Simulation

Gallet Ewen
Van Herzeele Maxime
BA3 Info

juin 2016

Contents

Chapter 1

Introduction

1.1 But du projet

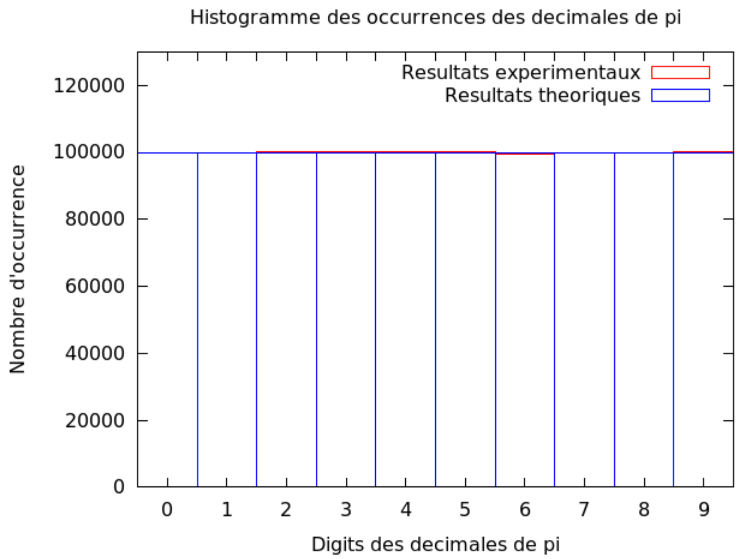
Le but de ce projet est de constater le caractères pseudo aléatoire des 1 000 000 premières décimales de π et prouver que celles-ci suivent une loi uniforme. Ensuite nous créerons à partir de ces mêmes décimales, un générateur de nombre pseudo-aléatoire qui sera comparé avec celui du language python.

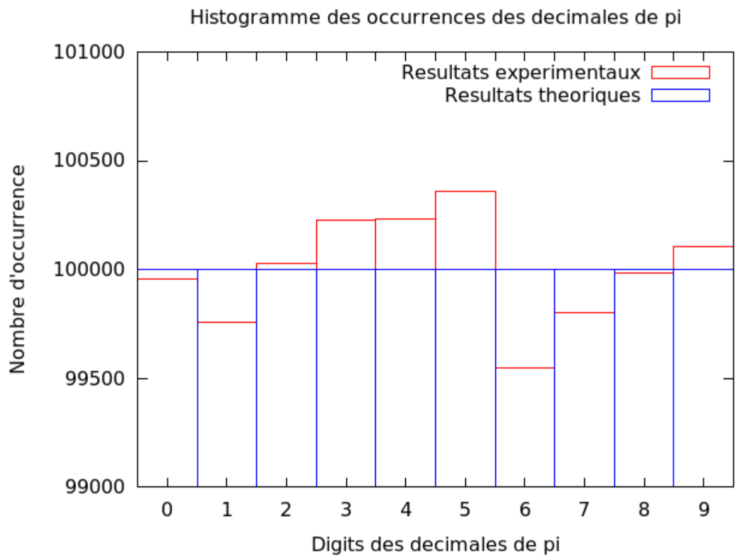
1.2 Outils utilisés

Python: Interpréteur utilisé pendant le projet pour implémenter les test vu en cours pour l'analyse des décimales de π ainsi que pour l'analyse du générateur de nombre aléatoire.
Gnuplot: Logiciel libre permettant de tracer des graphiques (dans notre cas, les histogrammes).

1.3 Quelques constatations avant de commencer le projet

On s'attend à ce que les décimales de pi suivent une loi uniforme puisque π ne contient pas de cycle et un nombre infini de décimales. Théoriquement, on devrait retrouver 100 000 fois chaques occurences de chiffre dans ces décimales(100 000 fois le chiffre 0, 100 000 fois le chiffre 1, ...)pour les 1 000 000 premières décimales.


