Fiche 1 – Nombres pseudo-aléatoires

«Anyone who considers arithmetical methods of producing random numbers is, of course, in a state of sin.» J. von Neumann

Introduction

Pour simuler un modèle stochastique, il est nécessaire de disposer d'une source de nombres «au hasard» susceptibles de jouer en pratique le rôle des variables aléatoires qui interviennent dans la définition du modèle. La génération de tels nombres s'accomplit en général au moyen de deux étapes de nature différente :

- la génération de nombres au hasard $U_1, U_2, \ldots$ jouant le rôle de variables aléatoires i.i.d. uniformes sur l'intervalle $[0, 1]$,
- la transformation des nombres $U_1, U_2, \ldots$ précédents en des nombres susceptibles de jouer le rôle des variables aléatoires intervenant dans la définition du modèle, et qui ne sont en général ni i.i.d. ni uniformes sur $[0, 1]$.

On parle en général de nombres **pseudo-aléatoires** pour désigner les nombres au hasard qui apparaissent au cours de ces deux étapes, pour souligner leur différence (que nous étudierons en détail dans la suite) vis-à-vis de véritables suites de variables aléatoires indépendantes identiquement distribuées.

La présente fiche est consacrée à l'étude de la première de ces deux étapes, c'est-à-dire à la génération de pseudo suites de variables aléatoires i.i.d. uniformes sur $[0, 1]$ (l'étape de transformation faisant l'objet de la fiche suivante). Cette question est cruciale pour la mise en œuvre pratique des simulations, mais elle soulève de nombreuses difficultés d'ordre tant conceptuel que pratique, aucune solution complètement satisfaisante n'ayant été proposée à ce jour. Nous nous contenterons d'en illustrer rapidement les différents aspects, en renvoyant à des références spécialisées pour plus de détails. Mentionnons, avant d'entrer dans le vif du sujet, qu'il n'existe à l'heure actuelle aucun standard concernant la génération de nombres pseudo-aléatoires, et que de nombreux procédés de génération employés aujourd'hui sont inadaptés à l'usage actuel, ayant conduit par le passé et conduisant encore à des résultats de simulation erronés (voir par exemple [G] et [FLW]). Qui plus est, l'usage consistant à ne pas documenter convenablement les procédés de génération de nombres pseudo-aléatoires

utilisés, en traitant ceux-ci comme des «boîtes noires» est encore largement répandu, y compris dans des logiciels récents et supposés de bonne qualité, alors que la spécification des procédés de génération employés doit pourtant être partie intégrante de la description d'un protocole de simulation. Cet état de fait rend particulièrement nécessaire une connaissance minimale de la question, avant de se lancer dans la simulation à proprement parler.

Pour des informations plus détaillées que celles présentées dans cette fiche, nous renvoyons à l'ouvrage classique de Knuth [K] cité dans la bibliographie, et aux excellents articles de synthèse de P. L'Ecuyer [LE] et [LE2], [LE3]. Le sites internet pLAB, que vous trouverez à l'adresse `http://random.mat.sbg.ac.at/` , ainsi que les pages de P. L'Ecuyer `http://www.iro.umontreal.ca/~lecuyer` et de L. Devroye `http://cgm.cs.mcgill.ca/~luc/` contiennent également de très nombreuses références ainsi que des implémentations explicites.

1 Sources de nombres pseudo-aléatoires

1.1 Générateurs algorithmiques

1.1.1 Principe général

Vous avez très probablement déjà utilisé le générateur par défaut de votre langage de programmation ou logiciel favori. En langage C, la fonction `rand` de la bibliothèque standard est supposée produire à chaque appel un nombre entier aléatoire uniformément distribué entre 0 et la constante prédéfinie RAND_MAX (égale par exemple à $2^{31}-1$ dans certaines versions d'UNIX), et indépendant des nombres produits par les appels précédents (dans la documentation, l'uniformité de la distribution et l'indépendance mutuelle des résultats fournis par des appels successifs à la fonction n'est en général pas mentionnée explicitement). Comment l'ordinateur peut-il produire des nombres censés constituer des entiers aléatoires, alors que son fonctionnement est en principe complètement déterministe ? Qu'y a-t-il dans la «boîte noire» que constitue la fonction `rand` qui permette à l'ordinateur, au moins en apparence, de disposer d'une source de hasard ?

En fait, voici à quoi peut ressembler la fonction `rand` du C[1].

```
int rand()
```

[1] Il s'agit de l'une des implémentations existantes, le procédé n'étant nullement standardisé.

```
/* Produces a random number between 0 and 32767.*/
{
    rand_seed = rand_seed * 1103515245 +12345;
    return (unsigned int)(rand_seed / 65536) % 32768;
}
```

Chaque appel à la fonction `rand` consiste donc à faire évoluer une variable globale `rand_seed` au moyen d'une relation de récurrence affine, et à renvoyer à l'utilisateur une certaine fonction de cette variable.

Pour obtenir des nombres réels entre 0 et 1 plutôt que des entiers entre 0 et RAND_MAX, il suffit de diviser le nombre obtenu par RAND_MAX.

On note que, dans le code ci-dessus, le seul élément possiblement aléatoire dans l'affaire est l'initialisation de la variable `rand_seed`, la suite des valeurs produites par les appels successifs à cette fonction étant alors entièrement déterminée.

L'exemple ci-dessus constitue un exemple extrêmement simple de générateur algorithmique de nombres pseudo-aléatoires. De manière générale, un procédé algorithmique de génération de nombres pseudo-aléatoires est caractérisé par un triplet (S, f, g), où

- S est un (grand) ensemble fini,
- f est une application de S dans lui-même,
- g est une fonction de S dans $[0, 1]$.

Le générateur fonctionne en itérant la fonction f à partir d'une valeur initiale $s_0 \in S$, appelée graine, et choisie par l'utilisateur, devant parfois satisfaire un certain nombre de contraintes (en plus d'être un élément de S). En notant $(x_n)_{n\geq 1}$ la liste des valeurs successives (entre 0 et 1) renvoyées par le générateur, on a

$$x_n = g(\underbrace{f \circ \cdots \circ f}_{n \text{ fois}}(s_0)).$$

En pratique, le générateur ne garde en mémoire que son état courant $s_n \in S$, initialisé par s_0, et mis à jour lors de chaque nouvel appel au moyen de la relation de récurrence

$$s_n = f(s_{n-1}),$$

la valeur effectivement renvoyée étant égale à $g(s_n)$.

Ainsi, le choix de la graine détermine entièrement la suite de nombres pseudo-aléatoires produite par le générateur. Une fois celle-ci choisie, rien d'aléatoire ne

subsiste dans le fonctionnement du générateur, dont les sorties sont pourtant supposées mimer des variables aléatoires indépendantes et de loi uniforme sur $[0, 1]$. Nous discuterons dans une partie ultérieure la question de savoir quel degré de similitude peut exister entre de véritables suites de variables aléatoires i.i.d. et les nombres fournis par un générateur pseudo-aléatoire du type décrit ci-dessus.

En plus de (S, f, g) s'ajoute parfois une application h d'un ensemble H «simple» (par exemple les entiers de 0 à N) dans S, et qui permet à l'utilisateur de choisir indirectement la graine en fournissant seulement un élément t_0 de l'ensemble H, la graine étant alors prise égale à $h(t_0)$.

Le principe théorique de fonctionnement d'un générateur de nombres pseudo-aléatoires comprend la description précise des divers éléments présentés ci-dessus, et apparaît en général dans une publication scientifique de référence, qui examine également le comportement du générateur vis-à-vis de critères de qualité et de performances techniques que nous aborderons en détail dans les parties suivantes. Mentionnons dès maintenant que la définition d'un procédé permettant de produire des nombres pseudo-aléatoires satisfaisants, c'est-à-dire, pouvant être utilisés en pratique comme un bon substitut de véritables variables aléatoires, est un problème complexe, dont le traitement ne saurait être laissé au hasard.

En plus d'une description théorique, diverses implémentations d'un procédé de génération sont généralement disponibles, dans les principaux langages de programmation (C, C++, Fortran, Java,...), sous forme de morceaux de morceaux de code ou de bibliothèques, parfois intégrables (ou déjà intégrés) à des logiciels tels que R ou MATLAB.

Dans le code correspondant à l'implémentation d'un générateur de nombres pseudo-aléatoires, on s'attend à trouver au minimum :

- une procédure permettant d'initialiser la graine à partir d'un choix effectué par l'utilisateur (par exemple `srand` en C),
- une procédure renvoyant un nombre pseudo-aléatoire calculé en appliquant la fonction g à l'état courant du générateur, et mettant à jour l'état courant en lui appliquant la fonction f (par exemple `rand` en C)
- des commentaires utiles et des exemples, concernant les détails de la mise en œuvre pratique, en particulier les questions de portabilité (le calcul des fonctions f et g peut faire appel à des manipulations arithmétiques élaborées, pour lesquelles la portabilité du code n'est pas absolument garantie).

Une valeur par défaut pour la graine est souvent choisie lorsque l'utilisateur n'en fournit pas. Pour initialiser la graine, on doit a priori faire appel à un tirage aléatoire, pouvant être effectué de diverses manières. Un choix fréquent est d'utiliser les derniers chiffres du nombre indiquant le temps (en secondes, ou en centièmes de secondes...) écoulé depuis une date lointaine fixée, ce nombre étant en général fourni par l'ordinateur.

