Le hasard des ordinateurs

Didier Müller, août 2018

www.apprendre-en-ligne.net

Table des matières

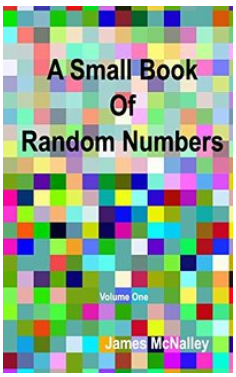
1. Introduction

1.1. Quelques mots sur le hasard

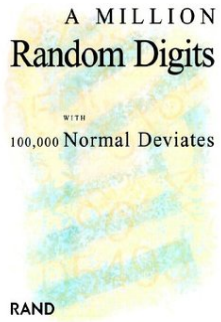
L'imitation du hasard trouve de nombreuses applications en science. C'est le cas de la simulation, des algorithmes probabilistes, de l'analyse numérique (méthode de Monte Carlo), des jeux, du traitement du signal, de la cryptographie, etc.. Et c'est pourquoi mathématiciens et informaticiens ont depuis longtemps unis leurs forces pour résoudre le problème de créer un « **générateur de nombres aléatoires** » fiable, c'est-à-dire totalement imprévisible !

C'est moins facile que l'on pourrait le croire, car un ordinateur n'improvise pas : il ne sait que suivre son programme. Il lui est donc difficile de produire, à partir d'une procédure purement mathématique, des chiffres réellement aléatoires de façon totalement imprévisible. Si on connaît la procédure mathématique, on peut obtenir la même suite de nombres. Ce n'est donc pas du hasard. Il faut se contenter de produire des séquences de nombres qui ont toutes les apparences du hasard. Dans la pratique, on se contente habituellement de nombres « pseudo-aléatoires » générés à partir d'une variable difficile à reproduire.

Utilisation de tables de nombres au hasard



On peut aussi créer de longues tables de nombres aléatoires avec de bons vieux dés : pour obtenir des listes de chiffres de 0 à 9, on prendra comme dé un dodécaèdre régulier dont les faces sont numérotées de 0 à 11 (on ne tiendra pas compte du 10 et du 11). On peut aussi utiliser d'autres moyens mécaniques.



Vivement la traduction en français !

EXTRAITS D'UNE TABLE DE NOMBRES AU HASARD

(Kendall et Babington Smith, table tirée de Christian Labrousse, Statistique, Tome2, Dunod, Paris, 1962)

02 22 85 19 48 74 55 24 89 69 15 53 00 20 88 48 95 08
85 76 34 51 40 44 62 93 65 99 72 64 09 34 01 13 09 74
00 88 96 79 38 24 77 00 70 91 47 43 43 82 71 67 49 90
64 29 81 85 50 47 36 50 91 19 09 15 98 75 60 58 33 15
94 03 80 04 21 49 54 91 77 85 00 45 68 23 12 94 23 44
42 28 52 73 06 41 37 47 47 31 52 99 89 82 22 81 86 55
09 27 52 72 49 11 30 93 33 29 54 17 54 48 47 42 04 79
54 68 64 07 85 32 05 96 54 79 57 43 96 97 30 72 12 19
25 04 92 29 71 11 64 10 42 23 23 67 01 19 20 58 35 93
28 58 32 91 95 28 42 36 98 59 66 32 15 51 46 63 57 10
64 35 04 62 24 87 44 85 45 68 41 66 19 17 13 09 63 37
61 05 55 88 25 01 15 77 12 90 69 34 36 93 52 39 36 23
98 93 18 93 86 98 99 04 75 28 30 05 12 09 57 35 90 15
61 89 35 47 16 32 20 16 78 52 82 37 26 33 67 42 11 93
94 40 82 18 06 61 54 67 03 66 76 82 90 31 71 90 39 27
54 38 58 65 27 70 93 57 59 00 63 56 18 79 85 52 21 03
63 70 89 23 76 46 97 70 00 62 15 35 97 42 47 54 60 60
61 58 65 62 81 29 69 71 95 53 53 69 20 95 66 60 50 70
51 68 98 15 05 64 43 32 74 07 44 63 52 38 67 59 56 69
59 25 41 48 64 79 62 26 87 86 94 30 43 54 26 98 61 38
85 00 02 24 67 85 88 10 34 01 54 53 23 77 33 11 19 68
01 46 87 56 19 19 19 43 70 25 24 29 48 22 44 81 35 40
42 41 25 10 87 27 77 28 05 90 73 03 95 46 88 82 25 02
03 57 14 03 17 80 47 85 94 49 89 55 10 37 19 50 20 37
18 95 93 40 45 43 04 56 17 03 34 54 83 91 69 02 90 72

Comment utiliser une telle table ?

Selon les auteurs, les méthodes différent. Toutefois la règle générale est la suivante :
On choisit, au hasard, un point d'entrée dans la table, puis on choisit un sens de parcours de la table pour prélever les chiffres et on respecte ce sens de parcours. Le sens de parcours peut être :

- soit à partir du point d'entrée, lire les nombres de la gauche vers la droite et du haut vers le bas ;
- soit à partir du point d'entrée, lire les nombres vers le haut et de droite à gauche ;
- soit à partir du point d'entrée, lire les nombres en diagonale, vers le bas et de gauche à droite...

Exemple

On veut tirer 7 nombres au hasard entre 100 et 600.

Méthode 1
On prend la ligne 5 (par exemple) et on regroupe le chiffres 3 par 3. On ne garde que ceux compris entre 100 et 600.

94 03 80 04 21 49 54 91 77 85 00 45 68 23 12 94 23 44

donnera

~~940~~ 380 ~~042~~ 149 549 177 850 ~~045~~ 682 312.

Méthode 2
On prend la ligne 7 (par exemple) et on regroupe le chiffres 4 par 4 (la table s'y prête mieux que 3 par 3). On ne garde que les 3 premiers chiffres, s'ils forment un nombre entre 100 et 600.

09 27 52 72 49 11 30 93 33 29 54 17 54 48 47 42 04 79

donnera

0927 5272 4911 3093 3329 5417 5448 4742 0479

On ne gardera que les nombres 527, 491, 309, 332, 541, 544, 474.

Remarque : On aurait aussi pu utiliser des colonnes ou des diagonales. Cela doit être choisi avant de regarder la table, de manière à ne pas être influencé.

Exercice 1

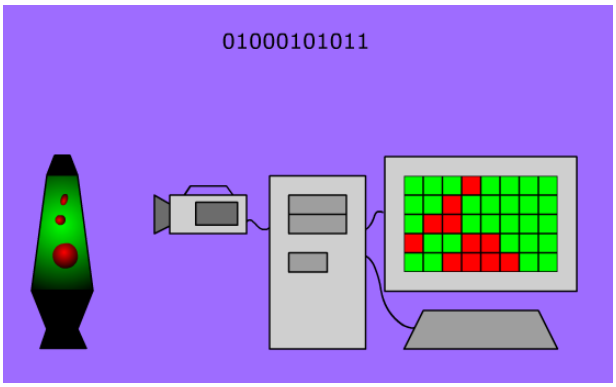
Utilisez la table de nombre aléatoire ci-dessus pour générer 5 tirages du Swiss Loto. Il s'agit de tirer 6 numéros entre 1 et 42.
Que pensez-vous des résultats ?

Utilisation de l'horloge système

Lorsque l'on a besoin de quelques bits aléatoires, on peut utiliser le bit le moins significatif de n'importe quel registre d'horloge.
Il faut néanmoins éviter d'obtenir trop de bits de cette façon : utiliser la même routine plusieurs fois de suite peut facilement biaiser le résultats; Par exemple, si chaque exécution de la routine de génération utilise un nombre pair de clics d'horloge, on obtient une suite infinie de bits identiques.

Utilisation de lampes à lave

On peut aussi, comme le faisait le laboratoire *LavaRnd*, filmer les mouvements d'une matière visqueuse dans un liquide. On appelle ces appareils des lampes à lave (*lava lamp* en anglais). Les motifs obtenus sont complètement imprévisibles et en digitalisant les images obtenues, puis en les convertissant en une série de 0 et de 1 (0 = lave, 1 = liquide), on obtient une séquence aléatoire.



Fabriquer une lampe à lave

Recette :
Migros Magazine – Alain Portner
Publié dans l'édition MM 40
28 septembre 2015

Fabriquer une lampe à lave des années 70 sans sortir de sa cuisine? Un véritable jeu d'enfant!

1. Un récipient transparent (bouteille ou vase), de l'huile végétale, du vinaigre, du bicarbonate de soude (poudre à lever) et du colorant alimentaire. Voilà ce dont on a besoin pour fabriquer sa lampe à lave psychédélique.
2. Mettre deux cuillères à soupe de bicarbonate de soude dans le récipient transparent. Ajouter l'huile en la versant tout doucement de manière à ce que le bicarbonate de soude reste au fond.

3. Remplir le tiers d'un verre avec du vinaigre et ajoute six gouttes de colorant alimentaire. Mélanger le tout avec enthousiasme et verser cette préparation dans le récipient contenant l'huile et le bicarbonate de soude.

4. Des bulles de vinaigre colorées montent et descendent, Puis remontent et redescendent dans une chorégraphie éphémère qui dure quelques minutes.

Le phénomène
Dans cette expérience, on joue avec des liquides qui ne se mélangent pas entre eux (ici, le vinaigre et l'huile). Le vinaigre, parce qu'il est plus lourd que l'huile, descend au fond du récipient. Celui-ci réagit ensuite avec le bicarbonate de soude en dégageant du gaz carbonique.
Et ce CO_2 forme des petites bulles qui, en s'accrochant au vinaigre, oblige ce dernier à remonter. Une fois à la surface, le gaz s'échappe et les gouttelettes de vinaigre coulent à nouveau. Et ainsi de suite…

Utilisation du bruit thermique des circuits intégrés

Depuis les Pentium III et 4, Intel intègre à tous les chipsets de la famille i8xx (i815, i820, i840, i845, i860...) un générateur de nombres aléatoires basé sur le bruit thermique des circuits intégrés. Le principe repose sur le comportement de tout composant électrique traversé par un courant : les électrons s'agitent aléatoirement, produisant un « bruit thermique » (*thermal noise*, ou *Johnson noise* en anglais) qui fait fluctuer la tension mesurée à ses bornes.
L'intérêt de ce système repose dans son coût, et surtout dans sa vitesse d'environ 75 kbits/sec.
Ce système a été testé par le *Cryptography Research* qui déclare qu'il est de loin le plus sûr dispositif générant des données aléatoires disponibles sur un ordinateur personnel.

Exercice 2

Il n'est pas si simple d'avoir un générateur aléatoire non biaisé. On pourrait jouer à pile ou face pour générer des nombres en binaire (pile = 1 et face = 0 par exemple). Mais est-on sûr que la pièce est bien équilibrée ? Imaginons qu'avec une pièce de monnaie pile apparaisse avec la probabilité p (différente de ½) et face avec la probabilité $1–p$.
Il existe une méthode pour générer avec cette pièce de monnaie non équilibrée des 0 et des 1 avec une probabilité égale. Saurez-vous la découvrir ?

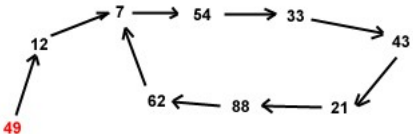
1.2. Nombres pseudo-aléatoires

Une séquence pseudo-aléatoire (*Pseudo Random Sequence* en anglais) est une suite de nombres entiers x_0, x_1, x_2, … prenant ses valeurs dans l'ensemble $M=\{0, 1, 2, ..., m-1\}$. Le terme x_n ($n>0$) est le résultat d'un calcul (à définir) sur le ou les termes précédents. Le premier terme x_0 est appelé le germe (*seed* en anglais).

