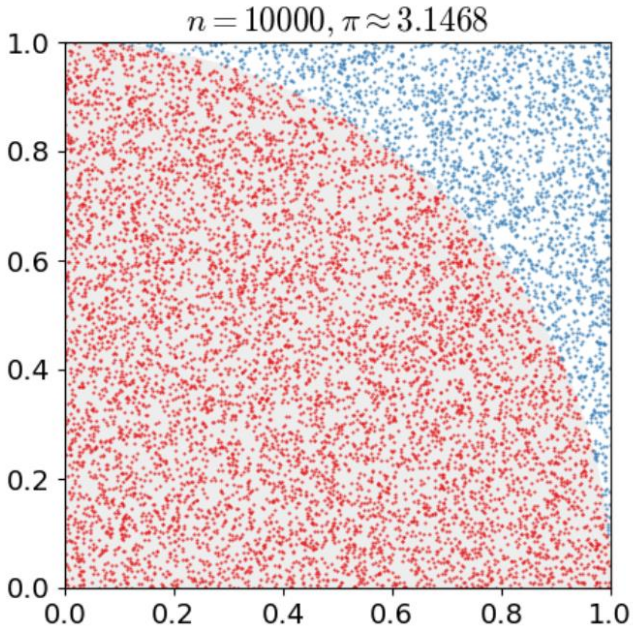
Simulation

TP Numéro 3 : Monte Carlo



Ao XIE

Responsable : David HILL

Date : 05 nove. 2022

Campus des Cézeaux, 1 rue de la Chébarde, TSA 60125, 63178 Aubière CEDEX

Introduction

Pour les algorithmes aléatoires, il existe deux algorithmes courants, l'algorithme de Monte Carlo et l'algorithme de Las Vegas. Parmi ceux-ci, l'algorithme de Monte Carlo renverra certainement un résultat, mais il n'y a aucune garantie qu'il soit optimal. L'algorithme de Las Vegas, en revanche, a la garantie de retourner un résultat optimal une fois qu'il l'a fait. Par conséquent, afin d'obtenir toujours un résultat plus proche du Π, nous utilisons l'algorithme de Monte Carlo dans cette expérience.

L'idée de l'algorithme de Monte Carlo est dérivée de l'aiguille de Buffon[1]. Il existe trois applications principales des méthodes de Monte Carlo[2] : l'optimisation, l'intégration numérique et les distributions de probabilité. Sawilowsky énumère les caractéristiques d'une simulation de Monte Carlo de haute qualité[3] :

- Le générateur de nombres (pseudo-aléatoires) présente certaines caractéristiques (par exemple, une longue "période" avant la répétition de la séquence)
- Le générateur de nombres (pseudo-aléatoires) produit des valeurs qui passent les tests de caractère aléatoire
- Il y a suffisamment d'échantillons pour garantir des résultats précis
- La technique d'échantillonnage appropriée est utilisée
- L'algorithme utilisé est valable pour ce qui est modélisé
- Il simule le phénomène en question.

Pour garantir l'obtention d'un algorithme de Monte Carlo de haute qualité, nous avons utilisé dans cette expérience l'algorithme de Mersenne Twister[4] pour générer des nombres pseudo-aléatoires. Ensuite, pour obtenir une valeur plus précise, nous avons écrit quatre fonctions différentes pour obtenir Π avec une précision différente. Ces quatre fonctions et les résultats des tests seront expliqués dans l'article.

Table des matières

Table des figures

I. Fonctions de développement

Dans cette section, nous allons illustrer et analyser algorithmiquement les quatre fonctions écrites : simPi, avgPi, calculRange et MonteCarlo.

i. La Procédure simPi

La fonction simPi permet d'utilise un simple algorithme de Monte Carlo pour générer des nombres aléatoires compris entre zéro et un à l'aide d'un générateur de nombres pseudo-aléatoires MT de haute qualité, puis calcule le nombre d'ensembles aléatoires générés qui se situent à l'intérieur du cercle et le rapport du nombre généré à utiliser pour calculer la circonférence du cercle. L'entrée de cette fonction est le nombre de matrices pseudo-aléatoires générées, et la sortie est la valeur finale de PI obtenue. L'algorithme exact est présenté dans la figure 1.