Histogramme des occurrences des decimales de pi
Resultats experimentaux
Resultats theoriques
Nombre d'occurrence
120000
100000
80000
60000
40000
20000
0
0 1 2 3 4 5 6 7 8 9
Digits des decimales de pi


Histogramme des occurrences des decimales de pi
Resultats experimentaux
Resultats theoriques
Nombre d'occurrence
101000
100500
100000
99500
99000
0 1 2 3 4 5 6 7 8 9
Digits des decimales de pi

	Théorique	Expérimentaux	Taux d'erreur
0	100.000	99959	0.0410168168949
1	100.000	99758	0.242587060687
2	100.000	100026	-0.0259932417571
3	100.000	100229	-0.228476788155
4	100.000	100230	-0.229472213908
5	100.000	100359	-0.357715800277
6	100.000	99548	0.45405231647
7	100.000	99800	0.200400801603
8	100.000	99985	0.0150022503375
9	100.000	100106	-0.105887758975

En conclusion on voit que les décimales de π semblent suivre une loi uniforme avec un taux d'erreur nettement inférieur a 1 pourcent.

Chapter 2

Test sur les Décimales de π

2.1 Test du χ^2

Le test du chi^2 est un test stastique qui permetra de determiner si notre série de donnée, c-à-d les décimales de pi, suit bien la loi uniforme comme nous le pensons. Les données doivent être triées par classse d'évènement (ici chaque digit correspond à un évènement)et les valeurs théoriques(100.000 pour chacune) vont être comparés aux valeurs expérimentales. La formule du χ^2 est la suivante:

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

avec:

- r le nombre d'évenement distinc observé
- n_i le nombre de fois que l'évènement i est observé.
- N est la taille de l'échantillon: 1 000 000
- p_i est la probabilité d'avoir l'évènement i(ici 1/10).

Par conséquent, Np_i est donc le nombre d'occurrence théorique de l'évènement i. On se donne par la suite une certaine probabilité α, erreur de première espèce, de rejeter l'hypothèse nulle H_0.
H_0 est acceptée si $K_n \leq \chi^2_{9,1-\alpha}$ avec r-1 le degré de liberté. Ici le degrés de liberté vaut donc 9 puisque r=10(10 chiffres).

2.1.1 Résultats

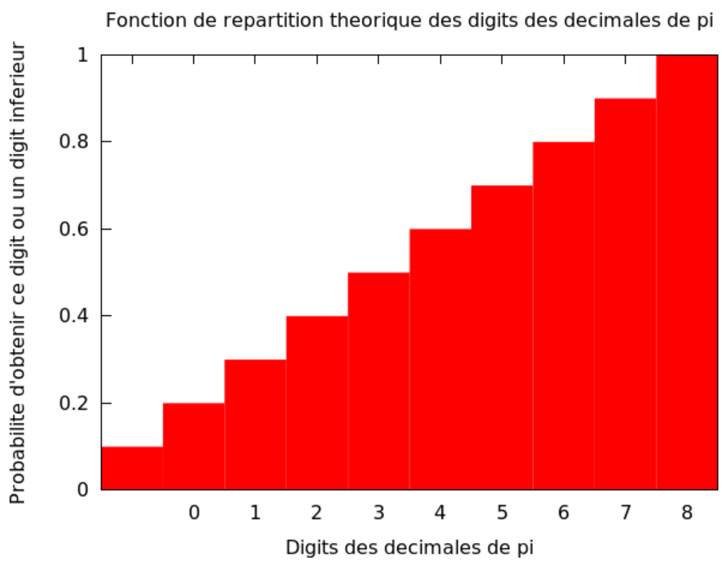
Kn=5.50908

α	$\chi^2_{9,\ 1-\alpha}$	Suit la loi uniforme ($K_n \leq \chi^2_{9,1-\alpha}$)
0.001	27.8772	True
0.05	16.919	True
0.1	14.6837	True
0,25	11,389	True

Le test du χ^2 est donc concluant.