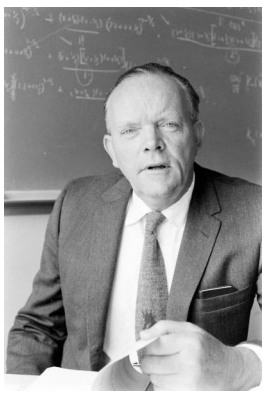
Algorithme général de récurrence

1. Choisir x_0 dans M
2. Poser $x_{n+1} = f(x_n)$, pour $n>0$

où f est une application adéquate de M dans M.



Une suite construite comme indiqué dans l'algorithme ci-dessus contient toujours un cycle de nombres se répétant à partir d'un certain endroit. Pour éviter de devoir employer plusieurs fois le même nombre au cours d'une simulation, on cherchera à rendre la longueur de ce cycle, appelée période, aussi grande que possible.
Les opérations à faire pour calculer les termes de la séquence, ainsi les différents paramètres de départ forment ce qu'on appelle un générateur de nombres pseudo-aléatoires.
Remarquons que si on ne change pas le germe, on obtiendra toujours la même séquence. Cela peut se révéler utile, en particulier en simulation. Si l'on veut avoir des séquences différentes, on peut, par exemple, prendre comme germe les décimales de l'heure exacte où le programme a été démarré ou encore sur l'intervalle de temps entre deux événements (ex. appuyer sur deux touches du clavier). Le mathématicien américain **D. H. Lehmer** donnait cette définition : « *Une suite aléatoire est une vague notion couvrant l'idée d'une suite dans laquelle chaque terme est imprévisible pour un non-initié, et dont les éléments satisfont un certain nombre de tests statistiques traditionnels, dépendant éventuellement de l'usage auquel est destinée la suite*. »



Derrick Henry **Lehmer**
(1905-1991)

Qualités requises d'un générateur

Qu'attend-on réellement d'un générateur de nombres pseudo-aléatoires ? La réponse à cette question conditionne la méthode à choisir, car dans ce domaine comme ailleurs, tout est affaire de compromis. En général on privilégie quatre critères :

- la vitesse : il faut que le calcul du nombre pseudo-aléatoire suivant soit rapide. Il n'est en effet pas rare de devoir générer des millions de nombres.
- la simplicité : une méthode très compliquée sera très difficile à programmer et surtout à tester. En général, on dit même en forme de boutade que plus l'algorithme est compliqué moins la séquence sera aléatoire.
- la méthode ne doit pas souffrir de faille grave : on en reparlera plus loin mais l'histoire des générateurs pseudo-aléatoires est pleine d'algorithmes qui se « coincent » lorsqu'ils arrivent sur un nombre particulier.
- les nombres produits ne doivent pas faire apparaître de suite logique, quelle que soit la façon de les regarder. Ce critère est le plus difficile à quantifier car il dépend fortement de l'application. Par exemple, imaginons qu'on veuille utiliser un générateur pour simuler un lancer de dés. On va vouloir absolument que tous les nombres de 1 à 6 aient exactement les mêmes chances d'apparaître. Mais si c'est pour jouer au 421, on va vouloir en plus que toutes les suites de deux et trois nombres consécutifs aient aussi exactement les mêmes chances d'arriver. Si la séquence 421 a moins de chances de se produire que 666, certains ne vont pas hésiter à prétendre que les dés sont pipés !

On dit qu'un générateur pseudo-aléatoire est acceptable s'il a passé avec succès toute une batterie de tests de statistiques généraux (voir chapitre 3).

2. Quelques générateurs classiques

2.1. Méthode du carré médian

Cette icône indique qu'il existe un page interactive sur ce sujet sur le site web compagnon : www.apprendre-en-ligne.net/random

Le premier exemple de générateur que nous verrons est le carré médian: on élève au carré un nombre de $2k$ chiffres et on prend comme terme suivant de la séquence pseudo-aléatoire le nombre formé des $2k$ chiffres du milieu du résultat (de $4k$ chiffres). Prenons un petit exemple avec $k = 1$.

Exemple

Le tableau ci-dessous montre la séquence pseudo-aléatoire obtenue en prenant comme germe $x_0 = 12$.

n	0	1	2	3	4	5	6	7	8	9	...
x_n	12	14	19	36	29	84	5	2	0	0	...
x_{n+1}	0144	0196	0361	1296	0841	7056	0025	0004	0000	0000	...

On voit apparaître un phénomène qui se produit souvent avec ce genre de générateur : aussitôt qu'un nombre de la suite vaudra 0, tous les suivants vaudront 0 aussi.
Il peut aussi arriver que l'on ne tombe pas sur 0, mais que l'on retrouve très rapidement les mêmes nombres (on dit que la période est courte).
Ces défauts sont évidemment inadmissibles, ce qui fait que ce genre de générateur est à oublier !

Exercice 3

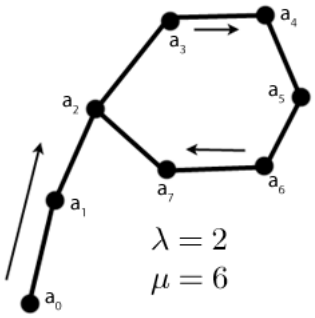
Programmez la méthode du carré médian. Les données seront le germe, la valeur k et la longueur de la séquence pseudo-aléatoire.
Quelques germes « amusants » avec $k = 1$: 15, 33, 57, 88, ...
Avec $k = 2$: 1296, 1567, 2002, ...
Testez ces germes et trouvez d'autres germes intéressants pour différentes valeurs de k.

Algorithme du lièvre et de la tortue

L'*algorithme de Floyd* de détection de cycle, encore connu sous le nom d'*algorithme du lièvre et de la tortue*, sert à détecter les cycles dans des suites de nombres pseudo-aléatoires.

Soit f une fonction. On considère la suite (a_i) définie par le germe a_0 et $a_{i+1} = f(a_i)$.
Il est clair que cette suite aboutit nécessairement à un cycle, le nombre de ses valeurs possibles étant limité à n. Une manière naïve de déterminer la longueur de ce cycle consiste à stocker chaque élément de la séquence et, à chaque étape, de vérifier si le nouvel élément fait partie des éléments déjà rencontrés.
Si l'on parvient à trouver deux éléments de la séquence a_i et a_j tels que $a_i = a_j$, alors i-j est un multiple de la longueur du cycle.
L'algorithme de Floyd de détection de cycle détermine une telle égalité en parcourant la séquence simultanément à deux vitesses différentes, respectivement 1 et 2. L'algorithme aboutit nécessairement sur deux valeurs égales, i.e. détermine m tel que $a_m = a_{2m}$.
Dès lors, $2m - m = m$ est un multiple de la longueur du cycle, μ.
Pour déterminer la valeur exacte de μ, il suffit de refaire tourner l'algorithme à partir de m+1, jusqu'à trouver un autre nombre k tel que $a_k = a_{2k}$. Dès lors on a d'une part $k \leq m+\mu$ (car on retombe alors sur a_m) et d'autre part μ qui divise $k - m$ (car il divise m et k), donc $\mu = k - m$.
La meilleure façon de visualiser l'algorithme consiste à représenter la séquence. Le

dessin obtenu ressemble à la lettre grecque ρ. La séquence démarre en bas, monte et emprunte le cycle dans le sens des aiguilles d'une montre. Suivant l'algorithme de Floyd, les deux parcours se rencontrent en a_6 après 6 itérations. Un deuxième tour de l'algorithme amène les deux parcours à se rencontrer après 6 nouvelles itérations, d'où la valeur $\mu = 6$.



Source de l'image : Wikipédia

Étape	0	1	2	3	4	5	6	7	8	9	10	11	12
Lent	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_2	a_3	a_4	a_5	a_6
Rapide	a_0	a_2	a_4	a_6	a_2	a_4	a_6	a_2	a_4	a_6	a_2	a_4	a_6

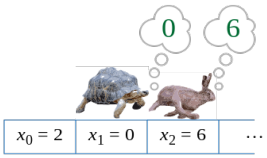
Pseudo-code

Le pseudo-code est une façon d'écrire un algorithme sans faire référence à un langage de programmation particulier.

```
tortue = f(x0)      # f(a0) est le 2ème élément de la liste
lievre = f(f(x0))
tant que tortue != lievre
   tortue = f(tortue)      # la tortue avance de 1
   lievre = f(f(lievre))   # le lievre avance de 2
```

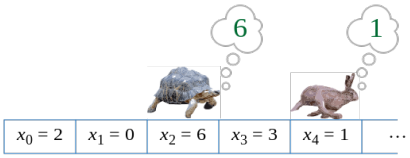
Pour obtenir la valeur de λ, la position de la première répétition, il suffit d'ajouter à la suite le code suivant :

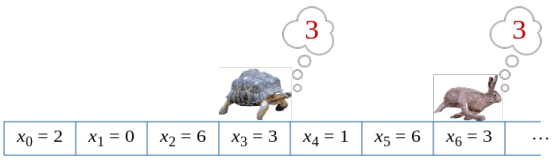
```
lam = 0
tortue = x0
tant que tortue != lievre
   tortue = f(tortue)
   lievre = f(lievre)
   lam += 1
```



Pour obtenir la valeur de μ, la longueur minimale du cycle, il suffit d'ajouter à la suite :

```
mu = 1
lievre = f(tortue)
tant que tortue != lievre
   lievre = f(lievre)
   mu += 1
```





Algorithme appliqué à la suite 2, 0, 6, 3, 1, 6, 3, 1, …

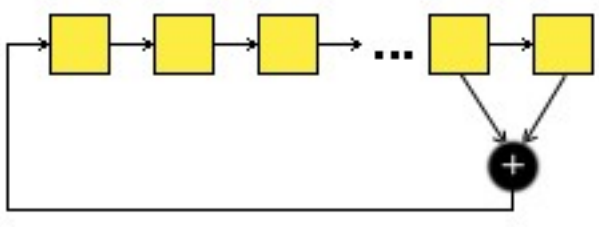
Source de l'image : Wikipédia

Exercice 4

Programmez l'algorithme du lièvre et de la tortue et appliquez-le à la suite {5, 7, 9, 12, 4, 3, 6, 13, 21, 16, 23, 19, 8, 11, 3, 6, 13, 21, 16, 23, 19, 8, 11, 3, 6, 13, 21, 16, 23, 19, 8, 11, 3, 6, 13, 21, 16, 23, 19, 8, 11, ...}

2.2. Les registres à décalage à rétroaction linéaire (LFSR)

Cette méthode pour générer des nombres pseudo-aléatoires est basée sur les registres à décalage à rétroaction linéaire (*Linear Feedback Shift Register* en anglais). L'idée est de commencer avec un registre rempli arbitrairement avec des 0 et des 1, puis de décaler cette suite d'un cran vers la droite. On remplira la position tout à gauche par la somme (modulo 2) du contenu des deux registres les plus à droite (avant le décalage), comme l'indique le schéma ci-dessous, où chaque carré jaune représente un bit et le rond noir l'addition modulo 2 :



On peut imaginer d'autres façons de remplir la position tout à gauche. Par exemple en additionnant modulo 2 les trois derniers bits, ou l'antépénultième le dernier, etc.