Algorithm 1 simPi()

Input: $TimesOfCalcul$

Output: Pi

```
1: CountIn ← 0
2: CountOut ← 0
3: for i → TimesOfCalcul do
4:     x ← random
5:     y ← random
6:     CountOut++
7:     if x^2 + y^2 ≤ 1 then
8:         CountIn++
9:     end if
10:    Pi ← (4 * CountIn)/CountOut
11: end for
```

Figure 1 La procédure simPi()

ii. La Procédure avgPi

La fonction avgPi permet d'utilise la fonction simPi à plusieurs reprises pour obtenir plusieurs valeurs de Pi, puis en fait la moyenne pour obtenir une valeur plus précise. L'entrée de cette fonction est le nombre de fois où la fonction simPi est calculée, et l'entrée de la fonction simPi est définie par une variable de définition de macro-TIME. La sortie de la fonction est également la valeur de Pi, mais elle est relativement plus précise. L'algorithme spécifique est présenté à la figure 2.

Algorithm 2 avgPi()

Input: $TimesOfCalcul$

Output: Pi

1: $avgPi \leftarrow 0$
2: $sumPi \leftarrow 0$
3: **for** $i \rightarrow TimesOfCalcul$ **do**
4: $\quad avgPi \leftarrow simPi()$
5: $\quad sumPi \leftarrow sumPi + avgPi$
6: **end for**
7: $Pi \leftarrow sumPi/TimesOfCalcul$

Figure 2 La procédure avgPi()

iii. La Procédure calculRange

La fonction calculRange permet de génère un intervalle de confiance final de quatre-vingt-quinze pour cent sur la base d'un algorithme de confiance. L'algorithme spécifique est présenté à la figure 3.

Algorithm 3 calculRange()

Input: $TimesOfCalcul$

Output: $ConfidenceInterval$

1: $avgPi \leftarrow 0$
2: $sumPi \leftarrow 0$
3: $T \Leftarrow TimesOfCalcul$
4: **for** $i \rightarrow TimesOfCalcul$ **do**
5: $\quad avgPi \leftarrow simPi()$
6: $\quad sumPi \leftarrow sumPi + avgPi$
7: **end for**
8: **for** $i \rightarrow TimesOfCalcul$ **do**
9: $\quad S \leftarrow S + (avgPi - (sumPi/TimeOfCalcul))^2$
10: **end for**
11: $S \leftarrow S/(TimesOfCalcul - 1)$
12: $S \leftarrow \sqrt{S}$
13: $ConfidenceInterval \leftarrow S * T$

Figure 3 La procédure calculRange()

iv. La Procédure MonteCarlo

Cette fonction combine les trois fonctions ci-dessus, en utilisant l'algorithme d'acceptation-rejet. La fonction utilise d'abord la fonction avgPi pour générer une valeur pour la circonférence obtenue en faisant la moyenne de plusieurs calculs, puis elle détermine le compromis entre la valeur réelle de la circonférence et l'intervalle de confiance, et lorsqu'un certain nombre de circonférences sont obtenues, la moyenne est calculée pour obtenir une circonférence plus précise. L'algorithme de la fonction est illustré à la figure 4.

Algorithm 4 $MonteCarlo()$

Input: $TimesOfCalcul$

Output: Pi

1: $valAvgPi \Leftarrow 0$
2: $sumPi \leftarrow 0$
3: $count \leftarrow 0$
4: **while** $count \leq TimesOfCalcul$ **do**
5: $valAvgPi \leftarrow avgPi(TimesOfCalcul)$
6: **if** (**then**$valAvgPi \leftarrow RealPi \pm calculRange(TimesOfCalcul)$)
7: $sumPi \leftarrow sumPi + valAvgPi$
8: $count++$
9: **end if**
10: **end while**
11: $Pi \leftarrow sumPi/count$

Figure 4 La procédure MonteCarlo()

II. Etudes de cas

Dans cette section, nous testons les quatre fonctions écrites pour obtenir des valeurs de circonférence avec différents degrés de précision.