2.2 Test de Kolmogorov-Smirnov

Ce test n'aurait pas de sens pour une analyse des décimales de π puisque la fonction de répartition théorique doit être continue pour ce test. Or nous pouvons voir que dans ce cas ci on obtient une fonction de répartition non-continue(en escalier) et par conséquent une loi discrète. Nous n'avons donc pas procédé à ce test.



2.3 Test du Gap

Ce test consiste a ce donner un intervalle [a,b] avec $0 \leq a < b \leq 1$ et marquer tous les nombres trouvés dans cet intervalle. Ensuite il faut compter la longueur du gap, c'est à dire combien de nombre ce trouve entre deux nombres marqué successif.

Ce test utilisant un test du χ^2 il nous faut des classes. Une classe sera l'ensemble des gap de cette longueur. Par exemple, pour qu'un nombre soit dans la classe des gap de longueur minimale 2, il faudra qu'il fasse partie d'un gap 2 donc au **minimum** un nombre le sépare d'un autre nombre marqué dans [a,b].
Dans notre cas des décimales de π, nous dirons que [a,b] = 1 digit. La probabilité théorique l_r d'obtenir un gap de longueur r est de :

$$l_r = (\frac{1}{10}).(\frac{9}{10})^{(r-1)}$$

, car nous devons avoir le digit concerné ($\frac{1}{10}$), suivi de $r-1$ fois un autre digit ($\frac{9}{10}$).

2.3.1 Resultat

Chiffre 0:
K_n = 30,9518
Plus grand gap = 114

alpha	χ^2	Reussi?
0,001	99.608	Oui
0,05	79,082	Oui
0.1	74.397	Oui
0.25	66,981	Oui

Chiffre 1:
K_n = 27,8228
Plus grand gap = 127

alpha	χ^2	Reussi?
0,001	99.608	Oui
0,05	79,082	Oui
0.1	74.397	Oui
0.25	66,981	Oui

Chiffre 2:
K_n = 16,2696
Plus grand gap = 129

alpha	χ^2	Reussi?
0,001	99.608	Oui
0,05	79,082	Oui
0.1	74.397	Oui
0.25	66,981	Oui

Chiffre 3:
K_n = 19,8573
Plus grand gap = 122

alpha	χ^2	Reussi?
0,001	99.608	Oui
0,05	79,082	Oui
0.1	74.397	Oui
0.25	66,981	Oui

Chiffre 4:
$K_n = 59.3571$
Plus grand gap = 157

alpha	χ^2	Reussi?
0,001	99.608	Oui
0,05	79,082	Oui
0.1	74.397	Oui
0.25	66,981	Oui

Chiffre 5:
$K_n = 40{,}7545$
Plus grand gap = 107

alpha	χ^2	Reussi?
0,001	99.608	Oui
0,05	79,082	Oui
0.1	74.397	Oui
0.25	66,981	Oui

Chiffre 6:
$K_n = 30{,}7810$
Plus grand gap = 103

alpha	χ^2	Reussi?
0,001	99.608	Oui
0,05	79,082	Oui
0.1	74.397	Oui
0.25	66,981	Oui

Chiffre 7:
$K_n = 21{,}1570$
Plus grand gap = 104

alpha	χ^2	Reussi?
0,001	99.608	Oui
0,05	79,082	Oui
0.1	74.397	Oui
0.25	66,981	Oui

Chiffre 8:
$K_n = 16{,}2850$
Plus grand gap = 105

alpha	χ^2	Reussi?
0,001	99.608	Oui
0,05	79,082	Oui
0.1	74.397	Oui
0.25	66,981	Oui

Chiffre 9:
$K_n = 21{,}8158$
Plus grand gap = 125

alpha	χ^2	Reussi?
0,001	99.608	Oui
0,05	79,082	Oui
0.1	74.397	Oui
0.25	66,981	Oui

On remarque que c'est le chiffre 4 qui s'éloigne le plus de la valeur théorique avec un Kn plus elevé et un gap maximal plus grand.