Exemple

Prenons un registre à 4 bits que l'on remplira pour commencer avec le motif 1111. En utilisant le schéma ci-dessus on obtiendra :

Étape	0	1	2	3	4	5	6	7
Registre	1111	0111	0011	0001	1000	0100	0010	1001
Base 10	15	7	3	1	8	4	2	9

Étape	8	9	10	11	12	13	14	15
Registre	1100	0110	1011	0101	1010	1101	1110	1111
Base 10	12	6	11	5	10	13	14	15

En convertissant les contenus successifs du registre en base 10, on obtient une suite pseudo-aléatoire. Évidemment, le processus cycle quand on retrouve le motif initial. Remarquons que l'on a obtenu tous les motifs possibles, sauf 0000, et que cela aurait été le cas pour tous les états initiaux possibles, sauf 0000. On a trouvé dans notre exemple un cycle de longueur maximale.
En général, pour un registre à n bits, il est possible d'arranger les choses pour que la longueur du cycle soit $2^n - 1$. Ainsi, pour n grand, on obtient de bons générateurs de nombres pseudo-aléatoires. Typiquement, on peut utiliser $n = 31$ ou $n = 63$.
Comme pour les générateurs congruentiels linéaires, les propriétés mathématiques de ces registres ont été beaucoup étudiées. Par exemple, on sait beaucoup de choses sur le choix des positions à utiliser pour la rétroaction pour obtenir un cycle de longueur maximale. Ainsi, pour $n = 31$, on peut faire la rétroaction avec la position 0 et au choix les positions 4, 7, 8, 14, 19, 25, 26 ou 29 (les positions sont numérotées de droite à gauche en commençant par 0).

Exercice 5

Programmez un générateur de nombres pseudo-aléatoires basé sur un registre à décalage de n bits.
Ajoutez à votre programme une fonction permettant de calculer la longueur d'un cycle.

Pour n = 7, testez avec votre programme les 21 couples de positions possibles pour effectuer la rétroaction. On supposera que le motif initial du registre est 1,1,1,1,1,1,1.

2.3. La méthode de Fibonacci

C'est la suite bien connue :

$$x_n = (x_{n-1} + x_{n-2}) \bmod m$$

Cette méthode est un cas particulier des générateurs à congruence additive. Avec un ensemble initial de nombres $x_0, x_1, x_2, \ldots, x_{k-1}$, et pour $n = k, k+1, k+2, \ldots$

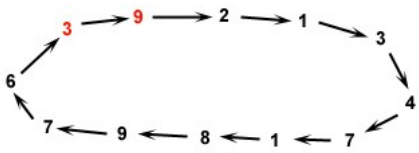
$$x_n = (x_{n-1} + x_{n-k}) \bmod m$$

Cette méthode présente deux inconvénients :

- il faut de la mémoire pour stocker les k nombres
- pour k petit, les nombres consécutifs obtenus sont fortement corrélés.

Cycles

Pour k = 1, un cycle apparaît dès qu'une paire de nombres consécutifs apparaît pour la deuxième fois : on voit dans l'exemple ci-dessous (en rouge les deux germes) qu'un même chiffre peut apparaître plusieurs fois avant l'apparition d'un cycle.



Exercice 6

Programmez la méthode de Fibonacci. Les données seront les deux germes, le modulo et la longueur de la séquence pseudo-aléatoire.

Essayez les données suivantes et commentez les résultats :

- germe 1 = 1, germe 2 = 1, modulo = 50
- germe 1 = 20, germe 2 = 30 modulo = 100
- germe 1 = 42, germe 2 = 7, modulo = 49
- germe 1 = 5, germe 2 = 15, modulo = 90
- germe 1 = 6, germe 2 = 42, modulo = 72

2.4. Générateurs à congruence linéaire (GCL)

Il s'agit de l'algorithme le plus utilisé pour produire des nombres aléatoires depuis qu'il a été inventé en 1948 par **D. H. Lehmer**. C'est la suite :

$$x_{n+1} = (a \cdot x_n + c) \bmod m$$

avec a (multiplicateur), c (incrément), x_0 (germe), et m qui sont quatre nombres entiers non-négatifs.

- Si on désire produire toujours la même séquence (ce qui est pratique à des fins de tests), on rentre toujours la même valeur du germe x_0.
- Si on préfère que la séquence soit toujours différente, on initialise x_0 avec une grandeur toujours différente, l'heure système par exemple.

Dans tous les cas, les nombres de la suite sont compris entre 0 et m–1.
La formule est simple mais le choix des trois paramètres a, c et m ne doit pas être fait à la légère, comme on le verra ci-dessous.

Critères de Knuth

Donald Erwin **Knuth**
(1938)

Knuth est connu comme l'auteur de l'ouvrage « *The Art of Computer Programming* » (une des références dans le domaine de l'informatique). Le volume 2, *Seminumerical Algorithms*, traite notamment des générateurs de nombres pseudo-aléatoires.

Si on choisit c non nul, il est toujours possible d'obtenir une période de longueur m (donc pas de risque de blocage puisqu'on va retrouver tous les nombres entre 0 et $m-1$ dans la suite). D. **Knuth** fait la démonstration des critères que doivent remplir a, c et m pour cela :

1. c et m doivent être premiers entre eux
2. $a-1$ doit être un multiple de p, pour tout p nombre premier diviseur de m
3. $a-1$ doit être un multiple de 4 si m est un multiple de 4.
4. si m est une puissance de 2, le bit de poids faible des nombres produits vaut alternativement 0 et 1 (ce n'est d'ailleurs pas le seul cas où cela se produit).

Si $c = 0$, la situation est plus compliquée parce qu'il reste toujours la possibilité de blocage (si $x_0 = 0$), donc la période du générateur ne peut être au mieux que $m-1$. L'analyse dans le cas général est assez difficile, pour $m = 2^n$ avec $n > 3$ on peut dire que :

- a mod 8 doit être égal à 3 ou 5
- la période du générateur est 2^{n-2} et tous les bits de poids faible des nombres produits sont identiques (il faut donc décaler le nombre produit de 1 vers la droite).

Soyons plus prudents et respectons scrupuleusement les critères précédents :

$$x_{n+1} = (31'415'821 x_n + 1) \bmod 100'000'000$$ (citée par R. **Sedgewick**)

Robert **Sedgewick** est un informaticien américain, surtout connu pour sa série de manuels « Algorithmes » qui présentent, expliquent et analysent les principaux algorithmes de l'informatique.
Il a fait sa thèse chez D. **Knuth**.

On a bien respecté les critères de Knuth :

1. 1 et 100'000'000 sont bien premiers entre eux. Par conséquent, on est certain qu'il n'y aura pas de problème de blocage ou de série de nombres tous pairs ou impairs. Tous les nombres entre 0 et 99'999'999 vont sortir et chacun une fois tous les 100'000'000 de tirages.
2. 31'415'821 – 1 = 31'415'820 est bien un multiple de 2 et de 5 (les deux seuls nombres premiers qui divisent 100'000'000).
3. C'est aussi un multiple de 4 (car 100 000 000 est lui-même un multiple de 4).

Regardons ce que ça donne, par exemple pour $x_0 = 0$. Les nombres produits sont:

0, 1, 31415822, 40519863, 62952524, 25482205, 90965306, 70506227, 6817368, 12779129, 29199910, 45776111, 9252132, 22780373, 20481234, 81203115, ...

Vous ne remarquez rien ? Regardez le dernier chiffre chaque nombre !
Avoir la certitude que tous les nombres vont sortir ne suffit pas à garantir qu'ils seront suffisamment aléatoires !

Quelques mauvais générateurs

Dans un article fameux, **Park** & **Miller** passent en revue de nombreux générateurs couramment utilisés et qui la plupart du temps présentent des défauts assez graves. Citons en vrac :

Park S. K., Miller K.W., *Random Number Generators: Good Ones Are Hard To Find*, Comm. of the ACM, Vol 31, 10, Oct. 1988, pp. 1192-1201

$$x_{n+1} = 65'539\, x_n \bmod 2^{31}$$
(RANDU, implanté sur les IBM System/370)

$$x_{n+1} = (129\, x_n + 907'633'385) \bmod 2^{32}$$
(le générateur de Turbo Pascal)

$x_{n+1} = (1'103'515'245\ x_n + 12'345) \bmod 2^{31}$
(le générateur d'UNIX, dont le comité ANSI C a heureusement recommandé de ne conserver que les 16 bits de poids fort. Même la documentation du Berkeley 4.2 admet que les bits de poids faible des nombres produits ne sont pas vraiment aléatoires)

Laissons la conclusion de tout ceci à Robert **Sedgewick**: « *C'est une règle générale, les générateurs de nombres aléatoires sont délicats et doivent être traités avec respect. Il est difficile d'être certain qu'un générateur donné est bon sans investir d'énormes efforts dans des tests statistiques variés...* ».

Quelques bons générateurs

Afin d'éviter que n'importe qui propose un générateur pseudo-aléatoire sans savoir très bien ce qu'il fait (comme dans les exemples dramatiques précédents), **Park** & **Miller** ont proposé un générateur standard convenablement testé et dont le code est portable sur toutes les machines. Ils l'ont appelé le *Standard minimal*. Il est défini par :

$$x_{n+1} = 16'807\ x_n \bmod (2^{31} - 1)$$

D. **Knuth** recense dans « *Seminumerical Algorithms* » un certain nombre de générateurs congruentiels linéaires de qualité. S'il n'est pas possible d'utiliser le *Standard minimal*, on pourra essayer par exemple :

$x_{n+1} = (1'664'525\ x_n + 1'013'904'223) \bmod 2^{32}$
(générateur de Knuth & Lewis)

$x_{n+1} = 69'069\ x_n \bmod 2^{32}$
(générateur de Marsaglia au multiplicande remarquable, utilisé sur le VAX)

$x_{n+1} = (31'167'285\ x_n + 1) \bmod 2^{48}$
(générateur de Lavaux & Jenssens)

$x_{n+1} = (6'364'136'223'846'793'005\ x_n + 1) \bmod 2^{64}$
(générateur de Haynes)

Exercice 7

Programmez les générateurs à congruence linéaire suivants :

- Standard minimal : $x_{n+1} = 16'807\ x_n \bmod (2^{31} - 1)$
- RANDU : $x_{n+1} = 65'539\ x_n \bmod 2^{31}$

RANDU est très impopulaire, car il possède de nombreux biais auxquels ont dû faire face les personnes qui l'ont utilisé.

3. Tests

3.1. Moyenne, écart-type et facteur d'auto-corrélation

Soit x_i, pour $i = 1, ..., n$, une suite de nombres pseudo-aléatoires obtenue d'un générateur à congruence linéaire. La suite des $u_i = x_i/m$, pour $i = 1, ..., n$ est une suite de nombres pseudo-aléatoires distribuée uniformément dans l'intervalle [0,1[.
Dans le cas idéal, on doit trouver les trois valeurs ci-dessous :

$$\bar{u} = \frac{1}{n}\sum_{i=1}^{n} u_i = \frac{1}{2} = 0.5 \quad \text{(moyenne)}$$

$$\nu = \frac{1}{n}\sum_{1}^{n} u_i^2 - \bar{u}^2 = \frac{1}{12} = 0.0833... \quad \text{(variance)}$$

$$E(u_i u_{i+1}) = \frac{1}{n}\sum_{i=1}^{n-1} u_i u_{i+1} = \frac{1}{4} = 0.25 \quad \text{(facteur d'auto-corrélation)}$$

Quelques résultats pour des suites pas du tout aléatoires

Suite (n = 10)	Moyenne	Variance	Auto-corrélation
0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5	0.5	0	0.225
0, 0, 0, 0, 0, 1, 1, 1, 1, 1	0.5	0.25	0.4
0, 1, 0, 1, 0, 1, 0, 1, 0, 1	0.5	0.25	0
0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9	0.45	0.0825	0.33

Remarque : pour simplifier, on a écrit 1 à la place de 0.9999999.

On constate que l'on n'est pas loin des trois valeurs espérées pour la dernière suite. Ce test n'est donc pas du tout suffisant...

Exercice 8

Programmez les calculs de la moyenne, la variance et le facteur d'auto-corrélation d'une liste de n nombres.
Testez les générateurs que vous avez déjà programmés, avec 10'000 nombres aléatoires.

