# Générateurs de nombres aléatoires

Un générateur de nombre aléatoire est une procédure permettant de reproduire le hasard. Plus précisément, on souhaite construire un programme qui génère des nombres entre 0 et 1 selon la loi uniforme.

Les générateurs que nous allons étudier sont tous définis par des suites récurrentes d'entiers à un pas. Une simple division permet ensuite de se ramener à des nombres de $[0, 1]$. L'objectif est que la suite obtenue ressemble à une suite aléatoire de nombres indépendants les uns des autres (ce qui est clairement faux).

L'objectif de ce projet est de tester la qualité de plusieurs générateurs en étudiant leurs propriétés statistiques et en les comparant à celles de la loi uniforme puis de les utiliser pour répondre à un problème. Les résultats et remarques devront s'appuyer sur de nombreuses simulations.

## 1 Construction des générateurs

**Le générateur Randu** développé par IBM dans les années 60. Il est défini par la suite récurrente :

$$u_0 \in \mathbf{N} \text{ impair} \quad \text{et} \quad \forall k, \quad u_{k+1} = 65539 u_k \mod 2^{31}.$$

**Le générateur de Windows** (jusqu'en 2000). Il est défini par la suite récurrente :

$$u_0 \in \mathbf{N} \quad \text{et} \quad \forall k, \quad u_{k+1} = 214013 u_k + 2531011 \mod 2^{16}$$

**Le générateur de Turbo Pascal**. Il est défini par la suite récurrente :

$$u_0 \in \mathbf{N} \quad \text{et} \quad \forall k, \quad u_{k+1} = 129 u_k + 907633385 \mod 2^{32}.$$

**Le générateur de Python** définissant la commande *random()*. Il repose sur le twist de Mersenne.

Pour les trois premiers générateurs, on obtient des suites d'entiers compris entre 0 et $2^k$. Pour obtenir un nombre réel compris entre 0 et 1, il suffit de diviser ces entiers par $2^k$.

1. Expliquer pourquoi le choix de $n$ est toujours une puissance de 2.
2. Dans un script Octave, commencer par définir une variable globale, c'est-à-dire une variable qui peut être modifiée par des programmes : *global u ;*
   On peut ensuite fixer une valeur initiale pour cette variable : *u=123* par exemple.
   Enfin, écrire pour chaque générateur étudié, un programme qui transforme $u$ en $au + c$ mod $n$ et qui renvoie la valeur de $\frac{u}{n}$. Il faut de nouveau déclarer $u$ comme variable globale du programme. Pour résumer :
   *global u ;*
   *u=u0 (à définir)*

   *function rand1()*
   *global u ;*
   *...*
   *endfunction*
3. Écrire des programmes qui génèrent des suites de nombres successifs tirés à l'aide des générateurs précédents.

# 2 Tests statistiques

**Densité**

1. Des nombres tirés selon la loi uniforme s'équirépartissent dans l'intervalle $[0, 1]$. Cela signifie qu'ils « remplissent »l'intervalle de manière homogène.
   Tester cela pour les générateurs en représentant les suites de points simulés.
2. Soyons plus précis. Pour un intervalle $[a, b]$ donné, la probabilité qu'un nombre tiré selon la loi uniforme appartienne à cet intervalle est égale à $b - a$.
   Vérifier cela empiriquement pour nos générateurs en simulant un grand nombre de valeurs. Recommencer avec plusieurs intervalles. Peut-on trouver des intervalles pour lesquels nos générateurs échouent forcément ?

**Indépendance**

On souhaite que les différents tirages effectués avec les générateurs soient deux à deux indépendants. Cela signifie vulgairement qu'une valeur obtenue pour un tirage n'a aucune influence sur les valeurs obtenues pour d'autres tirages.

1. Expliquer pourquoi les tirages successifs effectués avec nos générateurs ne peuvent pas être indépendants.
   Ils peuvent cependant avoir l'air indépendants. Si on considère une suite $(x_k)$ de nombres indépendants tirés selon la loi uniforme et si on considère les couples $(x_k, x_{k+1})$ dans le carré $[0, 1]^2$, alors ceux-ci sont équirépartis dans le carré. Si un générateur est correct, on retrouve cette propriété.
2. Tester ainsi graphiquement l'indépendance des tirages successifs des générateurs.
3. Tester également l'indépendance en estimant empiriquement une probabilité de la forme $\mathbf{P}(x_i \in [a, b] \text{ et } x_{i+1} \in [c, d])$.
4. Faire de même avec les triplets de tirages successifs : on représentera un grand nombre de triplets de la forme $(x_i, x_{i+1}, x_{i+2})$ et on estimera certaines probabilités.

   L'indépendance est le point le plus délicat de la simulation de nombre aléatoires. Il existe des tests plus subtils et plus précis que ceux évoqués ici comme le test du $\chi^2$ et le test du poker.

**Fonction de répartition et test de Kolmogorov-Smirnov.**

La fonction de répartition de la loi uniforme est donnée sur $[0, 1]$ par $F(x) = x$.

1. Écrire un programme *Répartition(l)* qui pour une liste de nombres dans $[0, 1]$, trace la fonction définie sur $[0, 1]$ par $F_l(x) = \frac{k}{N}$ où $N$ est la taille de la liste et $k$ est le nombre d'éléments de la liste inférieurs à $x$.
2. Pour chaque générateur, écrire un programme *Kolmogorov(N)* qui calcule, pour une suite $l$ de $N$ nombres générés, la distance entre la fonction de répartition théorique et sa version empirique : $\varepsilon_N = \|F_l - F\|_\infty$.

   Si le générateur est correct, $\varepsilon_N$ doit être faible et tendre vers 0 quand $N$ tend vers $+\infty$. Le test de Kolmogorov-Smirnov précise cela : pour chaque valeur de $N$, il existe une valeur $d_N$ telle que si $\varepsilon_N > d_N$, alors on peut considérer (avec un risque de 5%) que les nombres générés ne le sont pas selon la loi uniforme. Donnons quelques valeurs :

$$d_{20} = 0.294, \quad d_{50} = 0.188, \quad d_{100} = 0.150 \text{ et pour } N > 100, \quad d_N = \frac{1.358}{\sqrt{N}}.$$

3. Appliquer le test de Kolmogorov-Smirnov aux générateurs étudiés.

# 3 Le problème du recruteur

Un recruteur doit auditionner 100 candidats pour un poste. Il y a une contrainte : après chaque audition, il doit décider s'il recrute le candidat ou non et sa décision est alors irrévocable. Comment doit-il s'y prendre pour espérer recruter le meilleur des candidats ?

On propose la stratégie suivante : il auditionne $N$ candidats qu'il ne recrute pas. Puis il recrutera, parmi les candidats suivants, le premier candidat qui sera meilleur que les $N$ premiers. Si aucun n'est meilleur, il sera obligé de recruter le centième candidat.

En utilisant un ou plusieurs de vos générateurs, générer 100 nombres aléatoires correspondants aux niveaux des 100 candidats.

À partir d'un grand nombre de simulations, déterminer empiriquement, pour plusieurs valeurs de $N$ de votre choix, la probabilité qu'il recrute le meilleur candidat. Quelle semble être la meilleure valeur de $N$ ?