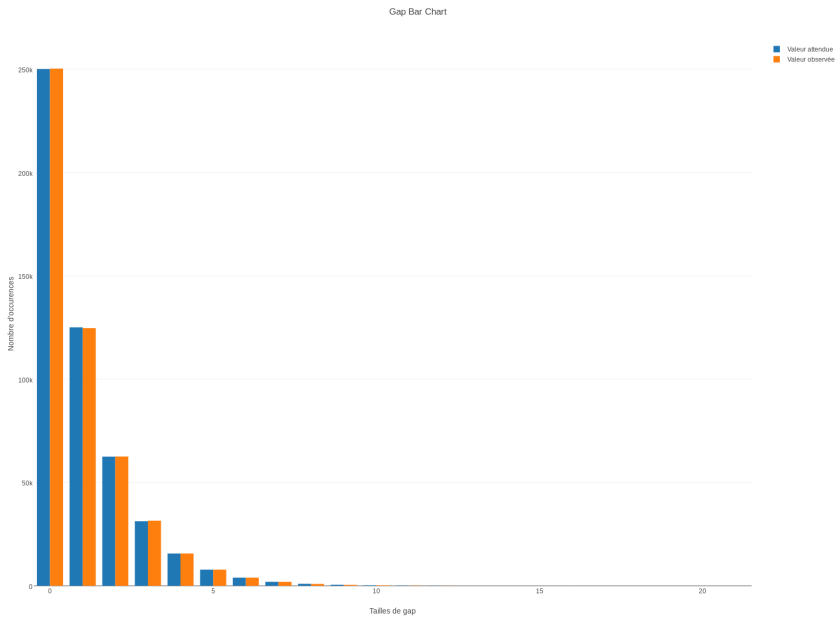

FIGURE 14 – Graphique de Gap

α	AValeur	Limite	Résultat
0.001	17.766	46.7970380416	réussi
0.01	17.766	38.9321726835	réussi
0.05	17.766	32.6705733409	réussi
0.1	17.766	29.6150894362	réussi

FIGURE 15 – Tableau de χ^2

Nous observons donc que nos valeurs sont elles aussi proches des valeurs théoriques. Et que notre test de χ^2 est réussit. Le générateur de python est donc bien répartit uniformément lui aussi.

4.3 Le 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.

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

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. .

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.

4.4 Interprétation des tests

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.

ACollectionneur de coupons	Valeur attendue	Valeur observée
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	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
21	1488.56759813	1482
22	1527.7253865	1576
23	1539.17477958	1515
24	1526.83570793	1543
25	1494.88494323	1456
26	1447.4141361	1470
27	1388.21268496	1345
28	1320.64693213	1317
29	1247.60902054	1224
30	1171.51303642	1145
31	1094.32096408	1105
32	1017.58554788	1018
33	942.500994353	968
34	869.955465262	883
35	800.58156873	817
36	734.802673351	772
37	672.873984191	680
38	614.918053757	640
39	560.954859475	522
40	510.926844132	506
41	464.719449185	456
42	422.177718467	406
43	383.119543979	379
44	347.346088582	351
45	314.649867464	324
46	284.820911145	280
47	257.651373497	266
48	232.938892283	219
49	210.488959104	212

FIGURE 16 – Tableau de Collectionneur de coupons 1

r	Valeur attendue	Valeur observée
50	190.116510903	185
51	171.646916633	197
52	154.916499868	142
53	139.772710642	148
54	126.074036915	115
55	113.689727286	113
56	102.499381192	87
57	92.3924503812	95
58	83.2676854074	77
59	75.0325528813	80
60	67.6026427858	69
61	60.9010801317	66
62	54.8579512498	62
63	49.4097519134	59
64	44.4988620923	44
65	40.0730502972	39
66	36.0850090818	34
67	32.4919222233	32
68	29.2550633396	22
69	26.3394251458	22
70	23.7133781718	27
71	21.3483575072	25
72	19.2185759825	27
73	17.3007621179	17
74	15.57392114	8
75	14.019117386	12
76	12.6192764564	12
77	11.3590055427	5
78	10.2244304333	13
79	9.20304778734	9
80	8.28359135557	12
81	7.45591091794	8
82	6.71086279872	7
83	6.04021090687	8
84	5.43653733358	4
85	4.89316161903	4
86	4.40406787558	5
87	3.96383902519	6
88	3.56759747407	3
89	3.21095160896	1
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 17 – Tableau de Collectionneur de coupons 2

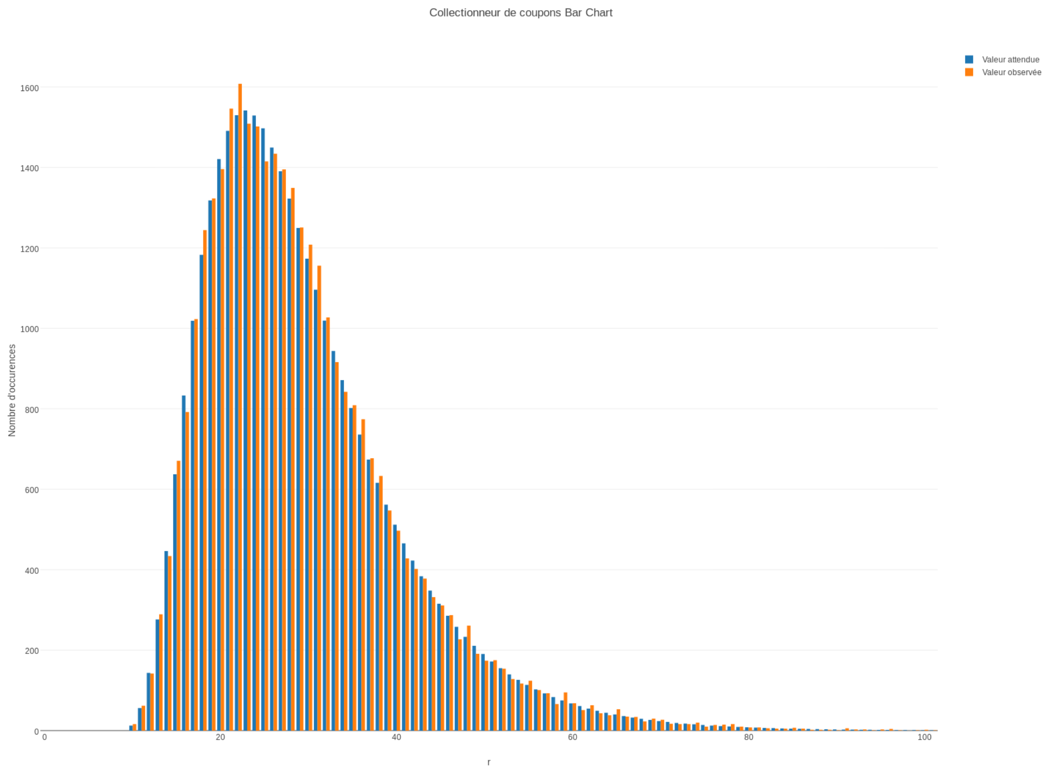

FIGURE 18 – Graphique de Collectionneur de coupons

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 19 – Tableau de χ^2