3.2. Test du χ^2

Le test du χ^2 (prononcez khi deux ou khi carré) fournit une méthode pour déterminer la nature d'une répartition, qui peut être continue ou discrète. Nous nous occuperons ici de déterminer si une répartition est uniforme dans le cas discret.

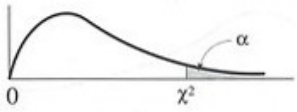
Méthode

1. On répartit les valeurs de l'échantillon (de taille n) dans k classes distinctes et on calcule les effectifs de ces classes. Il faut vérifier que pour les i de 1 à k, on a $n \cdot p_i \cdot (1-p_i) > 5$ (éventuellement répartir les valeurs autrement). Appelons o_i (i=1, ..., k) les effectifs observés et e_i les effectifs théoriques.
2. On calcule $Q = \sum_{i=1}^{k} \frac{(o_i - e_i)^2}{e_i}$.
3. La statistique Q donne une mesure de l'écart existant entre les effectifs théoriques attendus et ceux observés dans l'échantillon. En effet, plus Q sera grand, plus le désaccord sera important. La coïncidence sera parfaite si $Q = 0$.
4. On compare ensuite cette valeur Q avec une valeur $\chi^2_{k-1,\alpha}$ issue d'un tableau

(voir extrait ci-dessous) à la ligne k–1 et à la colonne α. (k–1 est le nombre de degrés de liberté et α la tolérance).

5. Si $Q > \chi^2_{k-1,\alpha}$, et si n est suffisamment grand, alors l'hypothèse d'avoir effectivement affaire à la répartition théorique voulue est à rejeter avec une probabilité d'erreur d'au plus α.

Table χ^2 : points de pourcentage supérieurs de la distribution χ^2



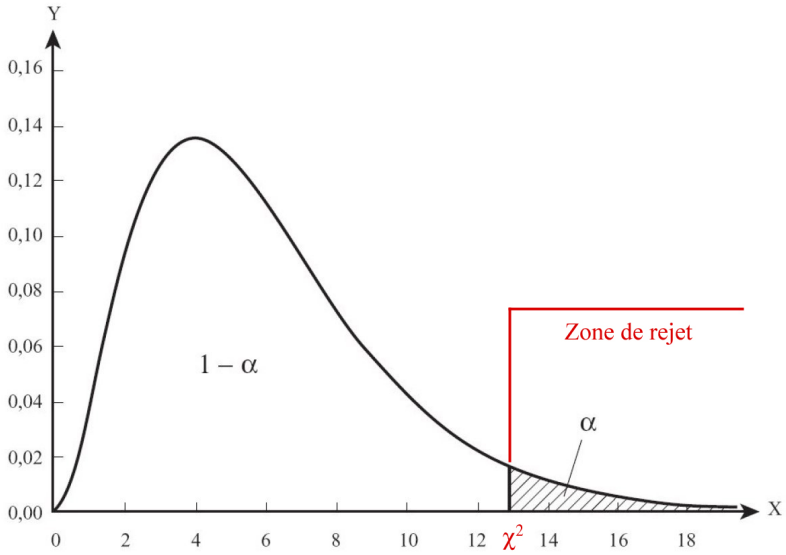
dl	.995	.990	.975	.950	.900	.750	.500	.250	.100	.050	.025	.010	.005
1	0.00	0.00	0.00	0.00	0.02	0.10	0.45	1.32	2.71	3.84	5.02	6.63	7.88
2	0.01	0.02	0.05	0.10	0.21	0.58	1.39	2.77	4.61	5.99	7.38	9.21	10.60
3	0.07	0.11	0.22	0.35	0.58	1.21	2.37	4.11	6.25	7.82	9.35	11.35	12.84
4	0.21	0.30	0.48	0.71	1.06	1.92	3.36	5.39	7.78	9.49	11.14	13.28	14.86
5	0.41	0.55	0.83	1.15	1.61	2.67	4.35	6.63	9.24	11.07	12.83	15.09	16.75
6	0.68	0.87	1.24	1.64	2.20	3.45	5.35	7.84	10.64	12.59	14.45	16.81	18.55
7	0.99	1.24	1.69	2.17	2.83	4.25	6.35	9.04	12.02	14.07	16.01	18.48	20.28
8	1.34	1.65	2.18	2.73	3.49	5.07	7.34	10.22	13.36	15.51	17.54	20.09	21.96
9	1.73	2.09	2.70	3.33	4.17	5.90	8.34	11.39	14.68	16.92	19.02	21.66	23.59
10	2.15	2.56	3.25	3.94	4.87	6.74	9.34	12.55	15.99	18.31	20.48	23.21	25.19
11	2.60	3.05	3.82	4.57	5.58	7.58	10.34	13.70	17.28	19.68	21.92	24.72	26.75
12	3.07	3.57	4.40	5.23	6.30	8.44	11.34	14.85	18.55	21.03	23.34	26.21	28.30
13	3.56	4.11	5.01	5.89	7.04	9.30	12.34	15.98	19.81	22.36	24.74	27.69	29.82
14	4.07	4.66	5.63	6.57	7.79	10.17	13.34	17.12	21.06	23.69	26.12	29.14	31.31
15	4.60	5.23	6.26	7.26	8.55	11.04	14.34	18.25	22.31	25.00	27.49	30.58	32.80
16	5.14	5.81	6.91	7.96	9.31	11.91	15.34	19.37	23.54	26.30	28.85	32.00	34.27
17	5.70	6.41	7.56	8.67	10.09	12.79	16.34	20.49	24.77	27.59	30.19	33.41	35.72
18	6.26	7.01	8.23	9.39	10.86	13.68	17.34	21.60	25.99	28.87	31.53	34.81	37.15
19	6.84	7.63	8.91	10.12	11.65	14.56	18.34	22.72	27.20	30.14	32.85	36.19	38.58
20	7.43	8.26	9.59	10.85	12.44	15.45	19.34	23.83	28.41	31.41	34.17	37.56	40.00
21	8.03	8.90	10.28	11.59	13.24	16.34	20.34	24.93	29.62	32.67	35.48	38.93	41.40
22	8.64	9.54	10.98	12.34	14.04	17.24	21.34	26.04	30.81	33.93	36.78	40.29	42.80
23	9.26	10.19	11.69	13.09	14.85	18.14	22.34	27.14	32.01	35.17	38.08	41.64	44.18
24	9.88	10.86	12.40	13.85	15.66	19.04	23.34	28.24	33.20	36.42	39.37	42.98	45.56
25	10.52	11.52	13.12	14.61	16.47	19.94	24.34	29.34	34.38	37.65	40.65	44.32	46.93
26	11.16	12.20	13.84	15.38	17.29	20.84	25.34	30.43	35.56	38.89	41.92	45.64	48.29
27	11.80	12.88	14.57	16.15	18.11	21.75	26.34	31.53	36.74	40.11	43.20	46.96	49.64
28	12.46	13.56	15.31	16.93	18.94	22.66	27.34	32.62	37.92	41.34	44.46	48.28	50.99
29	13.12	14.26	16.05	17.71	19.77	23.57	28.34	33.71	39.09	42.56	45.72	49.59	52.34
30	13.78	14.95	16.79	18.49	20.60	24.48	29.34	34.80	40.26	43.77	46.98	50.89	53.67
40	20.67	22.14	24.42	26.51	29.06	33.67	39.34	45.61	51.80	55.75	59.34	63.71	66.80
50	27.96	29.68	32.35	34.76	37.69	42.95	49.34	56.33	63.16	67.50	71.42	76.17	79.52
60	35.50	37.46	40.47	43.19	46.46	52.30	59.34	66.98	74.39	79.08	83.30	88.40	91.98
70	43.25	45.42	48.75	51.74	55.33	61.70	69.34	77.57	85.52	90.53	95.03	100.44	104.24
80	51.14	53.52	57.15	60.39	64.28	71.15	79.34	88.13	96.57	101.88	106.63	112.34	116.35
90	59.17	61.74	65.64	69.13	73.29	80.63	89.33	98.65	107.56	113.14	118.14	124.13	128.32
100	67.30	70.05	74.22	77.93	82.36	90.14	99.33	109.14	118.49	124.34	129.56	135.82	140.19

Exemple

On a lancé un dé 90 fois et on a obtenu les issues 1 à 6 (k=6) avec les effectifs suivants : 12, 16, 20, 11, 13, 18 (on a vérifié que 90 lancers sont suffisants: $n(1/6)(5/6) > 5$ implique que $n > 36$). Si le dé n'est pas pipé (notre hypothèse), on attend comme effectifs moyens théoriques 15 pour toutes les issues.

$$Q=\frac{(12-15)^2}{15}+\frac{(16-15)^2}{15}+\frac{(20-15)^2}{15}+\frac{(11-15)^2}{15}+\frac{(13-15)^2}{15}+\frac{(18-15)^2}{15}=4.266$$

Pour k–1 = 5 degrés de liberté et un seuil de tolérance de 5 %, la valeur $\chi^2_{k-1,\alpha}$ du tableau est 11.1. Cela signifie que la probabilité que Q soit supérieur à 11.1 est de 5 % (voir figure ci-dessous). Comme 4.266 < 11.1, on accepte l'hypothèse selon laquelle le dé est régulier.



Exercice 9

Testez avec le test du χ^2 les générateurs que vous avez déjà programmés :

1. en simulant 1800 lancers d'un dé à six faces
2. en simulant 1800 lancers de deux dés à six faces

3.3. Test du poker

L'idée de ce test est de comparer les fréquences théoriques des mains au poker avec les fréquences observées en simulant ces mains (une main est un ensemble de cartes). On va adapter ce test de la manière suivante.

Méthode

1. Générer 4000 nombres aléatoires entiers compris entre 0 et 9.
2. Grouper ces 4000 nombres aléatoires en 1000 listes de 4 nombres aléatoires.
3. Pour chacune des listes de 4 nombres aléatoires, compter les fréquences des cas où :
   - les chiffres sont **tous différents** (par exemple 1359)
   - la liste comporte **une paire** (par exemple 1130)
   - la liste comporte **deux paires** (par exemple 3377)
   - la liste comporte **trois chiffres identiques** (par exemple 7477)
   - la liste comporte **quatre chiffres identiques** (par exemple 2222)
4. Compléter le tableau suivant :

Cas	Effectif observé (o_i)	Effectif théorique (e_i)
les chiffres sont **tous différents**		504
la liste comporte **une paire**		432
la liste comporte **deux paires**		27
la liste comporte **trois chiffres identiques**		36
la liste comporte **quatre chiffres identiques**		1
Total	1000	1000

5. Utiliser le test du χ^2 (avec $k = 5$ car on a 5 cas observés) pour voir si les effectifs observés sont suffisamment proches des effectifs théoriques.

Exercice 10

Programmez le test du poker.
Testez avec le test du poker les générateurs à congruence linéaire que vous avez déjà programmés.

On peut généraliser le test du poker en groupant les nombres aléatoires par 5. Dans ce cas, il faut comparer les fréquences calculées aux fréquences théoriques suivantes :

Cas	Effectif théorique (e_i)
les chiffres sont **tous différents** (exemple 12098)	3024
une paire (exemple 29562)	5040
deux paires (exemple 55424)	1080
l**trois chiffres identiques** (exemple 62616)	720
trois chiffres identiques et une paire (exemple 82288)	90
quatre chiffres identiques (exemple 55551)	45
cinq chiffres identiques (exemple 22222)	1
Total	10'000

Ensuite, il faut utiliser le test du χ^2 avec $k = 7$, car on a 7 cas observés.

