1101001011010101000101011101001101011 01 $\sin^2 x + \cos^2 x = 1$ 00101010110100101011010000111100010100001110011001 $e^{i\pi}+1=0$ 01010110100101010110100101011010101011010110101 **Tilleuil** 01010110101010101111111011010010101101010010110101010101011101 $\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}$ 10101101001010110101101001010101101001010101101001011001010 **Desbonnez** 11010101010011011010101 $f'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$ 0101101011011010100101011010110101 **InforMathématique** 0101000101101101001011011010010101 informathematiqueptijmd@gmail.com 10100101011010101010100101011010 www.informathematique.be

Citoyens, simulez ! Il y a tout à voir… (Partie 1)

Ce document a été partiellement publié par la SBPM dans la revue Losanges, n°30, 2015, pp. 50–60, sous le titre *Algorithmique et langage Ruby, le hasard est probablement bien organisé !*, J.-M. Desbonnez.

Il est ici remis au goût du jour avec le langage *Python*, sans doute plus connu, mais tout aussi simple d'utilisation pour les expériences que nous allons mener.

Les probabilités sont des notions assez abstraites pour un non-initié. Lorsqu'on lance une pièce de monnaie, on comprend facilement que l'on a une chance sur deux d'amener *pile*. On accepte alors assez facilement que la **probabilité** d'obtenir *pile* se traduit par $\frac{1}{2}$. Mais, en pratique, lorsqu'on lance dix fois une pièce et que *pile* est amené neuf fois, on se pose des questions …

La rapidité de calcul d'un ordinateur va nous rendre de grands services : un *humain* peut-il envisager de lancer une pièce dix millions de fois ? Testons quelques manipulations de pièces, de dés, de puces, et autres.

1. Pourquoi simuler ?

L'un des résultats fondamentaux de la théorie des probabilités est celui que nous connaissons aujourd'hui sous le nom de « Loi des grands nombres ». Établi dès le début du XVIII^e siècle par Jacques Bernoulli *dans son Ars Conjectandi* [1]*, il rend légitime la conception de la probabilité d'un événement aléatoire comme étant la limite de la fréquence relative de cet événement lorsque l'expérience aléatoire considérée est répétée indéfiniment.*

On conçoit sans peine que sans un tel résultat, la théorie des probabilités ne pourrait guère être appliquée qu'à des situations — souvent artificielles, ou construites de toutes pièces par l'homme — où les probabilités sont calculées a priori, généralement sur la base de propriétés de symétrie des objets manipulés (dés, pièces de monnaie, boules que l'on extrait d'une urne). Si de telles situations ne sont pas nécessairement sans intérêt, elles ne couvrent cependant qu'une petite partie des applications actuelles. Que l'on songe par exemple à l'usage des probabilités dans le secteur des assurances ou en physique quantique.

En permettant le calcul — même approché — d'une probabilité à posteriori, c'est-à-dire après expérimentation et au vu de résultats statistiques, la loi des grands nombres permet l'utilisation effective de modèles probabilistes pour étudier des situations matérielles. Elle permet aussi de

(1). Sous la forme dite « faible » de la convergence en probabilité.

juger de l'adéquation de ces modèles à des cas particuliers. Elle constitue ainsi une des composantes essentielles de l'application de la méthode scientifique tant à l'étude du monde qui nous entoure qu'à des problèmes internes aux mathématiques.

On comprendrait donc difficilement qu'une rencontre avec la loi des grands nombres soit absente des activités probabilistes proposées dans les classes de la fin de l'enseignement secondaire. Les élèves ne disposant pas des bases théoriques nécessaires, cette rencontre ne peut être qu'informelle. Il s'agit de leur faire comprendre, par exemple, que, hormis des cas triviaux, la seule possibilité de vérifier qu'une pièce de monnaie est « bien équilibrée » est d'effectuer une longue série de jets de cette pièce et d'examiner l'évolution de la fréquence relative des deux événements « Pile » et « Face ».

