Simulation et Optimisation – TP2

# Introduction

Dans ce travail pratique, il nous est demandé de mettre en œuvre et de comparer plusieurs méthodes de Monte-Carlo :

1. Méthode d’intégration par échantillonnage uniforme,
2. Méthode d’intégration par échantillonnage préférentiel,
3. Méthode d’intégration par échantillonnage uniforme avec variable de contrôle.

Ces méthodes seront utilisées dans le but de déterminer l’aire de la fonction

Sur l’intervalle [0,15].

Dans un premier temps, nous testerons l’implémentation des différentes méthodes en générant un certain nombre de données et en analysant les résultats obtenus. Une fois ceci fait, nous passerons à la comparaison des performances de chacune de ces méthodes en utilisant deux approches :

* Comparaison des temps d’exécution requis pour l’obtention d’un intervalle de confiance d’une largeur ne dépassant pas une valeur fixée,
* Comparaison des largeurs des intervalles de confiance obtenus après un temps de calcul fixé.

# Notes préalables

* Le code a été compilé en C++11 en utilisant mingw 4.8.1.
* Les mesures ont été effectuées avec un processeur Intel®Core™ i7-2760QM CPU @ 2.40GHz.

# Description de l’implémentation

* Le code a été réalisé en C++11.
* Les générateurs de variables aléatoires ont été repris du premier travail pratique.
* Les points de la fonction affine par morceau à utiliser dans les méthodes 2 et 3 sont créés dans la classe Stats (reprise du premier TP et complétée). Nous reparlerons du choix de création de ces points plus tard.

L’architecture du code est la suivante :

* Les méthodes de Monte-Carlo sont représentées par une classe MonteCarloMethod, qui met à disposition plusieurs manières différentes de générer des échantillons : taille de l’échantillon fixé (*sampleWithSize*), échantillonnage jusqu’à une largeur d’intervalle de confiance maximale donnée (*sampleWithMaxWidth*) et échantillonnage jusqu’à un temps minimum donné *sampleWithMinTime*). En plus de leurs paramètres respectifs, les deux dernières prennent un paramètre *step*, représentant le nombre de valeurs à générer avant de revérifier les conditions d’arrêt.
* Les trois méthodes à implémenter dans ce travail pratique (échantillonnage uniforme. Préférentiel et uniforme avec variable de contrôle) sont représentées par des classes héritant de la classe MonteCarloMethod.
* La classe MonteCarloMethod garde en interne les différentes statistiques. Il est donc plus lisible et plus pratique d’effectuer divers calculs dans différentes fonctions sans devoir passer toutes les statistiques d’une fonction à l’autre.
* Pour la méthode d’échantillonnage uniforme avec variable de contrôle, une méthode *setSamplingSize* est mise à disposition afin de changer la taille « M » du « petit échantillon » à générer dans la première phase de la méthode. Avoir la taille du petit échantillon séparée des paramètres des trois méthodes *sample* (au lieu de *sampleWithSize(M,N)*, par exemple) permet de respecter la signature de ces dernières dans la classe MonteCarloMethod.

Le schéma UML de l’architecture (très simplifié) représentant les explications ci-dessus est le suivant:



Par souci de simplicité et de clarté, aucun des membres et méthodes privés n’ont été affichés et *sampleWithSize*, *sampleWithMaxWidth* et *sampleWithMinTime* n’ont pas été affichés dans les sous-classes de MonteCarloMethod.

## Choix de subdivision de l’intervalle d’intégration

L’intervalle d’intégration est subdivisé en sous-intervalles de largeurs égales. Autrement dit, lorsque nous voulons créer une fonction affine par morceau comprenant un certain nombre de points *P* à partir de notre fonction dont on calculer l’aire entre deux bornes *a* et *b*, on aura pour chaque sous-intervalle une largeur de . Pour notre fonction, nous obtenons par exemple, pour 15 points :

|  |  |
| --- | --- |
| x | f(x) |
| 0 | 102.831259 |
| 1.07142857 | 24.9472802 |
| 2.14285714 | 8.0826406 |
| 3.21428571 | 22.2737654 |
| 4.28571429 | 33.9732808 |
| 5.35714286 | 26.4666183 |
| 6.42857143 | 51.5447711 |
| 7.5 | 100.891916 |
| 8.57142857 | 75.3294967 |
| 9.64285714 | 16.3370198 |
| 10.7142857 | 23.4235276 |
| 11.7857143 | 6.51684889 |
| 12.8571429 | 9.54768305 |
| 13.9285714 | 90.1969334 |
| 15 | 89.6486364 |

Pour la suite de l’analyse, cette fonction par morceaux de 15 points sera utilisée pour les méthodes 2 et 3.

# Validation de l’implémentation

Avant tout, mentionnons que la valeur de l’aire de cette fonction peut être calculée au préalable (au moyen de wolfram alpha, par exemple), et vaut **601.971**. Nous nous y réfèrerons par **G**.

Sauf mention du contraire, les méthodes 1, 2 et 3 dans les tables et graphiques suivants font respectivement référence à l’échantillonnage uniforme, préférentiel et uniforme avec variable de contrôle.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Méthode | | N |  | IC |  |  | Temps [s] |
| 1 | 100000 | | 601.0275 | [598.240,603.815] | 449.67011 | 5.57417 | 0.015 |
| 1000000 | | 602.15816 | [601.275,603.042] | 450.71304 | 1.7668 | 0.172 |
| 10000000 | | 601.97038 | [601.691,602.250] | 450.88017 | 0.55892 | 1.797 |
| 2 | 100000 | | 601.86313 | [601.370,602.356] | 79.55771 | 0.98621 | 0.031 |
| 1000000 | | 601.86825 | [601.713,602.023] | 79.20599 | 0.31049 | 0.25 |
| 10000000 | | 601.93762 | [601.888,601.987] | 79.31759 | 0.09832 | 2.497 |
| 3 | 100000 | | 602.09849 | [601.549,602.648] | 88.61924 | 1.09854 | 0.015 |
| 1000000 | | 601.87316 | [601.698,602.048] | 89.1828 | 0.3496 | 0.187 |
| 10000000 | | 601.97101 | [601.916,602.026] | 88.97629 | 0.1103 | 2.047 |

Table 1 : génération d’échantillons de tailles pour les 3 méthodes.

Nous pouvons tout d’abord constater sur la table 1 que, pour chacune des méthodes, l’estimateur de l’aire () semble converger vers la valeur théorique (**G**) lorsque l’on augmente la taille de l’échantillon (**N**). Nous remarquons que **G** se trouve dans chacun des intervalles de confiance (**IC**).

Nous remarquons que le produit est bien stable (tend approximativement vers la même valeur) pour une méthode donnée, qu’importe le **N**. La largeur de l’IC () diminue et le temps augmente lorsque **N** augmente.

Tout ceci (principalement la convergence de ) peut nous amener à penser que l’implémentation des différentes méthodes est a priori correcte.

# Analyse des résultats obtenus

Pour tous les résultats qui suivent, nous prendrons un

La première approche de comparaison des performances consiste à comparer les temps de calculs nécessaires à l’obtention d’un IC d’une largeur ne dépassant pas une largeur fixée.