3.4. Test des paires en séries

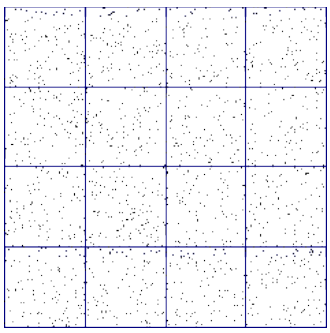
Le test des paires en série consiste à observer si les points dont les coordonnées sont obtenues en prenant deux nombres consécutifs de la séquence pseudo-aléatoire sont bien répartis dans le carré unité.

Méthode

1. Pour n nombres aléatoires compris entre 0 et 1 $\{x_1, x_2, ..., x_n\}$, on forme $\frac{n}{2}$ couples qui ne se chevauchent pas : (x_1, x_2), (x_3, x_4), ..., (x_{n-1}, x_n).
2. On les place dans un quadrillage k x k. Pour une distribution uniforme, le nombre de points par case est théoriquement $\frac{n}{2k^2}$.
3. On compare la distribution observée à la distribution théorique en utilisant le test du χ^2 avec $k^2 - 1$ degrés de liberté.

Exercice 11

En exercice, vous prendrez $k = 4$, ce qui donnera 16 cases, et vous générerez 1600 points (donc 3200 nombres) dont les coordonnées seront comprises entre 0 et 4 (voir exemple ci-contre).



Programmez le test des paires en série. Testez les générateurs que vous avez déjà programmés.

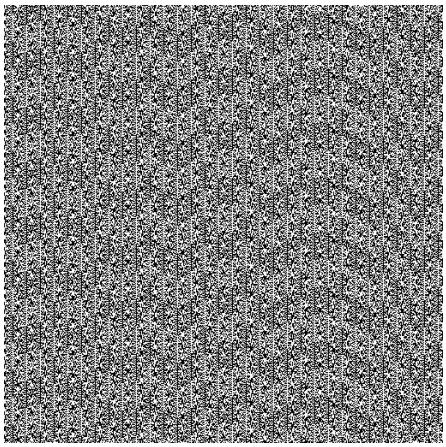
3.5. Test spectral

L'idée est de représenter visuellement la suite des nombres pseudo-aléatoires en 1 dimension (1D), 2D et 3D.

- 1D : on observe un histogramme composé de vingt bâtons (largeur 0.05).

Remarquons que ce test est insuffisant : il suffit pour s'en convaincre de penser à la séquence 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ... Les fréquences seront parfaites mais cette suite n'a rien d'aléatoire !

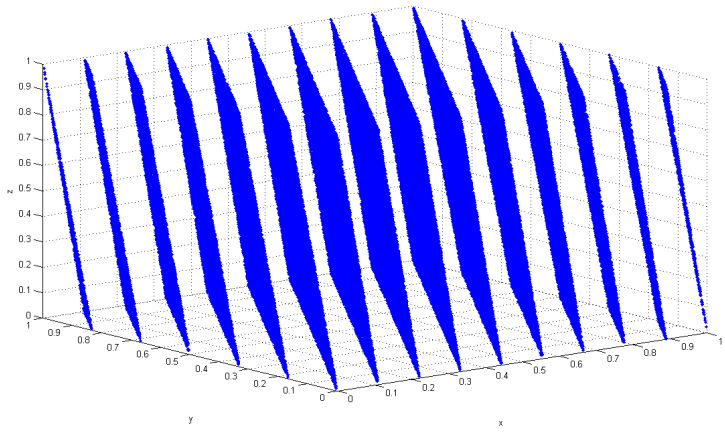
- 2D : deux valeurs consécutives seront les coordonnées d'un point du plan. On regarde si les points sont uniformément répartis dans un carré.



PHP rand() sur Microsoft Windows : il y a un souci, non ?

- 3D : trois valeurs consécutives seront les coordonnées d'un point de l'espace. on regarde si les points sont répartis uniformément dans un cube.
  En tournant le cube, on voit apparaître un effet indésirable : les plans de Marsaglia. On voit bien ci-dessous que les points sont situés sur des plans. En fait, tous les générateurs GCL souffrent de cet effet (cela est dû au fait que l'on ne génère pas tous les réels, mais seulement des fractions). Plus la distance inter-planaire est petite, meilleur est le générateur.

Voici par exemple ce que l'on obtient avec RANDU : les points de l'espace sont tous contenus dans 15 plans parallèles.



(source : Wikipédia)

Pour en savoir plus sur l'effet Marsaglia :

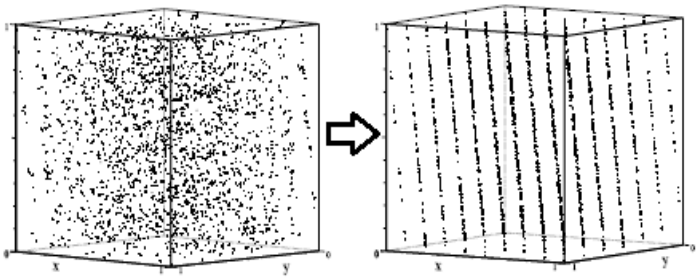
A collection of classical pseudorandom number generators with linear structures - advanced version, par Karl Entacher, http://random.mat.sbg.ac.at/results/karl/server/

Exercice 12 Programmez le test spectral 1D sous forme d'un histogramme.
Testez les générateurs que vous avez déjà programmés, avec 10'000 nombres aléatoires.

Exercice 13 Programmez le test spectral 2D.
Testez les générateurs que vous avez déjà programmés, avec 10'000 nombres aléatoires.

Exercice 14

Pour le test 3D, votre programme devra générer des points dans un cube et le faire tourner autour des 3 axes.



Générez 1000 points pseudo-aléatoires avec RANDU dans ce cube et observez les plans de Marsaglia.

3.6. Test des suites croissantes

Donald E. Knuth, **The art of Computer Programming, Vol. 2 (Seminumerical Algorithms)**, Addison-Wesley, pp. 66-69

Ce test, proposé par Donald **Knuth**, s'appelle aussi le « run up test ».

Méthode

1. Compter le nombre de sous-séquences croissantes de chaque longueur dans une séquence aléatoire de n nombres :

   r_i = nombre de sous-séquences croissantes de longueur i si $i = 1, 2, 3, 4, 5$
   r_i = nombre de sous-séquences croissantes de longueur > 5 si $i = 6$

   #### Exemple

   Séquence : 855 | 108 226 | 32 132 | 56 545 642 | 103 108 201 809 901 | 151
   1 2 2 3 5 1

   Ce qui donne: $r_1 = 2$, $r_2 = 2$, $r_3 = 1$, $r_4 = 0$, $r_5 = 1$, $r_6 = 0$

2. Calculer $R = \frac{1}{n-6} \sum_{i=1}^{6} \sum_{j=1}^{6} a_{ij} (r_i - n b_i)(r_j - n b_j)$ avec les coefficients a_{ij} et b_i ci-dessous :

a_{ij}	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$
$i = 1$	4529.4	9044.9	13568	18091	22615	27892
$i = 2$	9044.9	18097	27139	36187	45234	55789
$i = 3$	13568	27139	40721	54281	67852	83685
$i = 4$	18091	36187	54281	72414	90470	111580
$i = 5$	22615	45234	67852	90470	113262	139476
$i = 6$	27892	55789	83685	111580	139476	172860

b_i
1/6
5/24
11/120
19/720
29/5040
1/840

3. Pour une valeur de n élevée (disons plus de 4000 nombres), R suit une distribution du χ^2 avec 6 degrés de liberté (et non pas 5). Si le seuil de tolérance est de 5 %, la suite aléatoire passera donc le test si $R < 12.59$.

Exercice 15

Programmez le test des suites croissantes.
Appliquez le test des suites croissantes aux générateurs que vous avez déjà programmés.

3.7. Run test

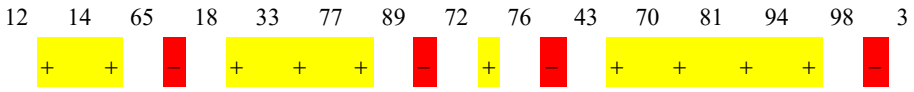
Ce test est basé sur le nombre de séquences croissantes et décroissantes. En effet, on perçoit bien intuitivement qu'une suite strictement croissante du genre 1, 5, 89, 112, 145, 167, 245, ... n'est pas une suite aléatoire.
Soit R le nombre de séquences dans l'échantillon de n nombres. Théoriquement, la moyenne et la variance sont :

$$E(R)=\frac{2n-1}{3} \text{ et } \mathrm{Var}(R)=\frac{16n-29}{90}$$

et pour n élevé (grand échantillon), la répartition de R suit approximativement une loi normale.
Soit $z=\frac{R-E(R)}{\sqrt{\mathrm{Var}(R)}}$. En fixant un seuil de tolérance de 5 %, | z | > 1.96 indique que l'échantillon n'est « pas vraiment aléatoire ».

Exemple

Prenons une suite de n = 15 nombres obtenus par un générateur aléatoire produisant des nombres entiers entre 0 et 99 :



Il y a R = 8 séquences : 4 croissantes (en jaune) et 4 décroissantes (en rouge).
Théoriquement, on devrait avoir :

$$E(R)=\frac{2\cdot 15-1}{3}=\frac{29}{3}=9.6... \text{ et } \mathrm{Var}(R)=\frac{16\cdot 15-29}{90}=2.34$$

Le calcul de z donne : $z=\frac{R-E(R)}{\sqrt{\mathrm{Var}(R)}}=\frac{8-9.66}{\sqrt{2.34}}=-1.089$.

Comme $|z| < 1.96$, le test est réussi.

Exercice 16

Programmez le run test.
Appliquez le run test aux générateurs que vous avez déjà programmés.

Exercice 17

Simulez le lancer d'une pièce de monnaie. Un compteur sera incrémenté de 1 si la pièce indique face et décrémenté de 1 si elle indique pile.
Dessinez une courbe polygonale qui montre l'état du compteur en fonction du nombre de lancers.

3.8. Test de Kolmogorov-Smirnov

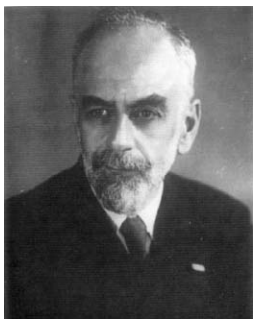
Une fonction de répartition F d'une variable aléatoire X est définie pour tout nombre réel x, par

$$F(x) = P(X < x)$$

En d'autres termes, $F(b)$ est la probabilité que la variable aléatoire X prenne une valeur inférieure à b. Pour un échantillon uniformément distribué entre 0 et 1, on a $F(x) = x$,

avec x compris entre 0 et 1.