Il n'y a pas de doute pour nous que des activités matérielles de ce type (jets de pièces, de dés, utilisation de matériel varié) sont indispensables en vue d'une bonne conceptualisation des probabilités. Tant qu'elles sont d'ampleur limitée, elles ne poseront pas de trop gros problèmes d'organisation : dans une classe de vingt élèves, on peut en quelques minutes réaliser cinq cents jets d'un dé en répartissant les élèves en équipes de deux, l'un lançant le dé, l'autre notant les résultats et chaque équipe effectuant cinquante lancers. (On aura pris la précaution de se munir de récipients fermés pour éviter que les dés ne roulent aux quatre coins de la salle . . .)

Mais il apparaîtra rapidement que si cinq cents lancers d'un dé permettent de vérifier que la fréquence relative d'apparition d'une face déterminée tend à se stabiliser, plus de lancers sont encore nécessaires pour pouvoir affirmer que la fréquence limite est raisonnablement proche de $\frac{1}{6}$*. Vu les difficultés de réalisation pratique d'expériences comportant éventuellement plusieurs milliers d'épreuves (dans le cas considéré ici, des jets de dés), l'enseignant est contraint d'utiliser des méthodes variées pour disposer de suffisamment de données significatives.*

Plusieurs possibilités existent pour se procurer les résultats de très longues séries d'expériences. Par exemple, la Loterie Nationale publie les résultats des tirages du Lotto depuis sa création. Il est ainsi possible de connaître les fréquences relatives d'apparition de chacune des boules sur plusieurs milliers de tirages. Les taux de natalité, repris dans le Bulletin de l'Institut National de Statistiques permettent de tester l'hypothèse d'équiprobabilité de naissance d'une fille ou d'un garçon.

Nous nous intéresserons plutôt dans ce qui suit à la simulation d'expériences aléatoires par ordinateur. Ce procédé est très souple et s'adapte à un très grand nombre de situations. Il permet aux élèves d'élaborer eux-mêmes, et donc de maîtriser, le modèle qui doit leur apporter les données d'observation dont ils ont besoin.

La puissance et la rapidité des ordinateurs actuels permettent de simuler en quelques secondes plusieurs milliers de réalisations d'une expérience simple. Les logiciels utilisés peuvent traiter immédiatement les données brutes. La simulation peut elle-même être répétée à plusieurs reprises au cours de la même période de cours.

De plus, la réalisation d'une simulation fait intervenir des compétences très variées : sur le plan théorique, l'élève d'avoir bien compris les principes probabilistes sous-jacents ; sur le plan pratique, il doit se débrouiller avec la « logique de l'ordinateur ». Il doit enfin analyser les résultats obtenus et en tirer les conclusions. Le tout constitue une démarche non triviale, créatrice et porteuse de sens.

Les principales situations traitées dans la suite seront accompagnées d'une description de leur simulation, et d'exemples de résultats. Mais d'abord intéressons-nous aux outils dont nous disposons pour réaliser des simulations.

Guy Noël, 2015

2. Choisir un outil de simulation

Il s'agit de fournir au professeur et à ses élèves, un environnement qu'ils peuvent, sans trop grande difficulté, programmer eux-mêmes, réalisant ainsi des simulations adaptées aux développements choisis par l'enseignant.

Le tableur [(2)] est incontestablement l'ami du professeur de mathématique, et permet de réaliser toutes les expériences qui suivent. Nous proposons une autre option, le langage de programmation *Python*. C'est un langage moderne, gratuit, et adopté par une très grande communauté d'utilisateurs. Il peut servir de base à un apprentissage de l'algorithmique, et nos besoins dans ce cadre-ci ne sont pas compliqués.

Il ne faut surtout pas croire que l'apprentissage des manipulations de base, que ce soit d'un tableur ou d'un langage de programmation comme *Python*, sont une perte de temps dans le cadre d'un cours de mathématique, bien au contraire !