1.1.2 Des exemples

Nous décrivons dans ce qui suit les principales familles de générateurs de nombres pseudo-aléatoires utilisés en pratique. Ceux-ci reposent sur des opérations arithmétiques relativement élaborées, et des procédés d'implémentation ingénieux ont souvent été développés, jouant sur certaines propriétés arithmétiques et reposant parfois sur certains choix particuliers de paramètres, de façon à rendre rapide le calcul de f et g. La rapidité d'exécution des appels au générateur est un paramètre vraiment important en pratique, car les simulations massives font souvent appel à un grand nombre d'entre eux. La gourmandise en mémoire des procédés employés peut également avoir de l'importance si l'on effectue un grand nombre d'utilisations du procédé en parallèle, ce qui est naturel en simulation stochastique.

Une première famille d'exemples est constituée par la classe des générateurs linéaires congruentiels, définis par :

$$S = \{0, \ldots, m-1\},\ f(s) = (as + c) \bmod m,\ g(s) = s/m,$$

où m est un grand entier. Voici quelques exemples concrets de choix des paramètres :

- la fonction `rand()` du langage C ANSI utilise ce type de générateur avec les paramètres :
$$m = 2^{31}\ ,\ a = 1103515245\ ,\ c = 12345,$$
- la fonction `drand48()` (toujours en C ANSI) qui utilise les paramètres :
$$m = 2^{48}\ ,\ a = 25214903917\ ,\ c = 11,$$
- le générateur `RANDU` implanté sur les ordinateurs IBM dans les années 60 :
$$m = 2^{31}\ ,\ a = 65539\ ,\ c = 0,$$

– le générateur du logiciel MAPLE :

$$m = 10^{12} - 11 \ , \ a = 427419669081 \ , \ c = 0,$$

Bien entendu, les valeurs particulières des paramètres proposées ci-dessus ne doivent rien au hasard, et sont la plupart du temps le produit d'une optimisation minutieuse, en rapport avec les critères de qualité que nous décrirons plus tard. Un choix quelconque de ces paramètres conduit généralement à de très mauvais résultats, c'est-à-dire à des générateurs dont les sorties miment très mal des familles de variables aléatoires uniformes et indépendantes (en donnant lieu, par exemple à des suites de nombres périodiques de très courte période), et il est donc hors de question d'utiliser sans raison des valeurs différentes de celles recommandées ou pré-implémentées.

Les générateurs linéaires congruentiels sont, de loin, les plus simples, mais pas nécessairement les plus performants, au moins sous leur forme la plus élémentaire, comme nous le verrons plus bas.

De nombreux raffinements de ces générateurs ont donc été proposés, par exemple, en utilisant des récurrences linéaires multiples (on parle alors de générateurs multi-récursifs) :

$$S = \{0, \ldots, m-1\}^k, \ f(s_1, \ldots, s_k) = (s_2, \ldots, s_k, a_1 s_1 + \cdots + a_k s_k \bmod m),$$

$$g(s_1, \ldots, s_k) = s_k/m,$$

ou encore, en combinant additivement les résultats de plusieurs générateurs multi-récursifs :

$$S = S_1 \times \cdots S_p, \ f(s_1, \ldots, s_p) = (f_1(s_1), \ldots, f_p(s_p)),$$

où chaque (S_i, f_i) est un générateur multi-récursif du type précédent, avec $S_i = \{0, \ldots, m_i - 1\}^{k_i}$, et

$$g(s_1, \ldots, s_p) = s_1/m_1 + \cdots + s_p/m_p \bmod 1,$$

(voir par exemple [LE4] pour des exemples d'implémentation avec des choix spécifiques de paramètres).

Une autre approche, dite «digitale» consiste à utiliser des récurrences matricielles modulo 2, avec pour espace $S = \{0, 1\}^k$ et des récurrences données par

$$f(s) = As \bmod 2,$$

où A est une matrice $k \times k$, et où

$$g(s) = \sum_{i=1}^{w} [(Bs)_i \bmod 2] 2^{-i},$$

et B est une matrice $k \times w$. Le générateur opère donc directement sur la représentation en base 2 des nombres manipulés, d'où le nom. Le choix d'opérations modulo 2 est particulièrement intéressant du point de vue de l'implémentation, car les opérations matricielles correspondant alors à des opérations simples (échange, décalage) sur les bits. C'est par exemple le cas des générateurs Mersenne Twister, (voir [MN]), ou des classes de générateurs WELL ou XORSHIFT (voir [LE5]).

Il existe également des générateurs reposant sur des itérations non-linéaires (générateurs «inversifs»), par exemple du type : $S = \{0, \ldots, p-1\}$,

$$f(x) = ax^{p-1} + b \bmod p,$$

$g(x) = x/p$. où p est un entier premier, (voir [Ni] pour des exemples d'implémentation).

Diverses combinaisons entre des générateurs des types précédents existent également.

1.1.3 Intérêt des sources algorithmiques

Les procédés algorithmiques de génération de nombres pseudo-aléatoires, dont nous venons de décrire quelques exemples, sont intéressants à plusieurs titres.

- la répétabilité : une fois la graine choisie par l'utilisateur (et stockée), il est possible de reproduire intégralement la séquence des nombres pseudo-aléatoires produits, ce qui peut être important, par exemple pour vérifier des résultats obtenus par simulation, ou débugger des codes de simulation ;
- la facilité d'utilisation : dans la plupart des implémentations, la génération est rapide et sans limitation réelle quant à la quantité d'appels au générateur que l'on peut effectuer ;
- la standardisation : le fait d'employer une procédure standardisée permet de disposer d'informations fiables sur la qualité et les performances des procédures que utilisées, en provenance de divers chercheurs et utilisateurs, sous la forme de publications scientifique et de documentation variée.

1.2 Sources physiques

Une idée qui peut sembler naturelle pour obtenir des suites de nombres aléatoires, consiste à choisir des valeurs issues d'un phénomène physique considéré comme aléatoire (lancer d'une pièce de monnaie, roulette, brassage de billes de loto, ou encore désintégration de particules radioactives, bruit de grenaille de composants électroniques,...)

Des dispositifs de ce type, effectuant des mesures en temps réel (au fur et à mesure que l'ordinateur «réclame» des nombres aléatoires), existent effectivement. Ils présentent plusieurs désavantages par rapport aux sources algorithmiques. D'une part, leur fonctionnement en temps réel est difficilement conciliable avec l'objectif de générer rapidement une grande quantité de nombres pseudo-aléatoires, du fait de la lourdeur de ces dispositifs et des contraintes pratiques (temps nécessaire à l'obtention des mesures, codage et transmission des données,...). D'autre part, les suites de nombres ainsi obtenues ne sont pas reproductibles. Qui plus est, ces dispositifs sont coûteux à acquérir, tandis que les procédés de génération algorithmique de nombres pseudo-aléatoires sont la plupart du temps gratuits et disponibles sur la toile, voire déjà installés dans votre langage de programmation ou logiciel favori.

Il a été envisagé d'enregistrer (par exemple sur un disque optique) un grand nombre de valeurs aléatoires mesurées expérimentalement, de façon à pouvoir les utiliser librement ensuite, afin de surmonter certaines des difficultés évoquées ci-dessus. Hormis le problème posé par la quantité de données à stocker, c'est, plus fondamentalement, le manque de fiabilité des dispositifs expérimentaux pour produire des suites de nombres aléatoires véritablement indépendants et uniformément distribués (dû, par exemple, aux imperfections des capteurs, aux interactions du dispositif avec son environnement,...) qui fait que ce type de méthode est très peu employée pour la génération massive de nombres au hasard en simulation. En revanche, ces méthodes peuvent présenter un intérêt dans notre contexte, pour initialiser les graines de générateurs algorithmiques, en fournissant le petit élément d'aléa sur lequel la suite produite par le générateur est construite.

1.3 Cryptographie

La cryptographie est un domaine bien distinct de la simulation stochastique, où l'on a parfois recours à des nombres au hasard, par exemple pour fabriquer des clés de

cryptage. Dans ce contexte, il n'est pas forcément nécessaire de produire rapidement un grand nombre de valeurs, et c'est davantage l'imprévisibilité des nombres produits que leur ressemblance avec des variables aléatoires de loi prescrite qui importe. Des sources physiques de nombre au hasard peuvent alors se révéler intéressantes. Inversement, soulignons le danger qu'il y a à utiliser la plupart des procédés algorithmiques de génération de nombres pseudo-aléatoires dans un contexte cryptographique. En effet, la simple connaissance du principe général de fonctionnement du générateur (c'est-à-dire la forme générale des fonctions f, g, h) peut suffire pour identifier les paramètres de celui-ci ainsi que la valeur de la graine, à partir seulement d'un échantillon des nombres qu'il produit. Il devient alors possible de reconstituer la totalité de la suite des nombres produits, ce qui peut s'avérer catastrophique pour la sécurité de l'application envisagée.

2 En quel sens les suites produites par un générateur algorithmique peuvent-elles être considérées comme aléatoires ?

Nous l'avons vu, le seul élément d'aléa véritable pouvant être présent dans les suites de nombres produites par des procédés algorithmiques réside dans le choix de la graine, les nombres produits étant entièrement déterminés une fois celle-ci choisie. On s'attend cependant à ce qu'un procédé de génération algorithmique correct produise des suites de nombres qui «ressemblent» d'assez près à des nombres aléatoires indépendants et uniformes sur $[0, 1]$ pour que la validité des résultats obtenus par simulation soit assurée.