Andreï Nikolaïevich **Kolmogorov** (1903-1987)

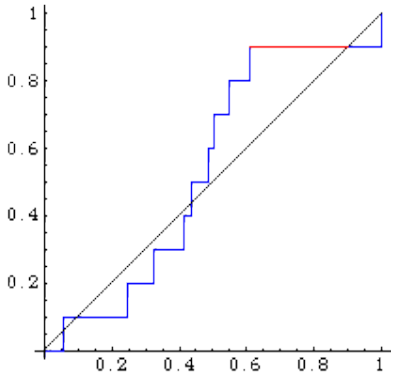


Vladimir Ivanovitch **Smirnov** (1887-1974)

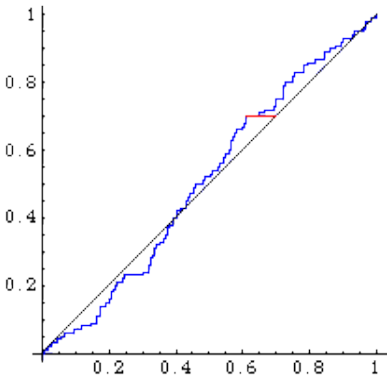
Exemple

Soit la suite ordonnée de n = 10 nombres pseudo-aléatoires : 0.0582795, 0.248849, 0.326321, 0.411576, 0.435057, 0.485013, 0.5035, 0.547168, 0.611682, 0.998397. La fonction de répartition $F_n(x)$ de cet échantillon sera en bleu dans le graphique :

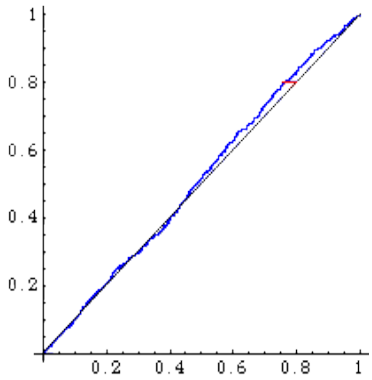
$$F_n(X) = \frac{\text{nombre de } X_i \leqslant x}{n}$$



10 nombres, D = 0.2883 < 0.410, accepté (α = 5 %)



100 nombres, D = 0.0833 < 0.136, accepté (α = 5 %)



1000 nombres, D = 0.0433 > 0.043, refusé de justesse (α = 5 %)

On remarque que quand n augmente, $F_n(x)$ se rapproche de $F(x)$. Le test de Kolmogorov-Smirnov compare la fonction de répartition uniforme avec la fonction de répartition de l'échantillon empirique. L'idée est de calculer la distance maximale (en rouge sur les graphiques) entre les fonctions théoriques et empiriques. Si cette distance dépasse une certaine valeur, qu'on lira dans le tableau ci-dessous, on dira que l'échantillon est mauvais.

n \ α	0.01	0.05	0.1	0.15	0.2
1	0.995	0.975	0.950	0.925	0.900
2	0.929	0.842	0.776	0.726	0.684
3	0.828	0.708	0.642	0.597	0.565
4	0.733	0.624	0.564	0.525	0.494
5	0.669	0.565	0.510	0.474	0.446
6	0.618	0.521	0.470	0.436	0.410
7	0.577	0.486	0.438	0.405	0.381
8	0.543	0.457	0.411	0.381	0.358
9	0.514	0.432	0.388	0.360	0.339
10	0.490	0.410	0.368	0.342	0.322
11	0.468	0.391	0.352	0.326	0.307
12	0.450	0.375	0.338	0.313	0.295
13	0.433	0.361	0.325	0.302	0.284
14	0.418	0.349	0.314	0.292	0.274
15	0.404	0.338	0.304	0.283	0.266
16	0.392	0.328	0.295	0.274	0.258
17	0.381	0.318	0.286	0.266	0.250
18	0.371	0.309	0.278	0.259	0.244
19	0.363	0.301	0.272	0.252	0.237
20	0.356	0.294	0.264	0.246	0.231
25	0.320	0.270	0.240	0.220	0.210
30	0.290	0.240	0.220	0.200	0.190
35	0.270	0.230	0.210	0.190	0.180
40	0.250	0.210	0.190	0.180	0.170
45	0.240	0.200	0.180	0.170	0.160
50	0.230	0.190	0.170	0.160	0.150
> 50	$\frac{1.63}{\sqrt{n}}$	$\frac{1.36}{\sqrt{n}}$	$\frac{1.22}{\sqrt{n}}$	$\frac{1.14}{\sqrt{n}}$	$\frac{1.07}{\sqrt{n}}$

Méthode

On procède de la façon suivante :

1. Générer n nombres aléatoires entre 0 et 1.
2. Les trier par ordre croissant : $X_1, X_2, ..., X_n$.
3. Calculer $A = \max\left(\frac{i}{n} - X_i\right)$ et $B = \max\left(X_i - \frac{i}{n}\right)$
4. Calculer $D = \max(A, B)$
5. Déterminer la valeur critique $D_\alpha(n)$ dans le tableau ci-dessous.
6. Si $D > D_\alpha(n)$ (voir tableau ci-dessus), on dit que l'échantillon est mauvais.

Exercice 18

Programmez le test de Kolmogorov-Smirnov. Prenez α = 5 %.
Appliquez le test de Kolmogorov-Smirnov aux générateurs que vous avez déjà programmés.

4. Lois de probabilité

4.1. Variable aléatoire discrète

On désigne une variable aléatoire par une lettre majuscule.
Les valeurs qu'elle prend sont écrites en minuscules (x_1, x_2, etc.)

Il arrive souvent qu'à propos d'une épreuve, on soit amené à attribuer des valeurs numériques à ses issues.

D'un point de vue formel, une **variable aléatoire** est une **fonction** $X : \Omega \to \mathbb{R}$, où Ω est l'univers des résultats. Si l'ensemble des valeurs de cette fonction est fini ou dénombrable, on dit que cette variable aléatoire est **discrète**.

Exemple

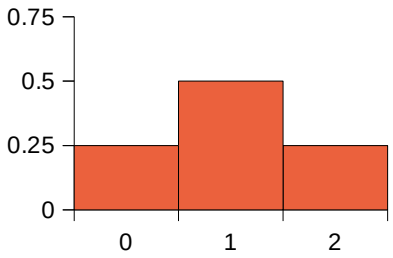
Prenons tout de suite un exemple : on lance deux pièces de monnaie et on compte le nombre de côtés pile, nombre que l'on désignera par la variable aléatoire X. Trois résultats sont possibles : 0, 1 ou 2. À chacun de ces trois résultats possibles, on **associe une probabilité** :

On remarque que la somme de des probabilités est égale à 1.

$$P\{X=0\} = P\{(F, F)\} = 1/4$$
$$P\{X=1\} = P\{(F, P), (P, F)\} = 1/2$$
$$P\{X=2\} = P\{(P, P)\} = 1/4$$

On obtient ce qu'on appelle une **loi de probabilité** ou **distribution**, que l'on peut représenter sous la forme d'un histogramme :

On remarque que si la largeur d'un rectangle vaut 1, la somme des aires des rectangles est égale à 1.



4.2. Densité de probabilité et fonctions de répartition

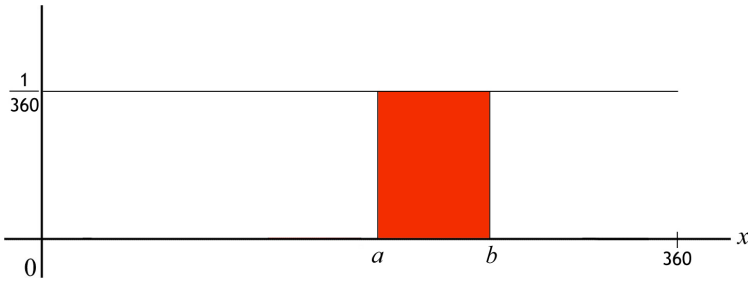
Imaginons une expérience où toutes les issues ont une probabilité 0. Un exemple simple est une girouette parfaite que l'on fait tourner à la main : à l'arrêt, elle peut indiquer n'importe quelle direction, et donc n'importe quel angle *réel* entre 0 et 360°. Il y a un nombre infini de valeurs possibles.
Certaines probabilités sont faciles à trouver, par exemple la probabilité d'indiquer un angle X compris entre 0 et 90° : $\Pr(0° \leq X \leq 90°) = 1/4$.
Mais qu'en est-il pour calculer la probabilité $\Pr(X = 90°)$? Comme il y a un nombre infini de valeurs possibles, la probabilité d'indiquer précisément 90° est nulle (!).
On peut représenter cette situation comme l'aire sous une courbe. Pour la girouette, cette courbe ressemble à ceci :

Remarquez que l'aire sous la courbe vaut 1.

La probabilité que la girouette indique un angle compris entre a et b est précisément l'aire orange sous la courbe, entre a et b.

Ici, l'aire orange, qui est la probabilité, vaut

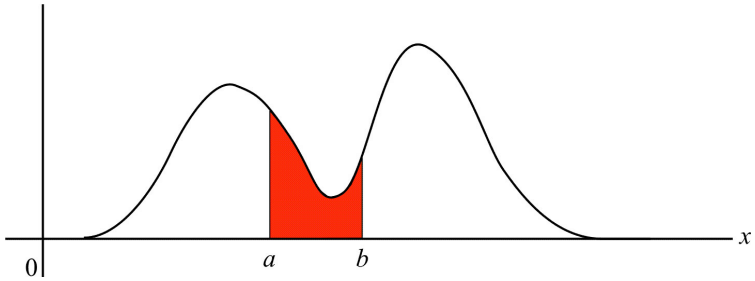
$$p=\frac{b-a}{360}$$



La courbe $y = f(x)$ est appelée **densité de probabilité** de la variable aléatoire continue X. Chaque variable aléatoire continue a sa propre fonction de densité. La probabilité $\Pr(a \le X \le b)$ est l'aire sous la courbe entre a et b.

Autrement dit, le calcul de cette probabilité revient au calcul d'une intégrale :

$$\Pr(a \leqslant X \leqslant b)=\int_a^b f(x)\,dx$$



Propriétés de $f(x)$

La courbe de densité $f(x)$ doit avoir certaines propriétés pour être compatible avec les axiomes des probabilités :

$$f(x)\geqslant 0 \qquad \int_{-\infty}^{+\infty} f(x)\,dx=1$$

Moyenne et variance

V. A. continues	à comparer avec	**V. A. discrètes**
$\mu=\int_{-\infty}^{+\infty} x\,f(x)\,dx$		$\mu=\sum_{\text{tout } x} x\,p(x)$
$\sigma^2=\int_{-\infty}^{+\infty} (x-\mu)^2\,f(x)\,dx$		$\sigma^2=\sum_{\text{tout } x} (x-\mu)^2\,p(x)$

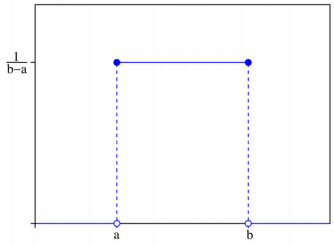
Fonction de répartition $F(x)$

On appelle **fonction de répartition** l'aire sous la courbe de densité de probabilité entre $-\infty$ et x.

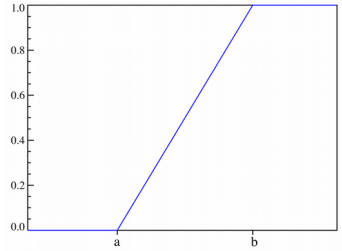
$$F(x)=\Pr(X \leqslant x)=\int_{-\infty}^{x} f(t)\,dt$$

4.3. La loi uniforme

La loi uniforme a la propriété suivante : tous les intervalles de même longueur ont la même probabilité. Cette loi continue est souvent notée $U(a, b)$.