Avant de commencer, voici l'immense liste d'instructions *Python* permettant d'engendrer un nombre pseudo-aléatoire, nécessaires pour les expérimentations qui suivront.
Tout d'abord, il faut importer la bibliothèque qui contient ces instructions : `import random`
Ensuite :

`random.randint(a, b)`	un entier dans $[a, b]$ (lancer de dé(s))
`random.random()`	un flottant dans $[0.0, 1.0]$ (lancer d'une pièce)
`random.uniform(a, b)`	un décimal dans $[a, b]$ (pour le dernier exemple)
`random.seed(a)`	fixe a comme graine (juste pour l'exemple qui suit)

3. Prends-en de la graine

Un ordinateur ne sait pas créer de pur hasard, il utilise un algorithme pseudo-aléatoire, une formule mathématique plus ou moins complexe. À une valeur initiale (appelée la *graine*, ou *seed* en anglais), on applique une suite de calculs, qui produit un résultat, puis utilise ce résultat comme entrée pour le calcul suivant. La graine est le terme initial de cette réaction en chaîne.

Si la graine est **fixée**, alors tous les résultats produits seront **strictement identiques** à chaque exécution ! Fixer la graine permet de rejouer exactement la même expérience pour vérifier des résultats ou corriger un bug.

Lorsque la graine n'est pas précisée, le système utilise par défaut l'horloge interne de l'ordinateur, le nombre de millisecondes écoulées depuis 1970. Comme ce temps change « souvent », chaque exécution produit une suite différente.

Voici un exemple concret avec *Python* pour tester le comportement d'une graine ; le principe est de fixer une graine (on a choisi 12 au hasard) et de sortir 10 nombres dans l'intervalle $[1, 100]$; ensuite on change de graine (on a choisi 99, toujours au hasard) et on sort 10 nombres : ce ne sont pas les mêmes ! Puis on fixe de nouveau la graine à 12, et on constate que la liste est la même que la première.

(2). Il existe des versions gratuites et performantes.

```python
# graine.py
# test de nombres pseudo-aléatoires

import random

random.seed(12)
for k in range(10):
  print(random.randint(1, 100))

random.seed(99)
for k in range(10):
  print(random.randint(1, 100))

random.seed(12)
for k in range(10):
  print(random.randint(1, 100))
```

Première liste : 61, 35, 85, 68, 86, 45, 19, 49, 2, 48.
Deuxième liste : 52, 49, 26, 77, 23, 30, 32, 18, 98, 12.
Troisième liste : 61, 35, 85, 68, 86, 45, 19, 49, 2, 48.

4. Pile je gagne, face tu perds

On lance une pièce de monnaie parfaitement équilibrée. Calculer la probabilité d'amener pile*, d'amener* face

Quand j'étais en terminale (il y a bien longtemps déjà, soupir...), un élève lançait une pièce, un autre notait le résultat au tableau ? pendant 2 heures [3] ? (100 minutes en fait, puisque dans l'enseignement secondaire belge, 1 heure vaut 50 minutes) [4] ; heureusement, pour faire participer tous les élèves, le couple était remplacé régulièrement, sans doute pour s'assurer de l'indépendance des expériences ?

Sérieusement maintenant, avec *Python*.

```python
# pile-face.py

import random

nblancers = 1000000000
pile = 0 # compteur de pile
face = 0 # compteur de face
for k in range(1, nblancers + 1):
  if random.random() < 0.5 :
    pile += 1
  else :
    face += 1
freqpile = pile / nblancers
freqface = face / nblancers
print("fréquence pile : ", freqpile)
print("fréquence face : ", freqface)
```

(3). C'est la vérité !
(4). Avec ce type de raisonnement, les 62 jours de vacances d'été sont réduits à 51,6 jours, soupir aussi.

Les résultats affichés pour 1 milliard de lancers :
fréquence pile : 0.499983989
fréquence pile : 0.50001601

Remarque : l'instruction `range(a, b)` produit `a, a+1, a+2, ..., b-1`.

Et si la pièce est truquée ?

Supposons que la pièce a deux fois plus de chance d'amener *pile* que *face*. Dans ce cas, il suffit de diviser l'intervalle $[0, 1]$ en trois parts égales et d'en attribuer deux à *pile*. Le test dans l'algorithme devient alors

```
  if random.random() < 2/3 :
    pile += 1
  else :
    face += 1
```

Les résultats affichés pour 1 milliard de lancers :
fréquence pile : 0.666637818
fréquence pile : 0.333362181

5. Un dé, fréquence de chaque face

On lance plusieurs fois un dé (équilibré) ; calculer la fréquence d'apparition de chaque face.

