Génération de nombres aléatoires

1 – Éléments pédagogiques

1.1 Objectifs

Bien que la connaissance théorique des outils soit indispensable pour résoudre les problèmes, la mise en oeuvre pratique de ces techniques dans un contexte proche de la réalité est tout aussi important.

L'objectif de cette suite d'exercices pratiques est de mettre les méthodes de programmation et les concepts du C++ présentés en cours à travers une application de simulation. Le travail proposé est une sensibilisation à la mise en place d'architectures logicielles dans le cadre des mathématiques appliquées.

Le travail est à réaliser en binôme uniquement. Vous devez programmer en C++ en utilisant les standards présents sur les compilateurs disponibles sur les stations de travail de l'école.

Votre travail devra faire l'objet d'un compte-rendu précis, clair et concis mettant en avant les résultats obtenus ainsi que vos observations sur les méthodes utilisées.

1.2 Livrables

Vous devez rédiger un compte-rendu dans lequel vous répondrez à toutes les questions de l'énoncé, expliciterez les méthodes employées, présenterez et commenterez les résultats obtenus. Il ny a qu'un seul compte-rendu à rendre par binôme.

Notez que ce document ne constitue que la base de ce qui vous est demandé : soyez critique par compte-rendu à vos résultats, proposez dautres idées, solutions ou tests. Chaque partie qui vous ai proposé peut faire l'objet d'extensions ou d'approfondissements.

Le compte-rendu ne devra pas excéder 15 pages et ne devra pas contenir de programmes. L'ensemble des fichiers de votre travail devra être soumis sous Teide.

Le compte-rendu devra être au format PDF.

La qualité de la rédaction, de la synthèse, de l'analyse des résultats obtenus sont des critères importants dans l'évaluation finale.

La seconde partie de votre rendu est l'ensemble des sources qui vous a servi à produire le travail attendu. Vous êtes libre d'organiser le code source de la manière que vous souhaitez.

La lisibilité du code et la pertinence des commentaires seront pris en compte dans l'évaluation.

1.3 Organisation du travail

Ce travail est à réaliser sur l'ensemble des 6 séances de mise en pratique du cours. Le travail demandé est conséquent. Chaque séance s'appuie sur le travail réalisé pendant les séances précédentes. Soyez donc rigoureux dans la réalisation du travail et n'attendez pas la dernière séance pour commenter votre travail.

N'hésitez pas à demander des conseils, des précisions ou à poser vos questions par mail aux enseignants.

1.4 Remise des livrables

Le rendu est à soumettre pour le Vendredi 11 Mai 2018 à 20h00 sous Teide. Votre livrable devra contenir

- le compte rendu au format PDF.
- le code que vous avez écrit.
- les images et animations mettant en valeur votre travail.

Un livrable soumis hors délai se verra attribuer la note 0.

2 – Modélisation

Cette partie présente différentes techniques permettant de générer des nombres pseudo-aléatoires. L'objectif de ces techniques est de pallier à certaines faiblesses du générateur proposé dans la bibliothèque standard.

Une fois ces modèles présentés, nous verrons comment utiliser les techniques du C++ facilitant la manipulation des différents générateurs développés dans un autre code.

2.1 Problématique

La fonction rand définie dans la bibliothèque standard dépend de l'implémentation. Le standard spécifie un certain nombre de propriétés et propose une implémentation possible sans pour autant fournir l'ensemble des détails.

La conséquence pour les utilisateurs, et en particulier les applications en calcul scientifique, est l'absence de consistance de comportement entre les compilateurs. L'exécution du même code sur des plateformes différentes risque de produire des séquences de nombres différentes qui vont dépendre de l'implémentation de `rand()`. Ces différences n'empêcheront pas les simulations de Monte-Carlo de converger vers le bon résultat, mais réduiront la capacité à vérifier le programme.

Le second inconvénient est que `rand()` n'est pas prévisible. La graine des générateurs aléatoires est une variable globale, ce qui signifie que les appels à la fonction `rand()` dans différentes parties du programme peuvent s'affecter mutuellement.

Un troisième inconvénient est que `rand()` est prévisible. Avec un seul nombre généré, il est possible de reproduire l'intégralité des nombres pseudo-aléatoire suivants.

Le dernier problème est la période sur laquelle les nombres sont générées. Dans l'implémentation standard, la période est de 2^{31} nombres entiers aléatoires soit un peu moins de 2.2×10^9 nombres ce qui est faible pour les simulations numériques.

Nous allons donc développer un ensemble de classes qui implémentent des générateurs aléatoires utilisables dans notre contexte.