2.4 Test du poker

Le principe de ce test est de jouer avec "une main" de taille fixe et de compter le nombre d'élement différent. Dans notre cas nous prendrons des mains de taille 5 et nous compterons les chiffres différents.

Il nous restera ensuite à effectuer un test de χ^2 sur les chiffres que l'on aura calculé en les comparant avec le nombre théorique, calculé grâce au nombre de Stirling, de la manière suivante :

$$Soit \begin{cases} \textbf{k} \; la\ longueur\ de\ sequence\ etudiee \\ \textbf{d} \; le\ nombre\ de\ digits\ possibles \\ \textbf{r} \; la\ main\ etudiee \end{cases}$$

$$Soit\ : \left\{ \begin{matrix} k \\ r \end{matrix} \right\} = \left\{ \begin{matrix} k-1 \\ r-1 \end{matrix} \right\} + r \cdot \left\{ \begin{matrix} k-1 \\ r \end{matrix} \right\}$$

$$Et\ le\ cas\ de\ base\ : \left\{ \begin{matrix} k \\ 1 \end{matrix} \right\} = \left\{ \begin{matrix} k \\ k \end{matrix} \right\} = 1$$

$$Alors\ la\ probabilite\ P_r\ d'obtenir\ la\ configuration\ r\ est\ :$$

$$P_r = \frac{\left\{ \begin{matrix} k \\ r \end{matrix} \right\} \cdot d(d-1)...(d-r+1)}{d^k}$$

Avec ici k = 5 , d = 10 , r = 1 à 5 , sans oublier que notre population pour le test de χ^2 n'est alors plus de 1000000 mais de 200000 = $\frac{1000000}{k} = \frac{1000000}{5}$

r	Theorique	Observé
1	20	13
2	2700	2644
3	36000	36172
4	100800	100670
5	60480	60501

K_n=4.60820965608

alpha	chi^2	Réussi?
0.001	18.466	Oui
0.05	9.488	Oui
0.1	7.779	Oui
0.25	5.385	Oui

2.5 Test du collectionneur de coupon

Ce test repose sur un principe simple: on va parcourir les décimales de π jusqu'à ce qu'on ait vu touts les chiffres de 0 à 9. Une fois une telle séquence trouvée, on recommence. On va ensuite faire un comptage des tailles de ces séquences. Ce sera nos classes.

Mais pour pouvoir effectuer un test de χ^2, nous avons besoin d'une probabilité théorique d'occurrence pour ces classes: La probabilité S_r d'avoir une séquence de longueur r est :

$$Soit \begin{cases} \mathbf{d} \ le\ nombre\ de\ digits\ possibles \\ \mathbf{r} \ la\ longueur\ etudiee \end{cases}$$

$$Et \ : \ \begin{Bmatrix} k \\ r \end{Bmatrix} = \begin{Bmatrix} k-1 \\ r-1 \end{Bmatrix} + r \cdot \begin{Bmatrix} k-1 \\ r \end{Bmatrix}$$

Avec comme cas de base :

$$\begin{Bmatrix} k \\ 1 \end{Bmatrix} = \begin{Bmatrix} k \\ k \end{Bmatrix} = 1$$

Alors la probabilite S_r d'obtenir une suite de taille r est :

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

Nous avons ici d=10 car 10 chiffres dans les décimales, On s'arrête ici a un r=100, au delà de ça, on considère la 100è classe comme étant celle des coupons de 100 élements ou plus.

