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 une approche de 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 morceaux 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 morceaux comprenant un certain nombre de points *P* à partir de notre fonction dont on veut 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 pour la méthode d’échantillonnage uniforme avec variable de contrôle une taille d’échantillon *M* de 10000 (l’échantillon étant utile pour trouver le coefficient utilisé par la suite lors de cette méthode).

L’approche de comparaison des performances que nous utiliserons ici consiste à comparer les temps de calculs nécessaires à l’obtention d’un IC d’une largeur ne dépassant pas une largeur fixée.

Voici ci-après les différents tableaux de mesures obtenus selon les différentes méthodes.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Temps min [s] | N |  | IC |  |  | Temps [s] |
| 1 | 5900000 | 601.78811 | [601.424,602.152] | 450.64632 | 0.72727 | 1 |
| 2 | 11600000 | 602.03329 | [601.774,602.293] | 450.74111 | 0.51878 | 2.016 |
| 4 | 23600000 | 602.0821 | [601.900,602.264] | 450.86187 | 0.36381 | 4.004 |
| 8 | 48000000 | 601.84245 | [601.715,601.970] | 450.78704 | 0.25506 | 8.008 |
| 16 | 95000000 | 602.03278 | [601.942,602.123] | 450.84561 | 0.18132 | 16.002 |
| 32 | 191800000 | 601.97847 | [601.915,602.042] | 450.81739 | 0.1276 | 32.002 |
| 64 | 380200000 | 601.99911 | [601.954,602.044] | 450.82847 | 0.09063 | 64.012 |
| 128 | 712300000 | 601.98141 | [601.948,602.015] | 450.81263 | 0.06621 | 128.014 |
| 256 | 1435400000 | 601.96767 | [601.944,601.991] | 450.8291 | 0.04665 | 256.012 |
| 512 | 3028600000 | 601.9682 | [601.952,601.984] | 450.82485 | 0.03211 | 512.012 |
| 1024 | 6106500000 | 601.97381 | [601.963,601.985] | 450.82516 | 0.02262 | 1024.015 |

Tableau 2 : mesures obtenues pour l’échantillonnage uniforme

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Temps min [s] | N |  | IC |  |  | Temps [s] |
| 1 | 3600000 | 601.95608 | [601.874,602.038] | 79.30804 | 0.16385 | 1 |
| 2 | 7300000 | 601.98993 | [601.932,602.047] | 79.2974 | 0.11505 | 2.016 |
| 4 | 14900000 | 601.96597 | [601.926,602.006] | 79.2751 | 0.08051 | 4 |
| 8 | 30400000 | 601.97972 | [601.952,602.008] | 79.2915 | 0.05637 | 8.001 |
| 16 | 61300000 | 601.96757 | [601.948,601.987] | 79.29188 | 0.0397 | 16.011 |
| 32 | 121600000 | 601.96613 | [601.952,601.980] | 79.29016 | 0.02819 | 32.011 |
| 64 | 241500000 | 601.97398 | [601.964,601.984] | 79.28433 | 0.02 | 64.002 |
| 128 | 480500000 | 601.97079 | [601.964,601.978] | 79.29256 | 0.01418 | 128.006 |
| 256 | 971800000 | 601.97064 | [601.966,601.976] | 79.29196 | 0.00997 | 256.045 |
| 512 | 1871300000 | 601.97294 | [601.969,601.977] | 79.29049 | 0.00719 | 512.005 |
| 1024 | 3614900000 | 601.97083 | [601.968,601.973] | 79.29093 | 0.00517 | 1024.015 |

Tableau 3 : mesures obtenues pour l’échantillonnage préférentiel

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Temps min [s] | N |  | IC |  |  | Temps [s] |
| 1 | 4610000 | 601.96771 | [601.886,602.049] | 89.04094 | 0.16256 | 1.02 |
| 2 | 8810000 | 601.97264 | [601.914,602.031] | 88.97818 | 0.11751 | 2.02 |
| 4 | 17210000 | 601.94721 | [601.905,601.989] | 89.04082 | 0.08414 | 4.008 |
| 8 | 34810000 | 601.98283 | [601.953,602.012] | 89.01051 | 0.05914 | 8.001 |
| 16 | 69910000 | 601.9516 | [601.931,601.972] | 89.03619 | 0.04174 | 16.003 |
| 32 | 135510000 | 601.97177 | [601.957,601.987] | 89.01703 | 0.02998 | 32.006 |
| 64 | 282310000 | 601.96949 | [601.959,601.980] | 89.01301 | 0.02077 | 64.003 |
| 128 | 574410000 | 601.9697 | [601.962,601.977] | 88.99723 | 0.01456 | 128.007 |
| 256 | 1148710000 | 601.97506 | [601.970,601.980] | 88.99557 | 0.01029 | 256.018 |
| 512 | 2317310000 | 601.97005 | [601.966,601.974] | 88.99452 | 0.00725 | 512.015 |
| 1024 | 4595510000 | 601.97029 | [601.968,601.973] | 88.9946 | 0.00515 | 1024.013 |

Tableau 4 : mesures obtenues pour l’échantillonnage uniforme avec variable de contrôle

Graphique 5 : représentation des trois méthodes

En observant le graphique (notons l’échelle logarithmique pour l’axe des ordonnées), nous pouvons constater que la méthode d’échantillonnage uniforme est la moins performante, alors que les deux autres méthodes semblent être aussi performantes l’une que l’autre. En effet, les graphes de ces dernières sont pratiquement confondus.

Ceci peut s’expliquer par le fait que la première méthode utilise une approche relativement simple (il s’agit d’une uniforme, après tout), alors que les deux dernières utilisent une fonction affine par morceaux (qui utilise 15 points, au lieu de 2 pour l’uniforme), afin de converger plus rapidement vers **G**.

Nous pouvons également noter que, bien que les deux dernières méthodes convergent aussi vite l’une que l’autre, il faut, d’après les mesures, plus de valeurs générées (un N plus grand) pour que la troisième méthode converge. Une génération est donc effectuée plus rapidement que dans la deuxième méthode, ce qui est logique car la troisième utilise un générateur uniforme alors que la deuxième un générateur de fonction inverse, vu au premier travail pratique.

# Conclusion

Nous avons pu voir que les deux dernières méthodes étaient apparemment aussi performantes l’une que l’autre et convergeaient bien plus rapidement que la première méthode. Il faudra donc favoriser une de ces méthodes lors d’un usage futur.