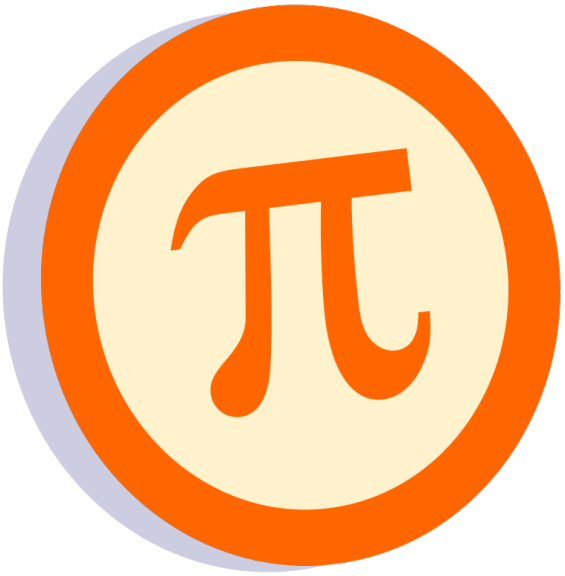

Pi randomness

Rapport de projet de simulation

Année académique 2016-2017

Auteurs :
SALEMI Marco
LECOCQ Alexis

Directeurs :
BUYS Alain
-

18 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

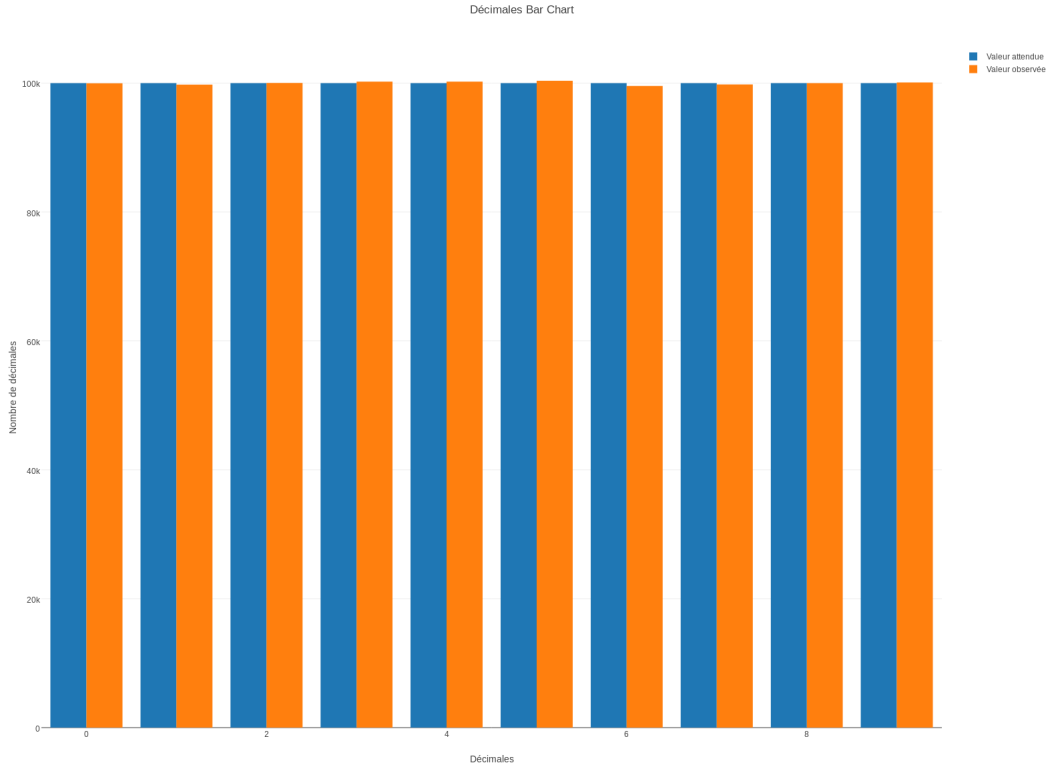

FIGURE 2 – Graphique des décimales

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.