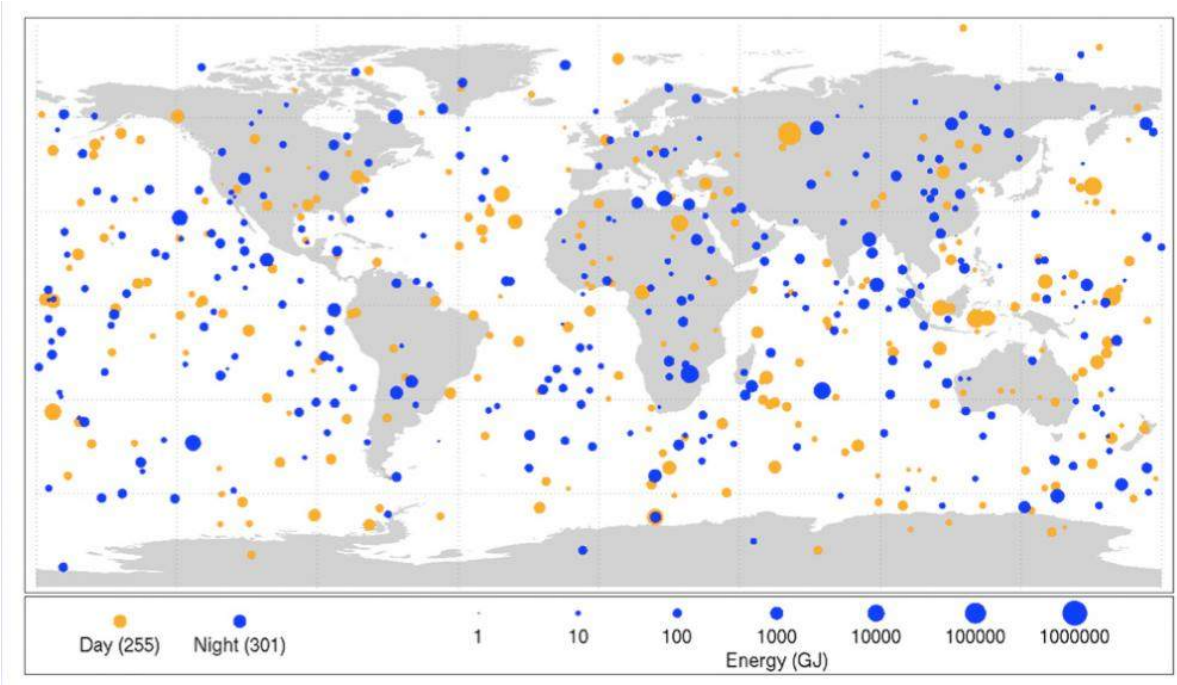


Loi de densité



Fonction de répartition

Tous les générateurs pseudo-aléatoires que l'on a vu jusqu'à présent produisent des nombres uniformément distribués.
La question qui se pose est maintenant la suivante : comment générer des nombres pseudo-aléatoires qui suivent une autre loi de probabilité que la loi uniforme ?



Cette carte de la NASA montre les 556 points d'impact de météorites dans l'atmosphère entre 1993 et 2014.
On voit que les impacts sont uniformément répartis sur tout le globe.

4.4. La transformation inverse

La **méthode de la transformée inverse** permet de produire une suite de nombres aléatoires de distribution donnée, à partir de l'expression de sa fonction de répartition F.

Soit U : $U(0, 1)$.
Si F est inversible (F^{-1} existe) alors $P(F^{-1}(\mathrm{U}) \leq x) = P(U \leq F(x)) = F(x) = P(X \leq x)$.
Donc, $F^{-1}(U)$ a la même distribution que X.

On procède alors comme suit :

1. générer z : $U(0, 1)$
2. poser $x = F^{-1}(z)$

Exemple 1

$$f(x) = \begin{cases} 2x & 0 \leqslant x \leqslant 1 \\ 0 & \text{sinon} \end{cases}$$

$z \sim U(0, 1)$ signifie que z suit une loi uniforme entre 0 et 1.

$$F(x) = \begin{cases} 0 & x \leqslant 0 \\ x^2 & 0 \leqslant x \leqslant 1 \\ 1 & x > 1 \end{cases}$$

$z = F(x) = x^2$, donc $x = F^{-1}(z) = \sqrt{z}$, où $z \sim U(0, 1)$

Exemple 2 : la loi exponentielle

Une **loi exponentielle** modélise la durée de vie d'un phénomène *sans mémoire*, ou *sans vieillissement*, ou *sans usure* : la probabilité que le phénomène dure au moins $s + t$ heures sachant qu'il a déjà duré t heures sera la même que la probabilité de durer s heures à partir de sa mise en fonction initiale. En d'autres termes, le fait que le phénomène ait duré pendant t heures ne change rien à son espérance de vie à partir du temps t.
Un domaine privilégié de la loi exponentielle est le domaine de la radioactivité. Chaque atome radioactif possède une durée de vie qui suit une loi exponentielle.
En théorie des files d'attente, l'arrivée de clients dans une file est souvent modélisée par une loi exponentielle.
Dans la loi exponentielle, $f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & \text{sinon} \end{cases}$ et $F(x) = 1 - e^{-\lambda x}$.

Donc, par la méthode de l'inversion :

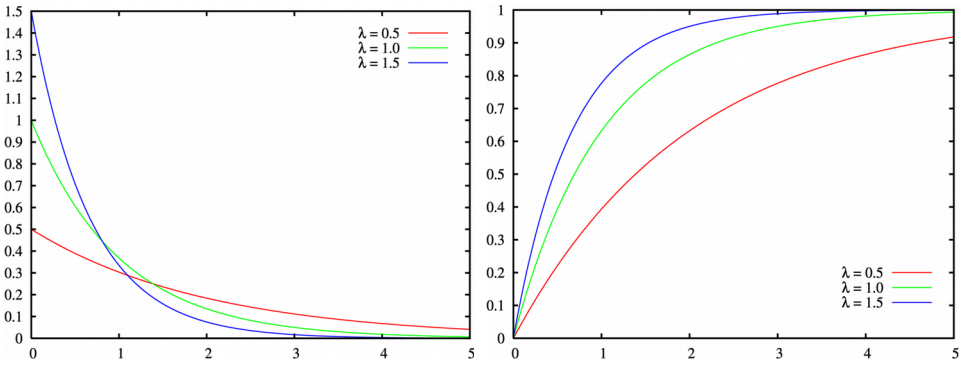
$$z = F(x) = 1 - e^{-\lambda x}$$

$$e^{-\lambda x} = 1 - z$$

$$x = -\frac{1}{\lambda}\ln(1 - z)$$

mais puisque $1-z \sim U(0, 1)$, alors $x = -\frac{1}{\lambda}\ln(z)$, où $z \sim U(0, 1)$

Source des images : Wikipédia



Loi de densité Fonction de répartition

Exemple 3 : une loi empirique

Une **loi empirique** est une loi ou formule issue de faits expérimentaux ou validée par l'expérience mais dont on ne connaît pas de base théorique, ou bien qu'on ne peut pas relier à une base théorique simple.
Par exemple, imaginons qu'un dé tétraédrique produise un 1 dans 30 % des cas, un 2 dans 10 % des cas, un 3 dans 25 % des cas et un 4 dans 35 % des cas. On a :

$$F(x) = \begin{cases} 0 & \text{si} \quad 0 \leqslant x < 1 \\ 0.3 & \text{si} \quad 1 \leqslant x < 2 \\ 0.4 & \text{si} \quad 2 \leqslant x < 3 \\ 0.65 & \text{si} \quad 3 \leqslant x < 4 \\ 1.0 & \text{si} \quad x \geqslant 4 \end{cases}$$

ou

$$F^{-1}(U) = \begin{cases} 1 & \text{si} \quad 0 \leqslant U < 0.3 \\ 2 & \text{si} \quad 0.3 \leqslant U < 0.4 \\ 3 & \text{si} \quad 0.4 \leqslant U < 0.65 \\ 4 & \text{si} \quad 0.65 \leqslant U < 1 \end{cases}$$

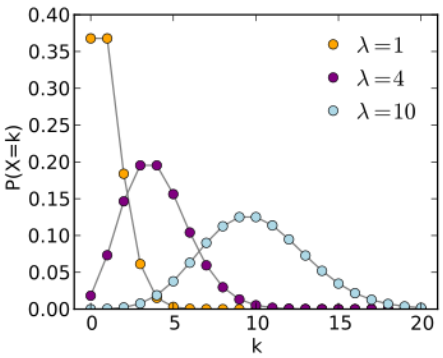
Exercice 19 Reprenez l'exercice 12 pour vérifier visuellement les trois exemples ci-dessus.

4.5. La loi de Poisson

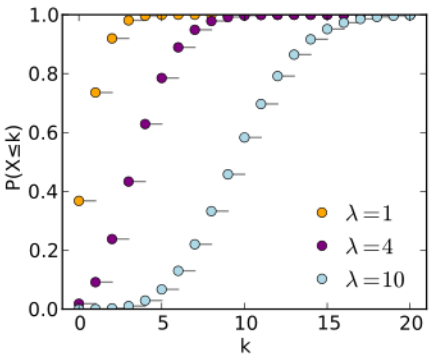
La **loi de Poisson** est une loi de probabilité discrète qui décrit le comportement du nombre d'événements se produisant dans un laps de temps fixé, si ces événements se produisent avec une fréquence moyenne connue et indépendamment du temps écoulé depuis l'événement précédent.
Le domaine d'application de la loi de Poisson a été longtemps limité à celui des événements rares comme les accidents dus aux coups de pied de cheval dans les armées.
Mais depuis quelques décennies son champ d'application s'est considérablement élargi. Actuellement, on l'utilise beaucoup dans les télécommunications (pour compter le nombre de communications dans un intervalle de temps donné), le contrôle de qualité statistique, la biologie (mutations), la météorologie, la finance pour modéliser la probabilité de défaut d'un crédit, …

La loi de densité est $f(x) = \frac{e^{-\lambda}\lambda^k}{k!}$

Source des images : Wikipédia



Loi de densité



Fonction de répartition

L'algorithme remarquablement simple qui suit, où $U \sim U(0, 1)$, produit en sortie une loi de Poisson X de paramètre λ :

1. Poser $X = 0$ et $P = 1$
2. $P = P \cdot U$
3. Si $P > e^{-\lambda}$ alors $X = X + 1$ et aller en 2
4. Afficher X

Exercice 20 Reprenez l'exercice 12 pour vérifier avec un histogramme que les nombres pseudo-aléatoires obtenus sont bien distribués selon la loi de Poisson.

4.6. La loi normale

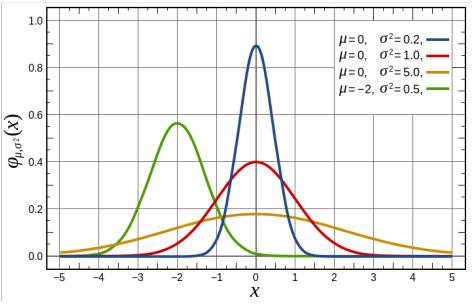
La **loi normale** est l'une des lois de probabilité les plus adaptées pour modéliser des phénomènes naturels issus de plusieurs événements aléatoires.

Un caractère observable et mesurable dans une population d'individus comparables a souvent une fréquence modélisée par une loi normale. C'est le cas par exemple de la taille humaine pour un âge donné (en séparant les hommes et les femmes), ou de la taille des becs dans une population d'oiseaux comme les pinsons de Darwin étudiés par Darwin.

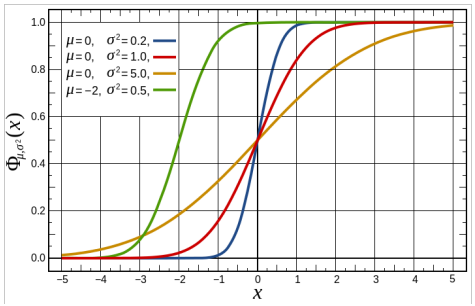
La loi de densité est $f(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$, où μ est la moyenne et σ l'écart-type.

Quand $\mu=0$ et $\sigma=1$, on parle de loi normale centrée réduite $N(0, 1)$: $f(x)=\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$

Source des images : Wikipédia



Loi de densité



Fonction de répartition

La méthode de Box-Muller utilise une représentation polaire de deux coordonnées uniformes donnée par les formules :

Si $U \sim U(0, 1)$, $V \sim U(0, 1)$,
alors $\sqrt{-2\ln(U)}\cos(2\pi V) \sim N(0, 1)$ et $\sqrt{-2\ln(U)}\sin(2\pi V) \sim N(0, 1)$
et les deux variables obtenues sont indépendantes.