Dans cette partie, nous discutons des tentatives théoriques de formaliser cette notion.

Pour simplifier la discussion, nous supposons que l'on dispose d'un générateur produisant, non pas des nombres réels entre 0 et 1, mais simplement des 0 ou des 1, censés mimer le comportement d'une suite $X_1, \ldots, X_N$ de variables aléatoires indépendantes et identiquement distribuées sur $\{0, 1\}$ vérifiant $P(X_i = 1) = P(X_i = 0) = 1/2$.

La question n'est pas aussi simple qu'elle en a l'air. Commençons par une petite discussion informelle.

Etant donnée une suite de nombres $x_1, \ldots, x_N \in \{0,1\}$, que signifie le fait que $x_1, \ldots, x_N$ puisse être considérée en pratique comme la réalisation d'une suite $X_1, \ldots, X_N$ de variables aléatoires indépendantes et identiquement distribuées sur $\{0,1\}$ vérifiant $P(X_i = 1) = P(X_i = 0) = 1/2$? Malheureusement (et de manière un peu paradoxale), la théorie des probabilités n'apporte pas de réponse directe à cette question. Elle définit seulement $X_1, \ldots, X_N$ comme une famille de fonctions définies sur un même espace probabilisé (Ω, P), et à valeurs dans $\{0,1\}$, vérifiant, pour tout $(y_1, \ldots, y_N) \in \{0,1\}^N$, la relation

$$P(\{\omega \in \Omega \ : \ X_1(\omega) = y_1, \ldots, X_N(\omega) = y_N) = \frac{1}{2^N}.$$

Une réalisation de cette suite de variables aléatoires est simplement une suite de la forme $X_1(\omega), \ldots, X_N(\omega)$, où ω peut prendre n'importe quelle valeur dans Ω. Autrement dit, n'importe quelle suite de longueur N constituée de 0 et de 1 est une réalisation de cette suite de variables aléatoires, et, d'ailleurs, ces suites ont chacune exactement la même probabilité d'apparaître, à savoir $\frac{1}{2^N}$. Ceci ne nous avance guère...

Pourtant, il semble clair intuitivement qu'une suite telle que

$$u_{20} = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$$

ressemble moins à une réalisation de $X_1, \ldots, X_n$ qu'une suite telle que

$$v_{20} = (0,0,1,1,0,0,0,0,0,1,1,0,1,1,1,1,0,1,1,1).$$

En effet, on s'attend à ce que la proportion de 0 et de 1 dans la suite $X_1, \ldots, X_{20}$ soit voisine de $1/2$ (cette proportion est de 100% de 0 et de 0% de 1 dans u_{20}, contre 45% de 0 et 55% de 1 dans v_{20}). Essayons de préciser cet argument intuitif. On peut vérifier que la probabilité pour que la fréquence de 0 (et donc de 1) dans la suite $X_1, \ldots, X_{20}$ doit être comprise entre 35% et 65% avec une probabilité d'environ 90%. Par conséquent, la suite u_{20}apparaît comme surprenante vis-à-vis de ce critère, tandis que v_{20} paraît dans la norme. Ce critère est-il suffisant ? Evidemment, non. Par exemple, la suite

$$w_{20} = (0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1)$$

vérifie également le fait que sa proportion de 0 et de 1 est comprise entre 35% et 65%, mais ne paraît pas très plausible en tant que réalisation de $X_1, \ldots, X_{20}$, par

exemple parce que la probabilité de n'observer aucune paire de 0 consécutifs dans la suite $X_1, \ldots, X_{20}$ est inférieure à 2% (exercice : le prouver).

On voit ainsi se dessiner la définition informelle suivante : (D1) on acceptera une suite $x_1, \ldots, x_N$ de 0 et de 1 comme une réalisation de $X_1, \ldots, X_N$ lorsque celle-ci vérifie toutes les propriétés que se doit de posséder $X_1, \ldots, X_N$ avec une forte probabilité. Bien entendu, cette définition est assez vague, et en particulier, il faut préciser ce que l'on entend par «forte» probabilité. Dans la limite où N tend vers l'infini, on serait tenté de ne conserver que les propriétés de $X_1, X_2, \ldots$ valables avec une probabilité égale à 1 ; on voit en particulier que cette notion doit dépendre de N, plus le nombre de termes de la suite étant grand, plus il semble possible d'être exigeant en termes de probabilités.

Malheureusement, toute tentative naïve de préciser (D1) ne donne lieu qu'à une définition vide : aucune suite ne peut la vérifier. Reprenons l'exemple de la suite v_{20}. La probabilité pour la suite $X_1, \ldots, X_{20}$ de commencer par une paire de 0 et de se terminer par un triplet de 1 est de $1/2^5 = 1/32$. Par conséquent, v_{20} est surprenante vis-à-vis de ce critère. De manière complètement générale, étant donnée une suite $x_1, \ldots, x_N$, la probabilité $P(X_1 \ldots X_N \neq x_1 \ldots x_N)$ est très proche de 1 lorsque N est grand, égale à $1 - 1/2^N$, et par conséquent n'importe quelle suite manque de vérifier au moins une propriété de $X_1, \ldots, X_N$ valable avec forte probabilité. Il y a pire, de manière générale, on peut remarquer le fait suivant : étant données deux suites de 0 et de 1 $x_1, \ldots, x_N$ et $y_1, \ldots, y_N$, et $\alpha \in [0, 1]$, le nombre de parties $A \subset \{0, 1\}^N$ telles que $P(X_1, \ldots, X_N \in A) = 1 - \alpha$ et $(x_1, \ldots, x_N) \in A$ est identique au nombre de parties $A \subset \{0, 1\}^N$ telles que $P(X_1, \ldots, X_N \in A) = 1 - \alpha$ et $(y_1, \ldots, y_N) \in A$. Par conséquent, on ne peut pas non plus chercher à distinguer parmi les suites de 0 et de 1 celles qui satisferaient un maximum de propriétés de $X_1, \ldots, X_N$ de celles qui n'en satisferaient qu'un petit nombre.

Essentiellement, on se heurte à nouveau au fait que deux suites quelconques de longueur N ont toujours exactement la même probabilité d'apparaître, à savoir $1/2^N$, et que rien ne permet de les distinguer de ce point de vue.

Cependant, parmi les propriétés mentionnées de $X_1, \ldots, X_{20}$ mentionnées ci-dessus, les deux premières semblent plus naturelles que la troisième et dernière : que dans une longue suite de 0 et de 1, la fréquence observée des 0 et des 1, ainsi que des paires successives de 0 ou de 1, soient voisines respectivement $1/2$ et de $1/4$ semble une exigence naturelle, tandis que la dernière propriété (ne pas commencer

par 00 et ne pas finir par 111) semble complètement ad hoc, calculée en fonction de v_{20}, et choisie exprès pour l'éliminer.

Il est possible de préciser ces différentes notions, et les travaux notamment de Von Mises, Ville, Kolmogorov, Chaitin et Martin-Löf ont permis de dégager des définitions relativement satisfaisantes de ce que peut signifier pour une suite (infinie) donnée le fait de correspondre à une réalisation d'une suite (infinie) de variables aléatoires i.i.d. équiprobables de 0 et de 1 (voir par exemple [K] pour une discussion générale, ou [LV] pour une description plus approfondie). Cependant ces approches ne sont pas extrêmement bien adaptées à notre contexte (ne serait-ce que parce que les procédés algorithmiques de génération de nombres pseudo-aléatoires n'ont aucune chance de satisfaire les définitions correspondantes), et, en tenant compte du fait que les suites produites par les générateurs de nombres pseudo-aléatoires ne sont pas des suites fixées, mais des suites rendues aléatoires par le choix de la graine, on peut parvenir à des définitions beaucoup plus satisfaisantes et d'une portée pratique plus significative.

Nous appellerons donc s_0 la graine, que nous supposerons choisie aléatoirement dans l'ensemble S selon une loi de probabilité bien définie, et $x_1, x_2, \ldots, x_N$ la suite des N premiers nombres produits par le générateur à partir de cette graine. Ainsi, $x_1, \ldots, x_N$ apparaît comme une suite de variables aléatoires sur $\{0, 1\}$. On voit immédiatement qu'il n'est pas raisonnable d'espérer que $x_1, \ldots, x_N$ soit une famille de variables aléatoires de même loi que $X_1, \ldots, X_N$ lorsque N est grand, puisqu'au mieux, s_0 contient de l'ordre de $\log(|S|)$ bits aléatoires (de quoi spécifier un élément de S), et par conséquent il en va de même de la suite $x_1, \ldots, x_N$, tandis que la suite $X_1, \ldots, X_N$ représente N bits indépendants et identiquement distribués équiprobables. Une autre manière de voir les choses est de dire que s_0 peut prendre au maximum $|S|$ valeurs distinctes, et donc que la suite $x_1, \ldots, x_N$ ne peut prendre que $|S|$ valeurs distinctes, contre 2^N pour la suite $X_1, \ldots, X_N$.