2.5.1 Resultat

i	experimental	theory
0	0	0.0
$\vdots$	$\vdots$	$\vdots$
9	0	0.0
10	12	12.39706944
11	62	55.78681248
12	154	143.186152032
13	265	276.144721776
14	496	445.677125782
15	645	636.605631934
16	869	832.196551932
17	1008	1017.53193425
18	1150	1181.29471772
19	1341	1316.26375399
$\vdots$	$\vdots$	$\vdots$
87	6	3.96511573605
88	3	3.5687465597
89	1	3.2119858227
90	1	2.89087837644
91	1	2.60186344817
92	4	2.34173543126
93	4	2.10760855074
94	1	1.89688502594
95	1	1.70722638771
96	1	1.53652764053
97	4	1.38289398981
98	0	1.24461988155
99	14	1.120170126

K_n = 78,101, nombre de coupon = 34163

alpha	chi^2	Réussi?
0.001	137.208	oui
0.05	113.145	oui
0.1	107.565	oui
0.25	98.650	oui

Chapter 3

Génération de nombre aléatoire

Notre générateur utilise la congruence linéaire vue au cours. La semence est prise a partir de la fonction " time.time() " permettant d'obtenir le temps en milliseconde écoulé depuis le 1 janvier 1970. La semence changera donc à chaque génération de nombre aléatoire, et la série de nombres sera donc toujours différente.

La technique consiste à utiliser la formule de la congruence linéaire avec cette semence, d'en lire 6 chiffres (afin d'obtenir un nombre entre 000000 et 999999) qui correspondra à une position d'une décimale de pi (à partir du fichier donné sur Moodle contenant 1 million de décimal de pi), en effectuant plusieurs fois cette technique, nous obtiendrons un nombre généré à partir de plusieurs décimal de pi. Pour ce test, 1 millions de nombres aléatoires ont été générés par notre générateur (que nous appellerons MyGen).

Pour en faire la comparaison avec le générateur de Python (que nous appellerons PythonGen), nous allons générer également 1 million de nombres aléatoires à l'aide de la fonction " random.random() ". A noter que ces nombres seront tous compris dans l'intervalle [0,1[.

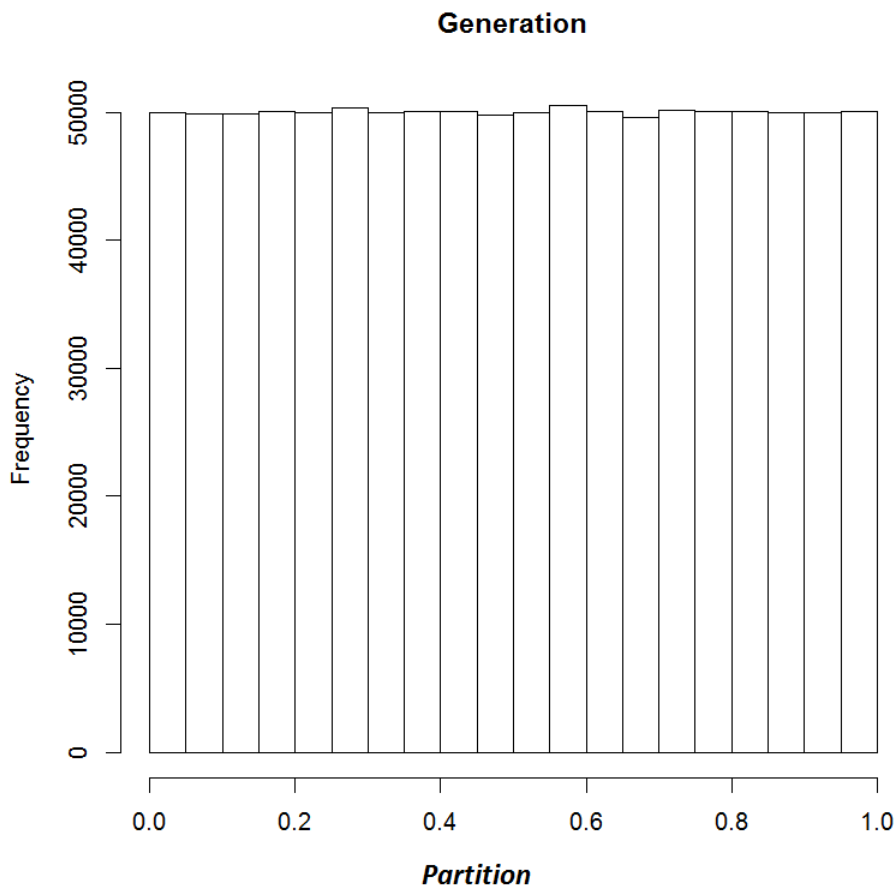
3.1 Comparaison des deux générateurs

Pour pouvoir comparer ces 2 générateurs (MyGen et PythonGen), nous allons utiliser le test de χ^2 ainsi que le test de Kolmogorov-Smirnov afin de vérifier si ils suivent bien une loi uniforme et de les comparer.