2.2 Générateurs aléatoires

Un générateur de nombre pseudo-aléatoire de bonne qualité doit satisfaire de bonnes propriétés théoriques et statistiques. Les propriétés théoriques sont souvent difficiles à obtenir. Pour un générateur aléatoire, il est préférable d'avoir une période assez longue, une autocorrélation faible et une corrélation importante entre différentes séries de tirage. En général un bon générateur aléatoire est utilisé pour estimer une quantité pour laquelle les probabilités proposent une réponse exacte. La mesure de l'aléatoire d'un générateur peut par exemple être la différence entre la valeur exacte et la valeur calculée.

Dans ce qui suit, nous allons proposer différents générateurs de nombres pseudo-aléatoires. Nous utiliserons des tests statistiques afin d'identifier leur forces et faiblesses.

2.3 Park-Miller

Le premier générateur que nous allons considérer s'appelle Park-Miller. Il fait parti de la famille des générateur congruents linéaires. Les nombres pseudo aléatoires sont définis par une suite récurrente de la forme

$$X_{n+1} = (aX_n + c) \mod m \tag{1}$$

avec X la suite de nombres pseudo-aléatoires, le module $0 < m$, le multiplicateur $0 < a < m$, l'incrément $0 < n < m$ et la graine $0 < X_0 < m$.

Dans le cas où $c = 0$, nous obtenons un générateur de Park-Miller. Cela ajoute des contraintes sur les paramètres du générateur : m doit être un nombre entier très grand, a est une racine primitive modulo m et X_0 la graine du générateur tel que m et X_0 soient co-premier entre eux.

Il existe de nombreux générateurs de cette forme. Le générateur le plus utilisé a pour paramètres $m = 2147483647$ et $a = 16807$.

Il s'agit ensuite de trouver une graine qui satisfasse la condition mentionnée et de réutiliser le générateur de manière itérative afin de générer d'autres nombres.

Afin de construire une graine qui respecte les conditions demandées, on utilise l'algorithme de Schrage. La graine se calcule de la manière suivante

$$X_{n+1} = \left(a(X_n - \lfloor \frac{X_n}{q} \rfloor * q) - r \lfloor \frac{X_n}{q} \rfloor\right) \mod m \tag{2}$$

avec les entiers m et a définis précédemment, $q = 127773$ et $r = 2836$.

Il existe d'autres combinaisons possible.

2.4 Xorshift

Les générateurs aléatoires `Xorshift` font partie des générateurs de type Registre à décalage à rétroaction linéaire (LFSR) qui permettent une implémentation très efficace. Chaque nombre aléatoire est générée à partir du précédent en appliquant une opération de type ou suivi d'une translation de bit. Ce sont deux opérations très efficaces sur les ordinateurs modernes. Les générateurs `Xorshift` s'écrivent sous la forme

Algorithme 1 : Algorithme de Xorshift

Data : (unsigned 64-bit) $x \neq 0$
Result : x
begin
  $x \leftarrow x \wedge (x \ll a_1)$;
  $x \leftarrow x \wedge (x \gg a_2)$;
  $x \leftarrow x \wedge (x \ll a_3)$;
end

Comme avec tous les générateurs aléatoires, il est nécessaire de choisir avec minutie les paramètres a_1, a_2 et a_3. Les paramètres suivant sont connus pour offrir de bonnes propriétés statistiques

a_1	21	20	17	11	14	30	21	21	23
a_2	35	41	31	29	29	35	37	43	41
a_3	4	5	8	14	11	13	4	4	18

TABLE 1 : Paramètres de Xorshift

3 – Éléments de conception

Afin de pouvoir de réaliser une solution logicielle efficace, nous allons développer de manière explicite une grande partie des objets mathématiques explicités dans la section précédente.

Cette section sera complétée au fur et à mesure du déroulement des TPs.

TP 1 : Le premier objet que nous allons devons écrire est celui qui va contenir l'ensemble des données de hauteur. Comme mentionné dans la construction du modèle, nous allons manipuler des données spatiales en dimension 2.

La tentation est donc d'écrire une classe `Array2D` qui serve de conteneur. Pour des raisons d'efficacité d'accès et d'alignement mémoire, il est préférable de travail avec des tableaux mono dimensionnelles. La première classe que nous allons écrire est donc une classe `Dvector`. Ce développement fera l'objet du premier TP.

TP 2 : Il s'agit d'enrichir la classe `Dvector` en ajoutant notamment la notion d'opérateur `C++` qui permettra de manipuler les objets de type `Dvector` comme des types intrinsèques.

Le système de `Makefile` sera remplacé par un système de génération automatique de `Makefile`, à savoir `cmake`.

TP 3 : Les notions importantes commenceront à apparaître avec l'utilisation de l'héritage et du polymorphisme que permettrons de créer les modèles mathématiques de génération des nombres aléatoires.

La gestion des exceptions sera également mise en place afin de pouvoir gérer les situations imprévues.