Il nous faudra 6 compteurs (un par face) ; le plus simple est d'utiliser une variable de type tableau. Il faudra importer la bibliothèque qui gère les tableaux : `import numpy as np`.

Un tableau est en fait une matrice (ici une matrice ligne) dont les éléments sont numérotés, à partir de zéro, et doit être initialisé avant sa première utilisation. Le plus simple sera d'utiliser les éléments numérotés 1 à 6, ce qui nous vaudra d'initialiser un tableau à 7 éléments. On devra aussi préciser qu'il contiendra des entiers, car par défaut ce sont des flottants.

```
# un-de.py

import random
import numpy as np

nblancers = 1000000000
compteurs = np.zeros(7, dtype = int)
for k in range(1, nblancers+1) :
  jet = random.randint(1,6)
  compteurs[jet] += 1

for k in range(1, 7) :
  print("fréquence de la face", k, ":", compteurs[k]/nblancers)
```

Résultats pour 1 milliard de lancers :
fréquence de la face 1 : 0.166657779
fréquence de la face 2 : 0.166653835
fréquence de la face 3 : 0.166663012
fréquence de la face 4 : 0.166682331

fréquence de la face 5 : 0.166682414
fréquence de la face 6 : 0.166660628

soit des valeurs relativement proches de $\frac{1}{6}$.

6. Deux dés, entiers consécutifs

On lance simultanément 2 dés ; calculer la probabilité que les 2 faces amenées soient des entiers consécutifs.

Il faudra ici tester si la valeur absolue de la différence des deux valeurs est égale à 1.

```
# deux-des.py

import random

nblancers = 1000000000
compteur = 0
for k in range(1, nblancers + 1) :
  jet1 = random.randint(1, 6)
  jet2 = random.randint(1, 6)
  if abs(jet1 - jet2) == 1 :
    compteur += 1

print(compteur/nblancers)
```

Résultat pour 1 milliard de lancers : 0.277760783

La réponse exacte est $\frac{10}{36} = 0.277777\ldots$ que l'on trouve facilement sur un tableau à 2 entrées.

7. Deux dés, somme des valeurs amenées

On lance simultanément 2 dés et on fait la somme des valeurs amenées ; calculer la probabilité de chacun des résultats possibles.

Comme pour les fréquences de chacune des faces d'un dé, on utilisera un tableau de compteurs (de sommes) ; les sommes possibles sont 2 $(1+1)$, 3, …, 12$(6+6)$; par facilité, nous utiliserons les indices 2 à 12, soit un tableau de dimension 13.

```python
# deux-des-somme.py

import random
import numpy as np

nblancers = 1000000000
compteurs = np.zeros(13, dtype = int)
for k in range(1, nblancers + 1) :
  de1 = random.randint(1, 6)
  de2 = random.randint(1, 6)
  compteurs[de1 + de2] += 1

for k in range(2, 13) :
  print("somme = ", k, ":", compteurs[k] / nblancers)
```

Résultats pour 1 milliard de lancers (entre parenthèses, les résultats exacts que l'on obtient facilement sur un tableau à 2 entées) :

somme = 2 : 0.027778075 ($\frac{1}{36}$)

somme = 3 : 0.055556812 ($\frac{2}{36}$)

somme = 4 : 0.083330916 ($\frac{3}{36}$)

somme = 5 : 0.111096966 ($\frac{4}{36}$)

somme = 6 : 0.138890501 ($\frac{5}{36}$)

somme = 7 : 0.166677055 ($\frac{6}{36}$)

somme = 8 : 0.138896802 ($\frac{5}{36}$)

somme = 9 : 0.111114436 ($\frac{4}{36}$)

somme = 10 : 0.08331813 ($\frac{3}{36}$)

somme = 11 : 0.055557292 ($\frac{2}{36}$)

somme = 12 : 0.027783015 ($\frac{1}{36}$)

À suivre...