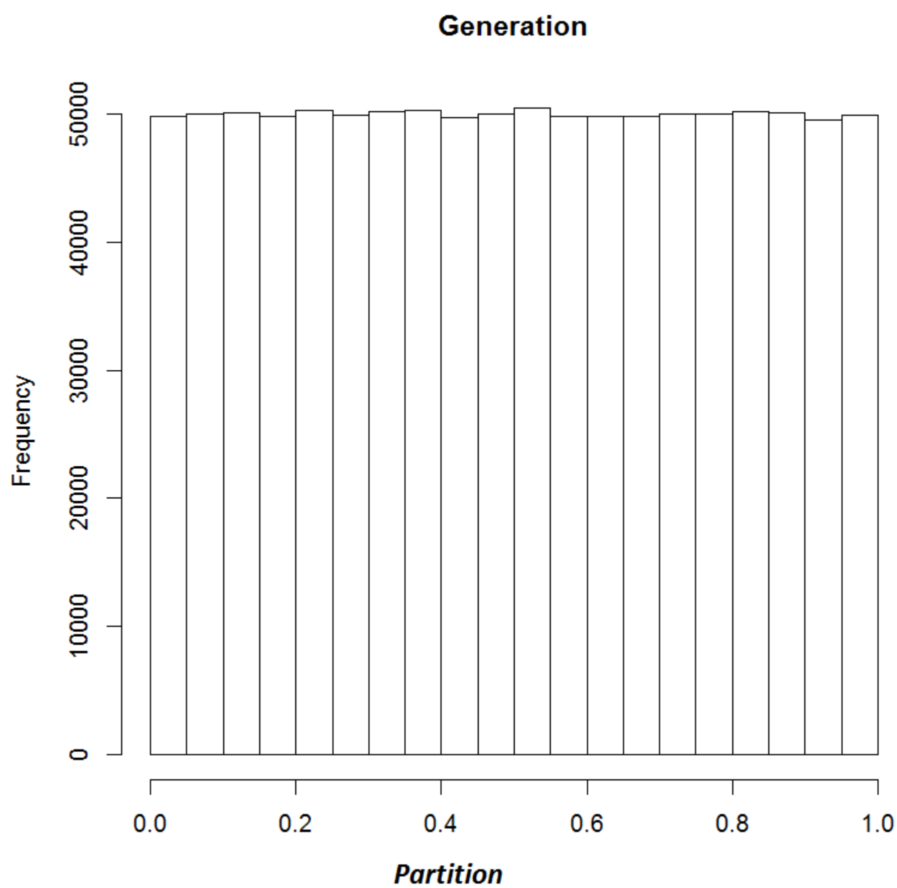
3.1.1 Histogramme

1 million de valeurs ont été générées pour chaque générateur, voici la répartition de ces nombres générés.

Histogramme MyGen:



Histogramme PythonGen:



Les nombres générés sont classé en 20 partitions avec un intervalle de 0,05 entre chacune. On peut remarquer que ces 2 histogrammes sont sensiblement identiques et que MyGen semble bien suivre la loi uniforme.