Cependant, ce qui nous intéresse réellement est que l'on ne constate pas de différence significative entre les résultats obtenus à l'aide d'une simulation utilisant notre suite $x_1, \ldots, x_N$ et une simulation qui serait effectuée à l'aide $X_1, \ldots, X_N$. Une définition informelle d'un générateur algorithmique satisfaisant pourrait donc être : **(D)** aucun algorithme doté de ressources raisonnables en temps d'exécution ne doit être capable de distinguer entre $x_1, \ldots, x_N$ et $X_1, \ldots, X_N$ avec une probabilité (relative à l'aléa contenu dans $x_1, \ldots, x_N$ par l'intermédiaire de s_0 et à l'aléa contenu dans

$X_1, \ldots, X_N$) significativement différente de 1/2.

On peut formuler une définition précise de ce que constituent des ressources croissant avec N de manière raisonnable et une probabilité significativement (eu égard à N) différente de 1/2, voir par exemple l'ouvrage de Knuth [K]. La restriction concernant le temps d'exécution dans la définition ci-dessus provient du fait que, si on laisse suffisament de temps à un algorithme, celui-ci, en essayant toutes les graines possibles, finira forcément, ou tout au moins très probablement, par distinguer $x_1, \ldots, x_N$ de $X_1, \ldots, X_N$, la probabilité pour qu'il existe une graine permettant d'engendrer $X_1, \ldots, X_N$ au moyen d'un algorithme donné étant extrêmement faible lorsque N est grand.

Ce type de définition semble bien adapté à notre objectif, qui est de simuler des modèles pour en étudier les propriétés. En effet, les quantités que l'on cherche à évaluer sont la plupart du temps les espérances de variables aléatoires définies dans le cadre du modèle, et leur évaluation par simulation repose sur un résultat de type loi des grands nombres, qui affirme que la quantité aléatoire (en général une moyenne empirique) que l'on calcule à partir des simulations possède une forte probabilité de se trouver à proximité de la quantité (tout ceci étant bien entendu exprimé de manière précise et quantitative) que l'on cherche à estimer. Une suite $x_1, \ldots, x_N$ vérifiant la propriété (D) ci-dessus est donc un bon candidat pour effectuer des simulations. En effet, une simulation constitue un algorithme tournant en un temps raisonnable (sans quoi on ne pourrait pas la mener à bien), et les résultats obtenus en utilisant la suite $x_1, \ldots, x_N$ en lieu et place des variables aléatoires $X_1, \ldots, X_N$ doivent nécessairement être voisins de la quantité que l'on cherche à évaluer avec une forte probabilité, sans quoi (D) serait contredite.

Un générateur algorithmique satisfaisant de nombres pseudo-aléatoires apparaît dans ce cadre comme un «amplificateur de hasard», prenant en entrée une petite quantité d'aléa, contenue dans la graine, pour la transformer en une longue suite indiscernable d'une véritable suite i.i.d. par des moyens raisonnables.

3 Critères de qualité de procédés de génération

Comment se comportent les procédés concrets de génération de nombres pseudo-aléatoires au vu de la définition (D) proposée dans la partie précédente ? Ces questions font encore l'objet de recherches, et il est envisageable que la situation évo-

lue rapidement et de manière imprévisible. D'une part, mentionnons l'existence de procédés de génération dont on peut prouver qu'ils satisfont (D) (une fois celle-ci convenablement précisée) sous des hypothèses, non-prouvées, quant à elles, de difficulté algorithmique de certains problèmes, tels que la factorisation de grands entiers. Ces procédés ne sont pas couramment utilisés en simulation, pour des raisons diverses, parmi lesquelles leur coût élevé de fonctionnement. D'autre part, la plupart des procédés de génération employés en simulation à l'heure actuelle ne satisfont pas (D), mais se révèlent suffisament efficaces et fiables en pratique pour que leur usage se trouve justifié, en respectant certaines précautions, qui visent à s'assurer que les imperfections du générateur n'affectent pas la validité des simulations menées, ou, autrement dit, que la structure particulière de dépendance existant entre les valeurs successives fournies par le générateur n'interfère pas (trop) avec la simulation menée. Ces procédés sont principalement validés au moyen de deux catégories de tests.

1. des tests dits théoriques, qui consistent à prouver mathématiquement que ces procédés possèdent un certain nombre de propriétés souhaitables ;
2. des tests empiriques, dans l'esprit de la définition (D), qui consistent à s'assurer que, dans un certain nombre de situations contrôlées, le comportement observé avec $x_1, \ldots, x_N$ ne diffère pas significativement du comportement que l'on observerait avec de $X_1, \ldots, X_N$ (il faut pour cela que ce dernier soit connu).

3.1 Manipulation de nombres réels

Avant de donner une description plus détaillée des divers types de tests utilisés pour valider les générateurs de nombres pseudo-aléatoires, rappelons que ceux-ci sont censés simuler des suites de variables aléatoire i.i.d. uniformes sur $[0, 1]$. Une première limitation de principe à cet objectif est que, même si l'on peut choisir la précision avec laquelle les nombres sont représentés dans la machine, on ne peut pas s'affranchir du fait que les ordinateurs ne manipulent les nombres réels qu'en précision finie. Par conséquent, le mieux que nous pourrions attendre serait que les nombres produits par le générateur représentent des variables aléatoires i.i.d. uniformes non pas sur $[0, 1]$ mais sur un ensemble de la forme

$$\left\{ \frac{0}{K}, \frac{1}{K}, \ldots, \frac{K}{K} \right\},$$

qui constitue une discrétisation de l'intervalle $[0, 1]$ d'autant plus fine que la valeur de K est élevée. Souvent, les générateurs sont modifiés de manière à ne jamais produire les valeurs extrêmes 0 et 1, qui pourraient produire des erreurs de calcul (par exemple des divisions par zéro) si le programme qui les utilise a été écrit sans précautions suffisantes.

3.2 Tests théoriques de validité

L'approche théorique consiste à *prouver mathématiquement* que le générateur possède un certain nombre de propriétés souhaitables, qui rapprochent les suites de nombres qu'il produit de véritables suites de nombres aléatoires. Les preuves mathématiques correspondantes sont souvent difficiles, basées sur des considérations avancées de théorie des nombres. Pour cette raison, on ne peut établir par cette approche que des propriétés relativement simples des générateurs... qui sont cependant fondamentales. Il ne serait pas raisonnable d'employer un générateur de nombres pseudo-aléatoires dont aucune analyse théorique convaincante ne viendrait justifier l'emploi, d'autant que jusqu'à présent, les performances théoriquement analysées ont toujours remarquablement bien prédit l'efficacité pratique des générateurs considérés.

3.2.1 Période

Une première remarque évidente est que toute suite décrite par un procédé algorithmique de génération de nombres pseudo-aléatoires tels que défini plus haut est ultimement **périodique** (car elle est construite à partir de l'itération d'une fonction définie sur et à valeurs dans un ensemble fini), et répète donc indéfiniment la même séquence de nombres à partir d'un certain moment. Dans la plupart des cas, la suite est exactement périodique, et la période ne dépend pas de la graine choisie (lorsque celle-ci est faite en respectant certaines recommandations, qui dépendent complètement du générateur utilisé). On ne peut donc en aucun cas s'attendre à ce que la suite obtenue mime convenablement une suite aléatoire lorsque l'on en extrait un nombre de termes supérieur, ou même comparable à la période du générateur. À titre d'exemple, dans certains cas, les imperfections du générateur peuvent être mises en évidence dès que l'on extrait un nombre de termes de l'ordre de la puissance $1/3$ de la période. Pour plus d'informations concernant le nombre maximal de termes qu'il est raisonnable d'extraire d'un générateur de nombres pseudo-aléatoires en rapport

avec sa période, voir [LE], [LE2], [LE3], ainsi que les références qui s'y trouvent. Une première condition minimale de qualité pour un générateur est donc de posséder une période élevée. Bien entendu, la notion de ce qu'est une «grande» période a évolué avec le développement de la puissance de calcul des ordinateurs, les machines actuelles se prêtant à des simulations d'envergure beaucoup plus importante que par le passé. Pour être considéré comme convenable aujourd'hui, un générateur se doit de posséder une période d'au moins 2^{100}.

Dans ces conditions, il est bien entendu impossible de calculer la période d'un générateur expérimentalement (en effectuant des appels répétés jusqu'à obtenir un élément de S égal à la graine), et une étude théorique du problème est incontournable.

La période des générateurs linéaires congruentiels simples décrits plus haut est au maximum égale à m, d'où en particulier la nécessité d'utiliser des congruences modulo de «grands» entiers m. Lorsque les paramètres vérifient des propriétés arithmétiques convenables (voir [K]), on peut effectivement obtenir des générateurs dont la période est égale à m. Par exemple, certaines implémentations de `rand` possède une période maximale de 2^{32}, `drand48` a pour période 2^{48}. Ces générateurs ne doivent donc plus être utilisés aujourd'hui. Bien que maximale (c'est-à-dire égale à m), leur période est trop petite. L'utilisation de générateurs multi-récursifs ou de générateurs linéaires congruentiels combinés permet, pour un coût comparable, d'obtenir des périodes beaucoup plus grandes, de l'ordre de m^k, en utilisant un espace S produit $S = S_1 \times \cdots \times S_k$, dans lequel f opère directement sur les coordonnées. Par exemple, le générateur MRG32k3a (voir [LE]) qui combine deux générateurs multi-récursifs d'ordre 3 possède une période d'environ 2^{191}. La même remarque vaut pour les générateurs digitaux, dont la période maximale de 2^k est atteinte sous réserve que les paramètres vérifient des propriétés arithmétiques convenables. Par exemple, Mersenne Twister possède une période de $2^{19937} - 1$. Notez que, dans les deux cas (MRG32k3a et Mersenne Twister) que nous venons de citer, la période des générateurs dépasse très largement la taille de l'ensemble $g(S)$, ce qui est rendu possible par le fait que l'espace d'états sur lequel opère la récurrence est plus gros que l'ensemble $g(S)$. Il est également possible d'analyser la période des générateurs non-linéaires. Ici encore, l'utilisation de générateurs combinés permet d'accroître considérablement la période à faible coût (voir [H] et [N]). Insistons sur le fait que la période des générateurs linéaires est particulièrement sensible au choix des paramètres : la plupart des combinaisons conduisent à des périodes déraisonnablement courtes, et il est indispensable

de se baser sur un critère théorique pour choisir ceux-ci. Plus généralement, le choix des paramètres (ou de la forme) d'un générateur de nombres pseudo-aléatoires ne doit certainement pas être laissé au hasard, sous peine de résultats catastrophiques (voir par exemple [K]).