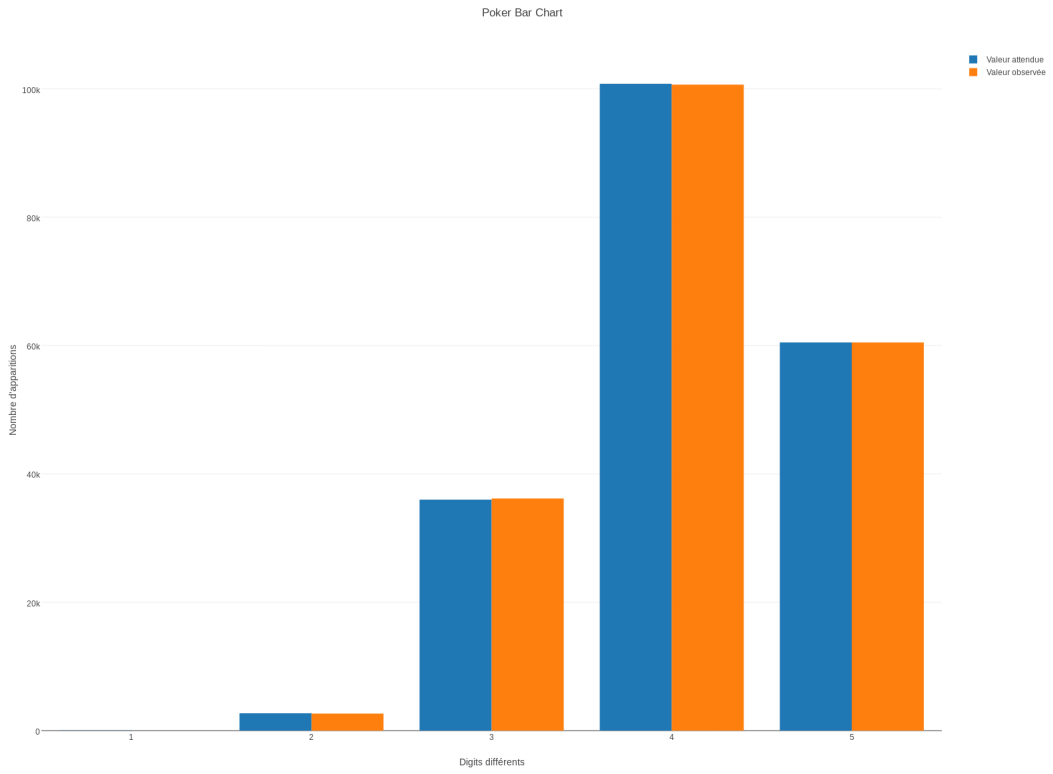

FIGURE 5 – Graphique du Poker

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

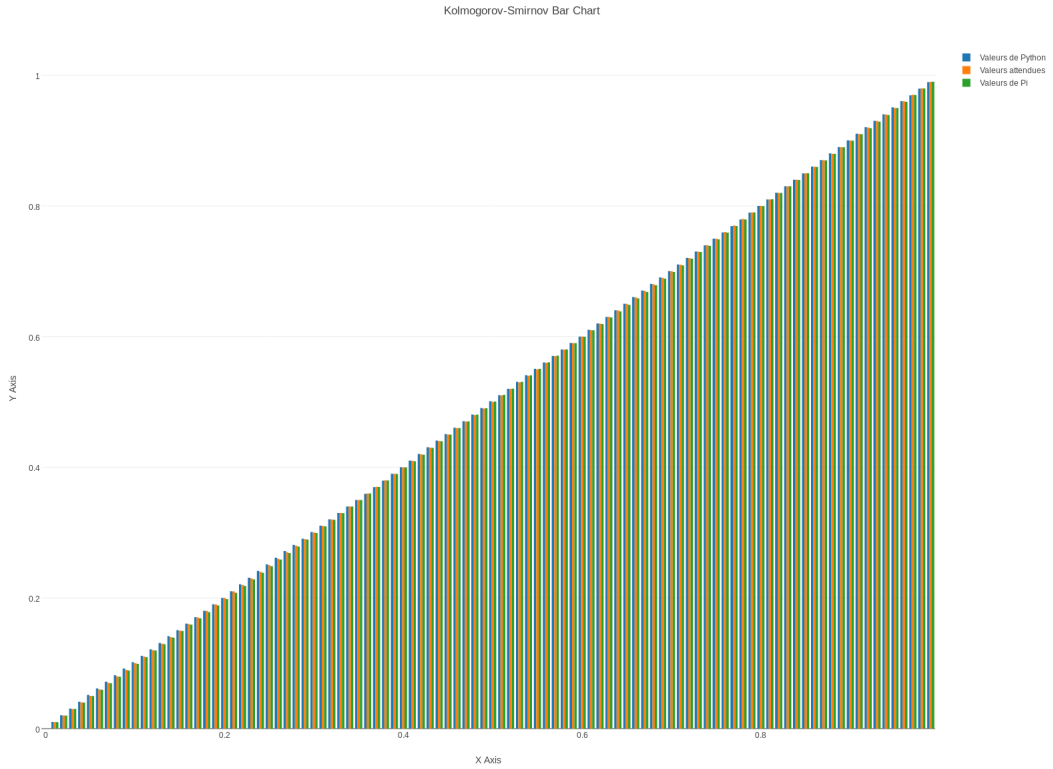

FIGURE 8 – Graphique de Kolmogorov-Smirnov

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 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 ffet, 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.