3.1.2 Test du χ^2

En utilisant la formule avec R = 20 (intervalles),
N_i = le nombre de valeur généré dans chaque intervalle,
p_i = 0.05
n_j = 1 000 000

$$K_r = \sum_{i=1}^{r} \frac{(n_i - (\sum_{j=1}^{r} n_j)p_i)^2}{(\sum_{j=1}^{r} n_j)p_i}$$

Résultat:

Alpha	Kr (MyGen)	Kr(PythonGen)	χ^2	Réussite MyGen?	Réussite PythonGen ?
0.001	14.93068	17.82628	43,8202	Oui	Oui
0.01	14.93068	17.82628	36,1909	Oui	Oui
0.1	14.93068	17.82628	27,2036	Oui	Oui
0.5	14.93068	17.82628	18,3380	Oui	Oui

Interprétation des résultats

La valeur Kr de MyGen est inférieur à celle de PythonGen -¿ MyGen génère des nombres suivant de plus près la loi uniforme.

3.1.3 Test de Kolmogorov-smirnov

Le principe est de comparer la fonction de répartition théorique F(x) à la fonction de répartition expérimentale

$$F_n(x) = \frac{nombre\ de\ val \leq x}{n}$$

Nous allons ensuite calculer

$$D_n = \sup_{X \in R} |F_n(x) - F(x)|$$

Le nombre d'échantillons sélectionné (N) sera de 1000 , 10 000, 100 000 et 1 million.
Pour que les calculs se fassent dans un temps raisonnable, la valeur de x sera comprise entre 0 et 1 par pas de 0.01.
Pour que le test soit réussi, il faut que D_n soit inférieur à D_α qui se calcule comme suit (pour N≥35) :

$n \setminus \alpha$	0.20	0.15	0.10	0.05	0.01
≥ 35	$\frac{1.07}{\sqrt{n}}$	$\frac{1.14}{\sqrt{n}}$	$\frac{1.22}{\sqrt{n}}$	$\frac{1.36}{\sqrt{n}}$	$\frac{1.63}{\sqrt{n}}$

Résultat:

N	Dn(MyGen)	Dn(PythonGen)	α	D_α	Réussite MyGen	Réussite PythonGen
1000	0.01099	0.03599	0.2	0.03384	Oui	Non
1000	0.01099	0.03599	0.15	0.03605	Oui	Oui
1000	0.01099	0.03599	0.10	0.03860	Oui	Oui
1000	0.01099	0.03599	0.05	0.04300	Oui	Oui
1000	0.01099	0.03599	0.01	0.05155	Oui	Oui
10000	0.00830	0.01089	0.2	0.01070	Oui	Non
10000	0.00830	0.01089	0.15	0.01139	Oui	Oui
10000	0.00830	0.01089	0.10	0.01219	Oui	Oui
10000	0.00830	0.01089	0.05	0.01360	Oui	Oui
10000	0.00830	0.01089	0.01	0.01630	Oui	Oui
100000	0.00242	0.00178	0.2	0.00338	Oui	Oui
100000	0.00242	0.00178	0.15	0.00360	Oui	Oui
100000	0.00242	0.00178	0.10	0.00386	Oui	Oui
100000	0.00242	0.00178	0.05	0.00430	Oui	Oui
100000	0.00242	0.00178	0.01	0.00515	Oui	Oui
1000000	0.00028	0.00074	0.2	0.00107	Oui	Oui
1000000	0.00028	0.00074	0.15	0.00114	Oui	Oui
1000000	0.00028	0.00074	0.10	0.00122	Oui	Oui
1000000	0.00028	0.00074	0.05	0.00136	Oui	Oui
1000000	0.00028	0.00074	0.01	0.00163	Oui	Oui

Interprétation des résultats

Le test fonctionne pour tous les échantillons sélectionnés pour MyGen, mais il échoue 2 fois pour PythonGen. On peut en déduire que MyGen suit mieux la loi uniforme que PythonGen.

3.2 Conclusion

Il semblerait au vu des tests que notre générateur suit mieux la loi uniforme que le générateur de python, et donc qu'il est plus performant. D'autre facteur peuvent encore être pris en compte pour vérifier l'efficacité réelle de notre générateur tel que le temps de génération des nombres qui est plus élevé que celui de python.