3.2.2 Discrépance

Bien entendu, une période élevée ne suffit absolument pas à garantir la qualité d'un procédé de génération de nombres pseudo-aléatoires. Par exemple, le «générateur» donné par $f(s) = s + 1 \bmod m$ et $g(s) = s/m$ possède une période égale à m (et qui peut donc être extrêmement grande), mais paraît difficilement acceptable, du fait de la corrélation entre les valeurs successives qu'il produit ! On peut évidemment fabriquer des suites de très longue période, mais qui ne se répartissent pas du tout uniformément entre 0 et 1, et ne sont donc pas acceptables en tant que générateurs de nombres pseudo-uniformes entre 0 et 1. Posséder une période élevée est indispensable, mais cela n'est en rien suffisant. Globalement, il est également indispensable, en plus d'une étude de la période, de procéder à une analyse de la **répartition** des nombres produits par le générateur, en particulier pour évaluer les **corrélations** qui peuvent exister entre les valeurs successives de la suite. Considérons la liste des $t-$uplets :

$$\Psi_t = ((x_1, x_2, \ldots, x_t) \ : \ s_0 \in S) \, .$$

En admettant que le générateur soit de période maximale, et que la période ne dépende pas du choix de s_0 (autrement dit, que l'action de f sur S ne possède qu'une seule orbite), Ψ_t représente l'ensemble (comportant éventuellement des répétitions) des t-uplets de nombres produits par le générateur au cours d'une période complète.

En admettant que la graine s_0 soit choisie uniformément au hasard dans S, la suite des t premières valeurs produites par le générateur est alors un élément choisi uniformément dans la liste Ψ_t (qui peut comporter des répétitions). Idéalement, ces t valeurs devraient former un t-uplet distribué uniformément sur l'ensemble $[0, 1]^t$ (à une discrétisation près). Il est donc nécessaire que l'ensemble Ψ_t soit en un certain sens proche, ou encore représentatif, de l'ensemble $[0, 1]^t$, autrement dit, qu'il en constitue un maillage serré et homogène (serré pour que la discrétisation soit fine, homogène pour que les points soient répartis de manière équilibrée, en prenant en compte leur multiplicité), sans quoi l'uniformité supposée des nombres produits par

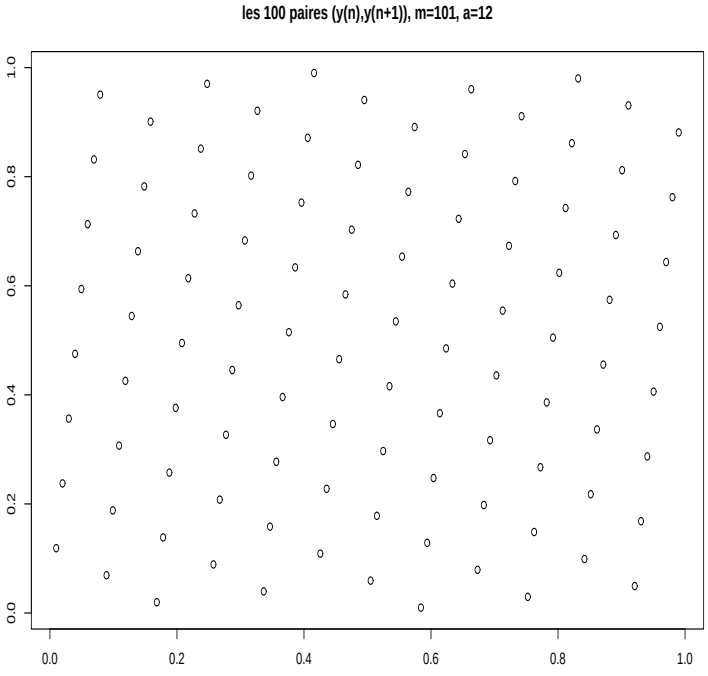
le générateur est visiblement prise en défaut.

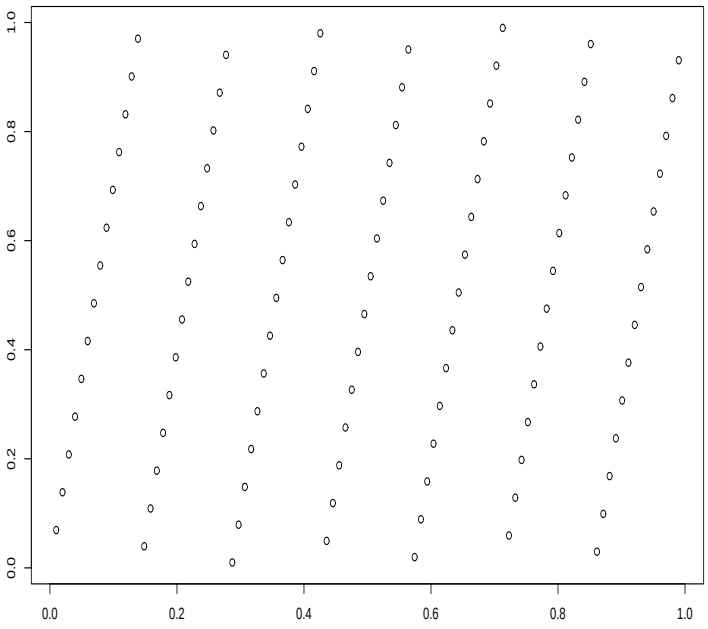
Dans le cas extrême du «générateur» décrit ci-dessus, l'examen de Ψ_2 fait immédiatement apparaître le problème, mais pas celui de Ψ_1. Plus généralement, il est indispensable d'étudier Ψ_t jusqu'à de grandes valeurs de t, la qualité du recrouvrement se dégradant globalement lorsque t croît.

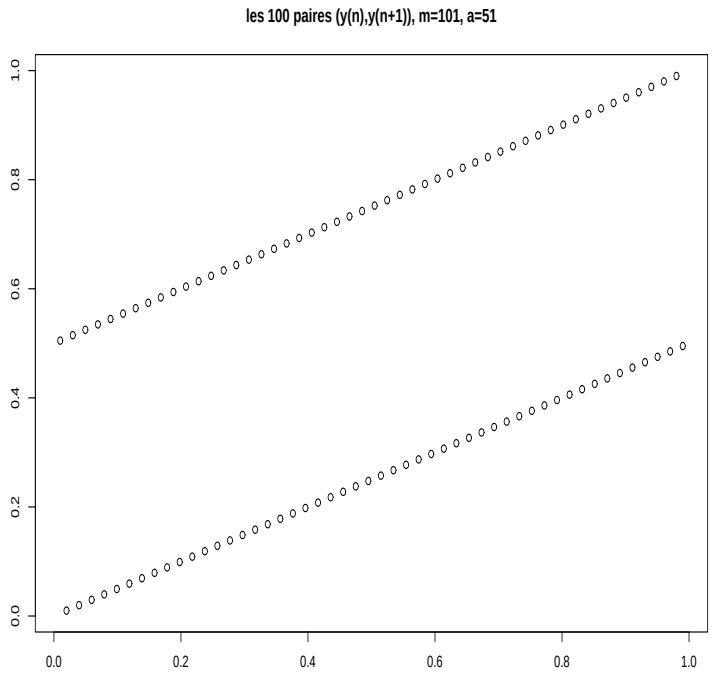
La **discrépance** du générateur fournit une mesure quantitative du caractère serré et homogène du recouvrement. Suivant le type de générateur considéré, on considère différentes définitions pratiques de la discrépance, de façon à pouvoir évaluer théoriquement sa valeur (il est bien entendu impossible de calculer numériquement la discrépance en épuisant une période complète du générateur). La discrépance permet de comparer entre elles les performances de générateurs différents, et également de donner une mesure absolue de la qualité des générateurs employés. On étudie également des discrépances correspondant au recouvrement par des t-uplets de valeurs non-consécutives (par exemple, de la forme $(x_{kp})_{1\leq k\leq m}$) produites par le générateur, de façon à étudier les corrélations à longue portée.