i. Le premier problème

Dans cette sous-section, nous testons la première fonction simPi, qui utilise directement la méthode de Monte Carlo, et nous la testons un nombre différent de fois pour obtenir les résultats présentés dans la Figure 5.

```
/* -------------- Test of question ONE -------------- */
Result of 1000 times is 3.124000
Result of 1000000 times is 3.144720
Result of 1000000000 times is 3.141541
```

Figure 5 Résulta du premier problème

ii. Le deuxième problème

Dans cette sous-section, nous testons la deuxième fonction avgPi. Nous prenons les résultats obtenus en utilisant 40 fonctions simPi, mais en raison des limites de la puissance de calcul de l'ordinateur, chaque fonction simPi n'effectue que 1 000 000 d'opérations pour obtenir le résultat. Par conséquent, les résultats obtenus par cette fonction et cet algorithme sont seulement comparés à ceux produits par simPi(1,000,000) et on peut voir que cette fonction et cet algorithme obtiennent une valeur de circonférence plus précise. Les résultats obtenus sont présentés dans la figure 6.

```
 /* -------------- Test of question TWO -------------- */
Result of question two is 3.141862
Absolte error is -0.000269
Relative error is 0.999914
```

Figure 6 Résulta du deuxième problème

iii. Le troisième problème

Dans cette sous-section, nous testons la deuxième fonction, la fonction MonteCarlo. Cette fonction utilise la fonction calculRange pour produire un intervalle de confiance avec un niveau de confiance de quatre-vingt-quinze pour cent, puis appelle la fonction avgPi pour produire la valeur de la circonférence du cercle. La fonction collecte les valeurs qui se situent dans l'intervalle de confiance, et lorsqu'elle a collecté quarante circonférences, elle en fait la moyenne pour obtenir une circonférence plus précise. Le résultat obtenu en exécutant cette fonction est illustré à la figure 7.

```
/* --------------- Test of question THREE --------------- */
Result of question three is 3.141550
Absolte error is 0.000042
Relative error is 1.000013
```

Figure 7 Résulta du troisième problème

Conclusion

Dans ce travail, nous avons implémenté un algorithme de Monte Carlo pour calculer la circonférence d'un cercle en utilisant le langage C. Une comparaison des résultats montre que nous avons obtenu une circonférence de 3.144720 en utilisant la méthode de Monte Carlo, avec une moyenne de 3.141862 à partir de calculs multiples, et un résultat de 3.141550 en utilisant les intervalles de confiance, ce qui nous donne une circonférence plus précise en utilisant des calculs multiples pour obtenir une moyenne, et en utilisant également les intervalles de confiance

Il convient de noter qu'au cours de cet exercice, nous avons identifié deux problèmes qui doivent être résolus. Tout d'abord, lorsque nous utilisons la barre de progression dans la sortie, le calcul devient très lent. Deuxièmement, nous avons utilisé la valeur réelle de la circonférence comme critère de calcul après avoir obtenu l'intervalle de confiance, et comme le but de ce travail est de calculer une circonférence plus précise en utilisant la méthode de Monte Carlo, nous pensons que cette partie du travail peut encore être améliorée.

Références bibliographiques

[1] A. royale des sciences (France), *Histoire de l'Académie royale des sciences*. 1735. [En ligne]. Disponible sur: https://books.google.fr/books?id=GOAEAAAAQAAJ

[2] D. P. Kroese, T. Brereton, T. Taimre, et Z. I. Botev, « Why the Monte Carlo method is so important today », *WIREs Comput. Stat.*, vol. 6, nº 6, p. 386-392, nov. 2014, doi: 10.1002/wics.1314.

[3] S. S. Sawilowsky, « You Think You've Got Trivials? », *J. Mod. Appl. Stat. Methods*, vol. 2, nº 1, p. 218-225, mai 2003, doi: 10.22237/jmasm/1051748460.

[4] M. Matsumoto et T. Nishimura, « Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator », *ACM Trans. Model. Comput. Simul.*, vol. 8, nº 1, p. 3-30, janv. 1998, doi: 10.1145/272991.272995.