Cet algorithme est simple à réaliser, mais le calcul d'un logarithme, d'une racine carrée et d'une fonction trigonométrique ralentit le traitement.
Une amélioration a été proposée par Marsaglia et Bray en 1964, en remplaçant les cosinus et sinus par les variables $\frac{V_1}{\sqrt{W}}$ et $\frac{V_2}{\sqrt{W}}$, où V_1 et V_2 sont indépendantes de loi $U(-1, 1)$ et $W=V_1^2+V_2^2$ lorsque $W<1$ (on rejette les couples qui ne vérifient pas cette dernière condition).

Ainsi : $V_1\sqrt{-2\frac{\ln(W)}{W}} \sim N(0, 1)$ et $V_2\sqrt{-2\frac{\ln(W)}{W}} \sim N(0, 1)$

Cet algorithme n'est pas plus lourd à mettre en œuvre et la simulation gagne en vitesse

Exercice 21

Programmez ces deux méthodes.
Reprenez l'exercice 12 pour vérifier que les nombres pseudo-aléatoires obtenus sont bien distribués selon la loi normale.

5. Applications

5.1. Estimation de probabilités

Calculer une probabilité n'est pas toujours chose aisée. L'usage d'un ordinateur peut s'avérer très utile.

Exercice 22

La tige de Lemoine (1873)
On coupe une tige de longueur ℓ en trois segments de longueur aléatoire. Estimez la probabilité que ces trois segments puissent former un triangle.

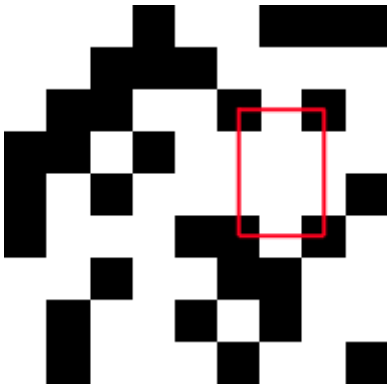
5.2. Optimisation

L'optimisation est une branche des mathématiques cherchant à modéliser, à analyser et à résoudre analytiquement ou numériquement les problèmes qui consistent à déterminer quelles sont la ou les solution(s) satisfaisant un objectif quantitatif tout en respectant d'éventuelles contraintes.

Exercice 23

Sur une grille n x n, on noircit k des n^2 cases de telle sorte que 4 des k cases choisies ne soient jamais les sommets d'un rectangle ayant ses côtés parallèles à ceux de l'échiquier. Quelle est la plus grande valeur de k pour laquelle ceci est possible ?

Solution admissible



Solution non admissible

Premier algorithme

1. Déterminer une case x au hasard.
2. Si la case x est noire, la rendre blanche. Aller à 4.
3. Si la case x n'est pas noire, la noircir après avoir vérifié qu'elle n'est pas le quatrième coin d'un rectangle.
4. Calculer le nombre de cases noires et garder le meilleur résultat obtenu en mémoire.
5. Retourner à 1.

Second algorithme

1. Poser $p = 1$.
2. Déterminer une case x au hasard.
3. Si la case x est noire, la rendre blanche avec une probabilité p. Aller à 5.
4. Si la case x n'est pas noire, la noircir après avoir vérifié qu'elle n'est pas le quatrième coin d'un rectangle.
5. Calculer le nombre de cases noires et garder le meilleur résultat obtenu en mémoire.

6. Diminuer la probabilité p.
7. Retourner à 2.

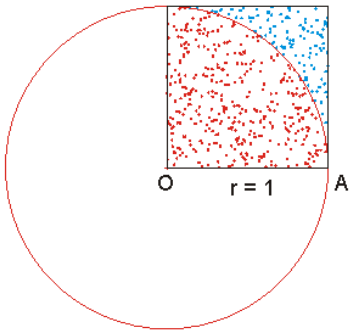
Programmez ces deux algorithmes et comparez leur efficacité.
Pour $n = 7$, la valeur maximale de k est 21.

5.3. Méthode de Monte-Carlo

Le terme **méthode de Monte-Carlo** désigne une famille de méthodes algorithmiques visant à calculer une valeur numérique approchée en utilisant des procédés aléatoires, c'est-à-dire des techniques probabilistes. Le nom de ces méthodes, qui fait allusion aux jeux de hasard pratiqués à Monte-Carlo, a été inventé en 1947 par Nicholas Metropolis, et publié pour la première fois en 1949 dans un article coécrit avec Stanislaw Ulam.

Exercice 24

Estimez la valeur de π en générant des poins dans le carré [0, 1] x [0, 1]. Soit un point M de coordonnées (x, y), où $0 < x < 1$ et $0 < y < 1$. On tire aléatoirement les valeurs de x et y. Le point M appartient au disque de centre (0, 0) et de rayon 1 si et seulement si $x^2 + y^2 < 1$.
En faisant le rapport du nombre de points dans le disque au nombre de tirages, on obtient une approximation du nombre π/4, si le nombre de tirages est grand.



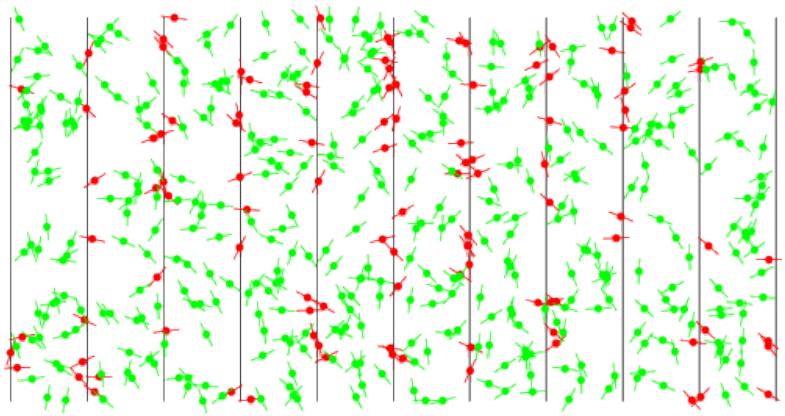
Programmez cette méthode.

Exercice 25

Les aiguilles de Buffon

En 1733, **Buffon** se pose la question suivante : si on jette au hasard une aiguille sur un parquet, quelle est la probabilité P qu'elle chevauche une rainure séparant deux lattes adjacentes ? Dans le schéma ci-dessous, les aiguilles rouges chevauchent les rainures (les trais noirs).

Georges-Louis Leclerc,
comte de **Buffon** (1707-1788)



Si a est la longueur d'une aiguille et ℓ la largeur d'une latte, on trouve $P = 2a/\pi\ell$. (on suppose que $a \leq \ell$).
Programmez cette méthode et adaptez-la pour obtenir une approximation de π.

5.4. Simulation

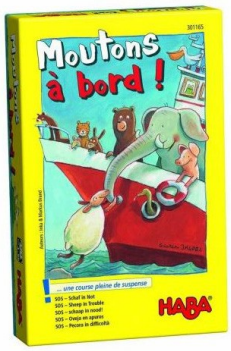
La simulation est apparue en même temps que l'informatique pour les besoins du projet Manhattan pendant la Seconde Guerre mondiale, afin de modéliser le processus de détonation nucléaire. Depuis, elle a évolué parallèlement à l'informatique.
Ces simulations informatiques sont rapidement devenues incontournables pour la modélisation des systèmes naturels en physique, chimie et biologie, mais également des systèmes humains en économie et en science sociale. Elles permettent de limiter le risque et d'éviter le coût d'une série d'épreuves réelles (p. ex. des essais de véhicules).
La simulation numérique est utilisée pour :

- prévoir l'état final d'un système connaissant son état initial ;
- déterminer les paramètres d'un système connaissant un ou plusieurs couples ;
- préparer des opérateurs à des conditions plus ou moins rares dans leur interaction avec un système complexe (simulation d'entraînement).

Exercice 26

Comment estimer le nombre de poissons dans un lac (sans vider le lac bien sûr) ?
Proposez une méthode basée sur le marquage des poissons.
Testez votre méthode par simulation.

Exercice 27



Moutons à bord est un jeu de hasard pur proposé par Inka et Markus Brand aux éditions Haba. Comme souvent dans les jeux de société, on peut se demander si le premier joueur est favorisé (ou défavorisé). Pour le savoir, nous allons simuler un très grand nombre de parties et tenir des statistiques.
Les règles sont très simples. Chaque joueur reçoit une tuile départ et une tuile arrivée.

Chaque joueur à son tour jette le dé puis retourne une tuile océan de la couleur indiquée et la place entre ses tuiles arrivée et départ, juste à gauche du bateau. Quand le dé tombe sur une couleur déjà présente dans son chemin, le joueur fait avancer son mouton d'autant de cases qu'il y a de rochers de cette couleur. Les chemins s'allongent donc au fil de la partie. Le premier joueur qui embarque son mouton sur le bateau gagne la partie.

Il y a 6 couleurs pour les tuiles océan et 4 tuiles de chaque couleur : 1 avec trois rochers, 2 avec deux rochers et 1 avec un rocher. On peut jouer de 2 à 4 joueurs.

Exercice 28

Cet énoncé s'inspire du livre *Devenez sorciers, devenez savants* de Georges Charpak et Henri Broch, dont est tiré cet extrait :

> Le présentateur se tourne vers la caméra principale, et d'un air très sérieux et enjôleur, regarde le téléspectateur droit dans les yeux en déclarant : « Allez-y ! Allumez cinq ou six lampes à côté de vous. » Puis il se tourne vers le médium et demande : « Vous pensez réellement pouvoir le faire ? » Après quelques moments d'hésitation, le mage se prononce : « J'espère avoir suffisamment de concentration ce soir, mais je ne suis pas dans les conditions vraiment idéales ; pour produire ce genre de phénomène à distance, d'habitude, je me retire pendant plusieurs jours dans une solitude totale et une profonde obscurité, après un jeûne strict. »
> Si jamais il échoue, le public le mettra au compte des circonstances et non pas de ces compétences. Et, pourtant, le médium n'échoue pas. Des ampoules grillent chez les téléspectateurs qui regardent cette émission. Ils font part au standard téléphonique de la chaîne qui diffuse en direct cet extraordinaire moment de culture. Le médium a donc bien réussi - comme il le prétendait -, en concentrant sa puissance spirituelle sur la matière, à griller des ampoules électriques à distance.
> Supposons que cette émission soit suivie par environ un million de téléspectateurs et qu'elle dure une heure ou plus. Cela signifie qu'environ cinq à six millions d'ampoules ont été allumées pendant une heure ou plus. Supposons que ce nombre soit de deux millions.
> La durée de vie moyenne d'une ampoule à incandescence est de mille heures. Ce qui signifie que, pendant la durée de l'émission, il y aura environ deux mille lampes grillées.

Comment pourrait-on réaliser une simulation numérique de cette expérience ?
Il est préférable de commencer les simulations avec de plus petites valeurs : 10'000 ampoules pour une durée de vie de 100 heures et une expérience qui s'étale sur 500 heures.
Pour réaliser cette simulation, on suppose donc qu'on a un tableau de 10'000 ampoules. Chaque case de ce tableau contient la durée de vie restante d'une ampoule, simulée selon une loi exponentielle de paramètre $\mu=1/100$. Au départ, toutes les durées de vie sont nulles. On considère qu'à chaque itération, une heure passe. Lors de chaque itération, pour chaque ampoule, on vérifie sa durée de vie restante. Si celle-ci est nulle, on la remplace par une autre dont on choisit aléatoirement la durée de vie (arrondie à l'entier le plus proche). Si la durée de vie n'est pas nulle, on la diminue d'une heure. À chaque itération, on compte le nombre d'ampoules grillées pour en faire la moyenne au bout de n itérations.