Les générateurs linéaires congruentiels présentent, du fait de la linéarité de f, une structure très particulière : l'ensemble Ψ_t possède une structure de réseau régulier. En particulier, si l'on représente les «points» de Ψ_t en dimension t, ceux-ci sont répartis sur un nombre limité d'hyperplans parallèles équidistants. La distance maximale d_t entre deux hyperplans parallèles mesure la finesse du recouvrement de $[0,1]^t$ par les suites de longueur t produites par le générateur. Plus d_t est grand, plus la largeur des «tranches» de $[0,1]^t$ désertées par le générateur est grande. Inversement, plus d_t est faible, plus la couverture de $[0,1]^t$ est fine et homogène. La valeur de d_t peut être évaluée théoriquement à partir des paramètres du générateur, et l'on compare habituellement celle-ci à des estimations de la valeur minimale d_t^* que pourrait prendre cette distance en choisissant un réseau régulier comportant le même nombre de points que Ψ_t (la valeur exacte de d_t^* n'est connue que pour $2 \leq t \leq 8$). C'est ce que l'on appelle le test spectral (voir [K]). (En dimension 1, il suffit que la période soit maximale, égale à m, pour que $d_1 = 1$, tous les entiers entre 0 et $m-1$ apparaissant une et une seule fois au cours d'une période.) Afin d'illustrer la structure en réseau des générateurs congruentiels linéaires, voici les représentations graphiques de Ψ_2, c'est-à-dire des paires successives, produites par le générateur (très simple) :

$$f(s) = as \bmod 101,\ g(s) = s/101,\ \text{pour } a = 12,\ a = 7 \text{ et } a = 51.$$


les 100 paires (y(n),y(n+1)), m=101, a=12
0.0
0.2
0.4
0.6
0.8
1.0


les 100 paires (y(n),y(n+1)), m=101, a=7
0.0
0.2
0.4
0.6
0.8
1.0



Une structure aussi régulière peut paraître discréditer d'emblée les générateurs linéaires congruentiels : une structure si régulière ne ressemble pas à ce que l'obtiendrait avec de véritables variables aléatoires indépendantes uniformes... Vraiment ? Un instant de réflexion montre que, avec des variables aléatoires indépendantes et uniformes tronquées à une certaine précision (ce qui est inévitable sur un ordinateur, qui représente les nombres en précision finie), on obtiendrait en toute dimension une structure tout aussi régulière, à condition de tirer suffisament de points. La différence porte surtout sur la finesse du recouvrement qu'il est possible d'atteindre, notamment en grande dimension (rappelons qu'un générateur peut générer au plus $|S|$ t-uplets distincts, quelle que soit la valeur de t). Une telle structure régulière «macroscopique» peut poser des problèmes, si cette elle interfère d'une manière ou d'une autre avec les propriétés spécifiques des suites de variables aléatoires i.i.d. uniformes exploitées implicitement par la simulation. Voir par exemple [G]. On tente parfois de faire disparaître cette structure en réseau en combinant le générateur en question à un (petit, pour obtenir une certaine rapidité) générateur non-linéaire.

Le dernier graphique ci-dessus illustre à quel point un mauvais choix des paramètres peut avoir des conséquences catastrophiques sur la structure du générateur,

même lorsque celui-ci est de période maximale (et relativement grande), et illustre la nécessité d'un choix minutieux des paramètres.

Voici les scores obtenus au test spectral par les générateurs linéaires congruentiels évoqués précédemment : le rapport d_t^*/d_t est indiqué pour $t = 2, \ldots, 8$.

`rand`	0,84	0,52	0,63	0,49	0,68	0,43	0,54
`drand48`	0,51	0,80	0,45	0,58	0,66	0,80	0,60
`RANDU`	0,93	0,012	0,059	0,16	0,29	0,45	0,62
MAPLE	0,75	0,74	0,65	0,73	0,63	0,56	0,56

Ces coefficients, qui mesurent donc le rapport entre la finesse de recouvrement produit par le générateur et la finesse optimale, semblent convenables, exception faite de `RANDU`, dont la structure de réseau en dimension 3 est catastrophique, quinze plans parallèles contenant l'intégralité des points. (Ce générateur et ses variantes ont néanmoins été utilisés pendant des dizaines d'années.)

L'utilisation de générateurs combinés et/ou multi-récursifs permet, pour des choix de paramètres convenables, d'améliorer significativement la structure de réseau des générateurs (voir [LE4]).

Les générateurs digitaux ou inversifs ne possèdent pas de structure en réseau comparable à celle des générateurs linéaires (mais possèdent d'autres types, moins apparents, de structures spécifiques, également susceptibles de fausser les simulations si celles-ci sont sensibles à ce type de structure), et, par conséquent, on ne peut pas mesurer la discrépance à l'aide de cette structure. Une autre manière de mesurer la qualité de la répartition de Ψ_t dans $[0,1]^t$ est d'évaluer la quantité D_t^*, définie comme le maximum, pris sur tous les pavés de la forme $R = [0, \alpha_1] \times \cdots \times [0, \alpha_t]$, de la différence

$$\left| \mathrm{Vol}(R) - \frac{\#(\Psi_t \cap R)}{\#\Psi_t} \right|.$$

Autrement dit, D_t^* mesure l'écart, sur tous les pavés R contenant l'origine, entre la fréquence d'apparition des points de Ψ_t dans R et une répartition parfaitement uniforme.

Ainsi définie, la discrépance des générateurs inversifs peut être évaluée théoriquement (voir [N]). Son étude fait apparaître d'excellentes propriétés de corrélation de ces générateurs, qui, contrairement au cas des générateurs congruentiels linéaires, sont remarquablement stables par rapport au choix des paramètres (il suffit que la

période soit maximale). De plus (voir [H]), ces propriétés sont conservées par combinaison de générateurs.

Pour les générateurs digitaux (en base 2, pour fixer les idées), on peut évaluer la discrépance en mesurant l'équirépartition des bits de poids élevé dans Ψ_t. Considérons les l bits de poids le plus élevé dans les suites de longueur $(x_i, \cdots, x_{i+t-1})$ produites par le générateur sur une période. Il y a donc 2^{lt} possibilités pour ces bits. Si chacune de ces possibilités apparaît exactement avec la fréquence 2^{-lt} dans Ψ_t, le générateur est dit t-équidistribué jusqu'à la précision de l bits : chaque motif constitué de t mots de l bits apparaît alors exactement, sur une période complète, avec la même fréquence que chaque autre, ce qui correspond à une répartition uniforme discrétisée. Par exemple, le générateur Mersenne Twister est 623-équidistribué jusqu'à la précision de 32 bits. Si l'on est moins exigeant concernant la finesse du maillage, on peut atteindre une équidistribution en dimension plus grande. Pour plus de précisions, voir [MN] et [LE5].

3.2.3 Flux parallèles

Il est fréquent d'avoir à simuler plusieurs systèmes indépendants évoluant en parallèle, et utilisant chacun leur propre flux de nombres pseudo-aléatoires. On peut envisager plusieurs manières de produire k flux distincts (et supposément indépendants) à l'aide de générateurs algorithmiques de nombres pseudo-aléatoires :

1. séparer la suite $(x_n)_{n\geq 1}$ produite par un même générateur en k suites, en retenant un terme sur k de la suite initiale : $(x_{kn})_{n\geq 1}, (x_{kn+1})_{n\geq 1}, \ldots, (x_{kn+k-1})_{n\geq 1}$,
2. choisir k graines différentes pour un même générateur (éloignées dans la séquence x_n) et générer les k séquences correspondantes,
3. utiliser k générateurs différents.

Avec les générateurs linéaires congruentiels simples, chacun de ces trois procédés peuvent conduire à des résultats catastrophiques, même si le ou les générateurs employés possèdent individuellement de bonnes propriétés de structure. Ainsi, concernant le point 1, des corrélations importantes peuvent apparaître entre les différentes suites extraites, ce que l'on mesure en étudiant la structure de réseau de l'ensemble $(x_{kn}, x_{kn+1}, \ldots, x_{kn+k-1})$. Par exemple, avec le générateur `rand`, la suite que l'on obtient en retenant un terme sur 25 vérifie : $d_2/d_2^* = 0,0822$. On peut également

observer avec les générateurs linéaires congruentiels des corrélations entre les k séquences suggérées dans le point 2, ou entre les suites obtenues par plusieurs procédés différents, matérialisées par de très mauvaises structures de réseau.

L'utilisation de générateurs combinés et/ou multi-récursifs permet, dans certains cas, d'obtenir des propriétés de structure satisfaisantes pour la génération de k séquences en parallèle à partir de k graines différentes (voir [LE2] et les références qui s'y trouvent).

Les générateurs inversifs explicites conservent en général leurs bonnes propriétés de corrélation lorsqu'on les utilise pour produire des séquences en parallèle.

Un certain nombre de générateurs fournissent directement des procédures spéciales («splitting facility», «multi-stream facility») permettant de générer plusieurs séquences en parallèle dans des conditions en principe contrôlées.

On doit au moins retenir des remarques précédentes le fait que les suites produites par des générateurs ne peuvent pas être traitées inconditionnellement comme de véritables suites de variables aléatoires i.i.d. Certaines opérations que l'on peut mener sans changer le comportement de ces dernières peuvent considérablement affecter celui des suites obtenues au moyen un générateur de nombres qui ne sont que pseudo-aléatoires.

