Devoir Maison n° 1 : Générateurs numériques de nombres aléatoires.

Ce Devoir Maison s'inspire de très intéressants documents de Jean Bérard (Univ. Lyon 1), que l'on pourra consulter sur sa page pour plus d'information,

`http://math.univ-lyon1.fr/~jberard/enseignement-cours.html`

ainsi que d'une feuille de TD de Pierre Hyvernat (Univ. de Savoie)

`http://www.lama.univ-savoie.fr/~hyvernat/enseignement.php`

Générateurs numériques aléatoires.

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

Pas très aléatoire, n'est-ce pas. Mais en même temps, que demander à un ordinateur ?

Un premier type de générateur aléatoire : les congruences linéaires.

Pour un fonction `rand` de ce type qui renverrai un nombre réel choisi uniformément dans $[0,1]$, la procédure est la suivante :

1. on travaille sur un entier entre 1 et 2^m-1 pour m grand (par exemple ($m = 31, 47, ...$).
2. On choisit aussi une fonction de $\{1, 2, \ldots, 2^m-1\}$ dans lui même.
3. On initialise ensuite le générateur aléatoire lors de son premier appel par le choix d'une *graîne* $s_0 \in \{1, 2, \ldots, 2^m-1\}$. Pour choisir cette graîne on utilise par exemple le temps donné par l'horloge de l'ordinateur et surtout les chiffres des secondes et millisecondes (le chiffre de l'année est assez peu aléatoire).
4. A chaque appelle de la fonction `rand`, on remplace s_{k-1} par $s_k = f(s_{k-1})$.

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

5. On ramène enfin l'entier s_k de $\{1, 2, \ldots, 2^m - 1\}$ dans $[0, 1]$, avec par exemple l'application
$$s_k \mapsto x_k = \frac{s_k}{2^m}.$$

Question 1 : Programmer un tel générateur aléatoire avec comme fonction f

$$f(k) = a \times k + b \pmod{2^m}$$

il faudra prévoir de pouvoir changer a et b, et m. Pour l'initialisation de la graîne, vous pouvez la choisir vous même, ou utiliser le temps machine lors du premier appel....

Question 2 : Expliquer pourquoi un tel générateur est toujours périodique (c'est-à-dire que la suite va toujours finir par se répèter).
Calculer par exemple les chiffres obtenus et la période lorsque $m = 24$, $a = 6$, $b = 2$ et $s_0 = 5$. La période est le nombre d'itérations nécessaires pour voir apparaître une répétition.

Question 4 : La page de manuel de la fonction rand contient (suivant les systèmes) la note suivante :

```
However, on older rand() implementations, and on current
implementations on different systems, the lower-order bits
are much less random than the higher-order bits.
```

Regardez ce qui se passe sur le bit de poids faible (c'est-à-dire regardez ce qui se passe uniquement modulo 2), et expliquer cette remarque.

Second type : registre à décalage et rétroaction.

Les générateurs de type "Linear feedback shift register" ("Registre à décalage à rétroaction linéaire") permettent de générer des suites de bits ressemblant à des suites aléatoires. Les plus simples sont de type Fibonacci où chaque nouveau bit est obtenu en faisant le XOR de quelques bits précédents. Par exemple :

$$b_n = b_{n-1} \oplus b_{n-4} \oplus b_{n-6} \oplus b_{n-30}$$

Dans cet exemple, on peut stocker les 30 bits précédents dans un nombre entier (32 bits). Le bit généré est calculé, puis on fait un décalage à gauche et on ajoute le bit généré dans le bit de poids faible.

Question 1. Un tel générateur est forcément périodique (c-à-d qu'il recommence à faire exactement la même chose au bout d'un certain temps). Pourquoi ? Quelle est la période maximale si on utilise 30 bits.

Question 2. Calculez à la main, les bits générés par

$$b_n = b_{n-3} \oplus b_{n-4}$$

si on commence avec un état qui ne contient que des 1. Quelle est la période du générateur ? Même question en commençant avec un état qui ne contient que des 0.

Question 2. Écrire une fonction qui travaille sur 8 bits, et qui utilise la formule :

$$b_n = b_{n-3} \oplus b_{n-5} \oplus b_{n-8}$$

et qui renvoie ensuite une valeur entre 0 et 1 grâce à la formule

$$x_n = \frac{b_n 2^7 + b_{n-1} 2^6 + b_{n-2} 2^5 + b_{n-3} 2^4 + b_{n-4} 2^3 + b_{n-5} 2^2 + b_{n-6} 2 + b_{n-7}}{2^8}$$

Précisément, cette fonction doit :

— Travailler sur un nombre de 8 bits, initialisé à la main ou à l'aide de l'horloge de l'ordinateur,
— à chaque appel de la fonction, la formule ci-dessus (pour b_n) est appliquée, et on décale vers la droite les valeurs du bits en ajoutant la valeur calculée à gauche,
— la fonction renvoie enfin la variable x_n donnée par la formule ci-dessus.

Prévoir de pouvoir modifier la formule donnant b_n pour la section suivante.

Qualité d'un générateur pseudo-aléatoire.

Supposons qu'on génére une suite de nombres réels entre 0 et 1. Évidemment il est difficile de comparer une telle suite déterministe (sauf pour le choix de la graine) avec une suite aléatoire. Pour cela, il faut d'abord choisir un critère de qualité. Mais, ce n'est pas si simple, il y a beaucoup de choses à vérifier pour être aléatoire, et le problème est de trouver des critères simples et pertients.
Un premier critère assez naturel, et de tester l'indépendance de deux termes consécutifs générés. Pour cela on représente graphiquement classiquement les points (x_k, x_{k+1}) dans le carré $[0,1]^2$. Un générateur sera bon (pour ce critère) s'il remplira assez rapidement tout le carré.

Question 1. Pour les deux générateurs étudiés dans les section précédentes (dans les deux question 2 des sections précédentes), et pour le gńérateur du language que vous utilisez :
— Construisez un vecteur$(x_1, \ldots, x_{1000})$ contenant 1000 appel consécutif de la fonction `rand`.
— Tracer dans un graphique les points (x_k, x_{k+1}) pour $k = 1, \ldots, 999$.

Question 2. Qu'obtenez vous ? Commentez les résultats obtenus. Quel semble être le meilleur générateur ?

Question 3. Modifiez les paramètres a et b de votre premier générateur. Voyez-vous des différences dans la façon de remplir le carré ?
Même question avec votre second générateur en modifiant la formule donannt b_n en fonction des 8 bits précédents.

Pour les réponses : les fichiers (html ou pdf ou notebook ou ...) sont acceptés, et même souhaités pour les graphiques. Sinon, vous pouvez toujours en faire une représentation sommaire dans votre copie (au sens de Devoir rendu sur papier) ou pourquoi pas m'envoyer des captures d'écran par mail.

Travail en groupe (de 2) possible.