Dressons un premier bilan pratique de ces résultats théoriques :

- les générateurs linéaires congruentiels simples ne doivent plus être utilisés : leur propriétés de corrélation sont généralement trop médiocres, et leurs propriétés vis-à-vis de la parallélisation catastrophiques,
- les générateurs linéaires congruentiels multi-récursifs et/ou combinés ont en général de meilleures propriétés de corrélation, pour un coût comparable, et peuvent dans certains cas être utilisés pour la parallélisation,
- cependant, les propriétés de ces générateurs sont peu robustes vis-à-vis du choix des paramètres (ce qui n'est pas grave lorsque l'implémentation choisie, rarement effectuée par l'utilisateur, emploie un jeu de paramètres convenables),
- tous les générateurs linéaires possèdent une structure particulière en réseau, qui, dans certaines applications, peut fausser gravement les résultats (voir par exemple [G]),
- les générateurs inversifs possèdent de bonnes propriétés de corrélation, stables vis-à-vis du choix des paramètres, et se prêtent mieux à la parallélisation, cependant, ils sont significativement plus lents que les générateurs linéaires,

– les générateurs digitaux tels que Mersenne Twister fournissent des générateurs rapides possédant d'excellentes propriétés de corrélation.

On peut noter le fait qu'il n'est souvent pas nécessaire pour assurer la validité d'une simulation qu'il y ait indépendance mutuelle, même approximative, de la totalité des valeurs de la suite produite par le générateur, des propriétés d'indépendance plus locales (telles que l'indépendance deux-à-deux des valeurs consécutives, ou encore l'indépendance entre des valeurs suffisament éloignées) pouvant s'avérer suffisantes, et ceci peut fournir une certaine justification (en plus des tests empiriques décrits ci-après) au fait que l'on n'hésite pas à extraire d'un générateur des suites de nombres pseudo-aléatoires d'une longueur bien supérieure aux valeurs de t pour lesquelles le recouvrement par Ψ_t est contrôlé et satisfaisant. Cependant, il est difficile, en toute généralité, de s'assurer que la validité d'une simulation est insensible à tel ou tel type de dépendance existant entre les nombres fournis par le générateur, ou que, en d'autres termes, il n'y a pas d'interférences particulières entre la structure du générateur et celle du modèle simulé.

Une limitation essentielle des propriétés théoriques étudiées ci-dessus est qu'elles portent sur l'ensemble de tous les t-uplets que peut fournir le générateur. L'uniformité du choix de la graine dans S conditionne donc, a priori, le fait que les t-uplets produits par le générateur puissent être considérés comme tirés uniformément dans la liste Ψ_t, et donc la portée pratique d'une discrépance évaluée mathématiquement à une faible valeur. Qui plus est, la qualité du maillage fourni par l'ensemble Ψ_t dégénère lorsque t devient grand[2], et il est de toute façon difficile de prévoir comment la structure particulière de l'ensemble Ψ_t (par exemple la structure en réseau des générateurs linéaires congruentiels), que t soit grand ou non, peut affecter la validité des simulations menées avec le générateur. Il est donc d'usage (et indispensable) de soumettre également les générateurs à des tests empiriques, permettant de vérifier expérimentalement leur comportement sur diverses simulations.

[2]On peut comprendre ceci intuitivement : plus on connaît de termes de la suite, plus on acquiert d'informations sur le terme suivant, et donc plus la relation existant entre les termes successifs doit se manifester. Ou encore, noter le fait que le générateur ne peut produire qu'au maximum $|S|$ $t-$uplets différents, ce qui exclut le fait d'obtenir un maillage de $[0,1]^t$ de bonne qualité lorsque t est grand.

3.2.4 Tests empiriques de validité

Le principe est celui des tests statistiques : on cherche à tester sur la base des données observées $x_1, \ldots, x_N$ la validité de l'hypothèse (H0) : « $x_1, \ldots, x_N$ est issue de la réalisation d'une suite de variables aléatoires indépendantes de loi uniforme sur [0,1]» en calculant une statistique $T(x_1, \ldots, x_N)$ dont la distribution est, sous (H0), connue à l'avance.

Une zone dite critique (ou encore «de rejet») de valeurs de T est définie, telle que T possède une forte probabilité de se trouver hors de cette zone si (H0) est vraie, et simultanément une probabilité aussi forte que possible (la puissance du test) de se trouver dedans si (H0) n'est pas vérifiée. Compte-tenu de l'extrême généralité des alternatives à (H0), il est assez difficile de définir des zones de rejet susceptibles de discriminer efficacement (H0) de toutes ces alternatives, et, en général, la zone critique est définie en fonction du type d'alternative à laquelle on souhaite confronter (H0). On testera en général différemment la dépendance à courte portée (entre des valeurs fournies par le générateur lors d'appels voisins), des dépendances à longue portée telles que la présence de phénomènes cycliques, la non-uniformité de la répartition,...

Soulignons que le contexte des tests empiriques de générateurs n'est pas exactement celui des tests statistiques usuels : la quantité de données disponibles n'est pas vraiment limitée (on peut en obtenir autant que l'ordinateur peut en produire en un temps raisonnable, et cela fait beaucoup, beaucoup,...), et les alternatives à l'hypothèse testée sont de nature très diverses.

Les tests que l'on peut pratiquer ne sont nullement restreints aux tests classiques d'adéquation à la loi uniforme. De fait, toute quantité aléatoire simulée à l'aide de $x_1, \ldots, x_N$ et dont la loi (sous l'hypothèse que $x_1, \ldots, x_N$ forme une suite de variables aléatoires indépendantes de loi uniforme sur [0,1]) est connue exactement ou approximativement, fournit un moyen de tester la qualité du générateur. Autrement dit, toute simulation d'un modèle dont les propriétés sont connues à l'avance fournit des tests empiriques, la zone de rejet restant à définir en fonction du type d'écart à (H0) que l'on souhaite tester et de l'influence prévisible d'un tel écart sur la localisation des valeurs prises par la statistique que l'on calcule.

Avant de donner quelques exemples pratiques de tels tests, (ré)insistons sur plusieurs points. Tout d'abord, le fait qu'un générateur passe avec succès un grand

nombre de tests empiriques ne signifie absolument pas qu'il se comportera bien vis-à-vis de l'application spécifique que l'on compte en faire (même si notre confiance dans le générateur en est accrue). Idéalement, il faudrait être en mesure de tester la qualité des résultats obtenus avec le générateur *pour l'application qui nous intéresse*, mais, bien entendu, on ne connaît pas en général complètement les propriétés du modèle que l'on simule (sans quoi on ne le simulerait pas), ce qui limite la possibilité d'effectuer ce genre de test. La sensibilité d'une simulation aux imperfections du générateur dépendant, entre autres, de la manière spécifique dont est utilisé le générateur dans celle-ci, il est a priori difficile de déduire du succès constaté d'un générateur dans une simulation «voisine» de celle envisagée le fait qu'il se comportera de manière satisfaisante dans celle-ci.

Cela étant, vérifier au cours d'une simulation que les propriétés connues du modèle sont bien satisfaites (au sens statistique du terme !) est important, voire indispensable (ne serait-ce que pour vérifier l'absence d'erreur dans le code de simulation !).

Plus généralement, l'approche empirique permet de discréditer les générateurs qui se révèlent incapables de passer avec succès des tests relativement simples. Ainsi, un générateur de nombres pseudo-aléatoires est «présumé innocent» jusqu'à ce qu'il soit éliminé par un test. L'utilisation de tests variés permet de détecter des défauts de différentes natures.

Ceci dit, il est souvent souhaitable d'utiliser plusieurs générateurs de nature différente dans une même simulation afin de vérifier les résultats obtenus, dans l'espoir que, si la simulation est sensible à un défaut particulier du générateur (par exemple la structure en réseau), ce défaut ne sera pas partagé par des générateurs appartenant à des familles différentes.

3.2.5 Tests classiques d'adéquation

Les tests usuels d'ajustement d'un échantillon à une loi donnée (voir cours de statistique) peuvent être employés. Par exemple, pour $t \geq 1$, on peut considérer l'échantillon de m points de $[0,1]^t$:

$$(x_{t(i-1)}, x_{t(i-1)+1}, \ldots, x_{ti-1})_{i=1,\ldots,\ell}$$

et tester son adéquation avec une loi uniforme sur $[0,1]^t$. Décrivons le principe du test du χ^2. Découpons $[0,1]^t$ en K cellules de tailles égales, et appelons χ_j le nombre

de points de l'échantillon se trouvant dans la cellule numéro j, alors la statistique

$$\chi^2 = \sum_{i=1}^{K} \frac{(\chi_j - \ell/K)^2}{\ell/K}$$

est, sous l'hypothèse (H0), distribuée selon une loi du χ^2 à $K-1$ degrés de liberté lorsque N tend vers l'infini à K fixé, et l'on peut appliquer le test du χ^2, qui consiste à rejeter (H0) lorsque la valeur de χ^2 dépasse un certain seuil. Cependant, dans notre contexte, il n'est pas forcément astucieux de choisir une zone de rejet de la forme $[a, +\infty)$. En effet, une simulation de dé qui donnerait, sur 6000 lancers, exactement 1000 lancers pour chaque chiffre serait hautement suspecte de ne pas être basée sur des tirages aléatoires et indépendants, mais (H0) serait néanmoins acceptée par un test du χ^2 classique (qui est parfaitement raisonnable lorsque l'on suppose a priori que les observations proviennent d'un échantillon i.i.d., (H0) se contentant de préciser cette loi). Il peut donc être intéressant de rejeter (H0)lorsque le χ^2 est trop grand, ou trop petit.

Une autre remarque est que les tests pratiqués sur les générateurs en extrayant un nombre de données relativement limité ont tendance à être plus puissants que ceux qui exploitent de très longues suites de nombres, dans lesquels les défauts du générateur finissent par être lissés. Knuth [K] recommande plutôt de calculer un grand nombre d'exemplaires d'une même statistique T à partir d'échantillons relativement petits, et de pratiquer un test d'adéquation entre la loi théorique de la statistique et les données collectées.

En dimension 1, on peut par exemple également appliquer le test de Kolmogorov-Smirnov (voir [K]).

Autres tests

Il existe une multitude de tests reposant sur telle ou telle statistique particulière (voir par exemple [K]). À l'heure actuelle, on peut trouver des batteries de tests déjà programmées, comme par exemple la «Diehard Battery of tests for randomness», développée par G. Marsaglia et disponible à l'adresse

`http://www.stat.fsu.edu/pub/diehard/`

Décrivons deux exemples simples (voir [LE]).

Le test de collision

On partitionne l'intervalle $[0,1]^t$ en K cellules de tailles égales. On utilise ensuite le générateur pour produire successivement ℓ points de $[0,1]^t$ «uniformément distribués» $(y_{t(i-1)}, y_{t(i-1)+1}, \ldots, y_{ti-1}), i = 1, \ldots, \ell$, et l'on appelle C le nombre total de fois où un point tombe dans un cellule contenant déjà au moins un point. Lorsque ℓ est grand, le rapport $\ell^2/2K$ restant de l'ordre de l'unité, la loi de de la variable aléatoire C est connue avec une bonne précision : il s'agit d'une loi de Poisson de paramètre $\ell^2/2K$. On peut ainsi procéder à un test statistique portant sur l'uniformité des nombres produits par le générateur.

Le test d'espacement des anniversaires

On répète la même procédure que dans le test précédent, mais la quantité que l'on évalue n'est pas la même. Numérotons les K cellules de 1 à K. Les cellules représentent ici les différents jours de l'année (il y a donc K jours dans l'année !), et le fait que le point $(y_{t(i-1)}, y_{t(i-1)+1}, \ldots, y_{ti-1})$ tombe dans la cellule numérotée L signifie que l'individu numéro i a son anniversaire le jour L. Ordonnons les ℓ dates d'anniversaire : $I_1 \leq I_2 \leq \cdots \leq I_\ell$, et faisons ensuite la liste des espacements $S_j = I_{j+1} - I_j$ entre deux anniversaires consécutifs. On évalue le nombre d'indices j pour lesquels $S_{j+1} = S_j$ (on compte le nombre de fois où la durée qui sépare le $j+1$-ème anniversaire du $j-$ème est égale à celle qui sépare le $j+2$-ème anniversaire du $j+1-$ème). Lorsque ℓ est grand, ℓ^3/K restant de l'ordre de l'unité, la loi de Y est également connue avec une grande précision, il s'agit également d'une loi de Poisson, de paramètre $\ell^3/4K$.

Notez bien que la différence entre les tests théoriques et empiriques ne réside pas seulement dans le fait d'avoir recours à l'expérience pour étudier une propriété du générateur, mais également sur le fait que seulement une petite partie (très indérieure à la période) des nombres produits par le générateur est explorée et testée.

L'objet n'est pas de donner ici les résultats détaillés obtenus à l'aide de ces tests, nous renvoyons pour cela aux références bibliographiques. On constate empiriquement que, pour un générateur et un test fixés, lorsque la taille de l'échantillon (N) dépasse une valeur critique, le générateur échoue systématiquement. Déterminer, pour une classe de générateurs donnés, l'ordre de cette valeur critique par rapport à la période, est une question importante en pratique, puisque l'on obtient ainsi une estimation du nombre d'appels successifs au générateur que l'on peut raisonnablement effectuer (vis-à-vis de la propriété testée).

Pour citer L'Écuyer ([LE]), «l'expérience montre que les générateurs possédant de très longues périodes, une bonne structure de Ψ_t, et basés sur des récurrences qui ne sont pas trop simplistes, passent la plupart des tests raisonnables, tandis que les générateurs possédant des périodes courtes ou une mauvaise structure sont aisément éliminés par les tests statistiques standards.»

Ainsi, les générateurs linéaires combinés, les générateurs inversifs et les générateurs digitaux considérés comme les meilleurs passent tous ces tests avec des niveaux de confiance considérés (aujourd'hui) comme raisonnables. En revanche, certains générateurs équipant les logiciels commerciaux, même à visée scientifique, échouent grossièrement.

Conclusion

Le but de ce cours n'est pas principalement de recommander l'utilisation de tel ou tel générateur particulier qui, dans quelques années, sera rendu obsolète par l'accroissement de la puissance de calcul des ordinateurs et l'évolution des critères de qualité, ou des avancées théoriques importantes, et corrigera certains défauts de ses prédécesseurs tout en en présentant de nouveaux non-encore découverts. Quoiqu'il en soit, il n'existe pas à l'heure actuelle un unique «bon» générateur dont l'utilisation systématique pourrait être recommandée inconditionnellement, la sensibilité d'une simulation aux défauts particuliers d'un procédé de génération étant difficile à apprécier a priori. Il est important d'adopter la bonne attitude vis-à-vis de l'utilisation de générateurs de nombres pseudo-aéatoires pour la simulation. Voici quelques recommandations de portée générale :

– ne jamais utiliser un générateur sans s'être assuré au préalable qu'il possède des propriétés théoriques convenables, et qu'il a passé avec succès diverses batteries de tests empiriques. Il est important de retenir que la plupart des générateurs proposés aujourd'hui par défaut sur nombre de logiciels commerciaux ou gratuits, à visée scientifique ou générale, sont **grossièrement inadaptés** à un usage scientifique sérieux, ou tout au moins obsolètes, et ne doivent plus être utilisés. Les bons générateurs (ceux considérés comme tels à l'heure actuelle) sont aisément disponibles, gratuits, et guère plus exigeants en mémoire et temps d'exécution que les générateurs obsolètes qu'ils devraient systématiquement remplacer.

– se tenir au courant (internet, publications spécialisées), car le domaine est susceptible d'évolutions, voire de bouleversements rapides (avancées théoriques, techniques,...) De nouveaux générateurs, de nouvelles procédures de test, seront disponibles.

Ensuite, voici une synthèse des recommandations tirées de l'exposé précédent :

– vérifier que les propriétés connues du modèle que l'on simule sont satisfaites,
– utiliser plusieurs générateurs différents pour contrôler les résultats obtenus, il n'y a pas «un» unique bon générateur qu'il suffirait d'employer systématiquement pour ne pas rencontrer de problèmes,
– lorsque les résulats semblent en désaccord avec d'autres informations, se rappeler que le générateur de nombres pseudo-aléatoires peut être en cause (même si la plupart du temps, c'est le code de simulation qui est faux).
– Préciser quel générateur a été employé : le générateur employé fait partie de la spécification des conditions expérimentales !
– à l'heure actuelle, il existe de bons générateurs (aux normes d'aujourd'hui) dans chacune des trois catégories : générateurs linéaires multi-récursifs combinés, inversifs, et digitaux. Consultez les références citées pour les descriptions détaillées, des exemples d'implémentation, et la liste des propriétés connues et/ou testées.

Bibliographie

[FLW] A. Ferrenberg, D. Landau et Y. Wong. Monte Carlo simulations : hidden errors from "good" random number generators. Phys. Rev. Lett. 69, 3382-3384.

[G] B. Gärtner. Pitfalls in Computing with Pseudorandom Determinants. Proc. 16th Annual ACM Symposium on Computational Geometry (SCG), 148–155, 2000.

[H] P. Hellekalek. Good random number generators are (not so) easy to find. Mathematics and Computers in Simulation 46 :485–505, 1998.

[K] D. Knuth. The Art of Computer Programming. Vol. 2. Seminumerical Algorithms. Third edition. Addison-Wesley, 1998.

[LE] P. L'Ecuyer. Software for uniform random number generation : distinguishing the good and the bad, Proceedings of the 2001 Winter Simulation Conference, IEEE Press, Dec. 2001, 95–105.

[LE2] P. L'Ecuyer. Random Number Generation, chapter 2 of the Handbook of

Computational Statistics, J. E. Gentle, W. Haerdle, and Y. Mori, eds., Springer-Verlag, 2004, 35–70.

[LE3] P. L'Ecuyer. Random Number Generation, chapter 3 of Elsevier Handbooks in Operations Research and Management Science : Simulation, S. G. Henderson and B. L. Nelson, eds., Elsevier Science, circa 2005.

[LE4] P. L'Ecuyer. Combined Multiple Recursive Random Number Generators. Operations Research, 44 :816-822, 1996.

[LE5] P. L'Ecuyer and F. Panneton. Fast Random Number Generators Based on Linear Recurrences Modulo 2 : Overview and Comparison. Proceedings of the 2005 Winter Simulation Conference.

[LV] M. Li and P. Vitanyi. An Introduction to Kolmogorov Complexity and its Applications. Springer–Verlag, 1997.

[MZ] G. Marsaglia et A. Zaman. The KISS generator. Tech. Report, Dept of Statistics, University of Florida, 1993.

[MN] M. Matsumoto et T. Nishimura. Mersenne twister : A 623-dimensionally equidistributed uniform pseudo-randim number generator. ACM Tansactions on Modelling and Computer Simulation, 1998. Disponible à l'adresse `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html`

[Ni] H. Niederreiter, New developments in uniform pseudorandom number and vector generation. In Monte Carlo and quasi-Monte Carlo methods in scientific computing, ed. H. Niederreiter and P.J.-S. Shiue. Berlin : Springer-Verlag, 1995.