Approximation de π avec la méthode de Monte-Carlo

Python - TP n°2

Novembre 2017

Au cours de ce premier TP, nous allons estimer la valeur de Pi en utilisant la méthode de Monte-Carlo (Metropolis and Ulam, 1949).

Pour ce TP, des modules Python contenant les prototypes des fonctions à coder, ainsi que des tests unitaires permettant de tester lesdites fonctions seront fournis. Il vous sera demandé de vous servir de ces modules et de tester systématiquement votre code.

Ce TP est constitué de 3 parties visant à implanter chaque étape de la méthode et de 2 parties bonus (à ne traiter que si les 3 premières sont terminées) permettant la visualisation de vos résultats et le calcul de l'erreur de la méthode. Afin d'assurer le bon déroulement de ce TP, les questions devront être traitées dans l'ordre.

Idée générale de la méthode

En plaçant des points aléatoirement dans un domaine d'aire connue, lorsque le nombre de points tend vers l'infini, la proportion de points tombés dans un sous-domaine permet de déterminer son aire.

Ainsi, on peut calculer l'aire d'un sous-domaine selon la formule suivante :

$$aire\_sous\_domaine = \frac{nb\_points\_dans\_sous\_domaine}{nb\_points\_total} \cdot aire\_domaine.$$

La figure 1 illustre ce phénomène. Ici, notre domaine d'aire connue sera le carré $\mathcal{P}$ composé des sommets (0,0), (0,1), (1,1) et (1,0). Notre sous-domaine correspondra au premier quart du cercle trigonométrique $\mathcal{C}$. Leurs aires respectives sont 1 et $\frac{\pi}{4}$.

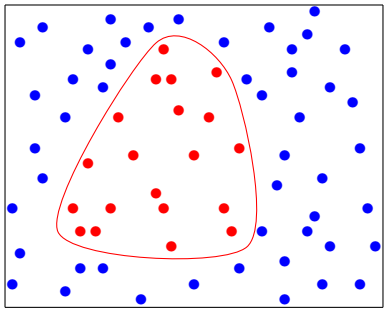

FIGURE 1 – Représentation d'un domaine $\mathcal{V}$ en gris, et d'un de ses sous-domaines $\mathcal{U}$ en rouge. Des points ont été tirés aléatoirement et ont été coloriés en rouge ou en bleu suivant leur appartenance à $\mathcal{U}$ ou à $\mathcal{V}\backslash\mathcal{U}$.

Pour réaliser ce TP, nous avons besoin de pouvoir :

- tirer des points aléatoirement
- déterminer leur appartenance à un sous-domaine
- compter le nombre de points ayant atterrit dans un sous-domaine particulier

1 Tirage de points aléatoires

Dans cette première partie, nous allons nous concentrer sur la rédaction d'une fonction permettant de tirer aléatoirement des points dans $\mathcal{P}$. Idéalement le nombre de points à tirer sera déterminé par l'utilisateur à l'exécution du script.

1. Écrivez une fonction *get_random_point()* qui permet de tirer aléatoirement un point ayant ses coordonnées comprises entre 0 et 1. Il sera nécessaire d'utiliser le package *random* de python !
2. Complétez la fonction *get_10_points()* pour tirer aléatoirement 10 points dont les coordonnées sont comprises entre 0 et 1.
3. Implémenter la fonction *get_points(nb_points)* afin de tirer un nombre de points spécifié en argument de la fonction.
4. Écrire la fonction *get_point_final(nb_points, display)*, qui reprend les spécification de la fonction précédente et permettant d'afficher les points tirés aléatoirement. On y ajoutera également des paramètres par défaut concernant le nombre de points à tirer et l'affichage.

2 Appartenance à un sous-domaine

Dans seconde cette partie, nous allons nous intéresser à l'écriture d'une fonction permettant de connaître l'appartenance d'un point à un sous-domaine de $\mathcal{P} : \mathcal{C}$.

5. Écrivez une fonction *compute_distance(x_point, y_point, x_circle, y_circle)* qui calcule la distance euclidienne entre les 2 points (x_point, y_point) et (x_circle, y_circle).
6. Modifiez la fonction *is_inside_circle(x_point, y_point, x_circle, y_circle, r)*, afin qu'elle renvoie un booléen **True** si le point est dans le cercle de centre (x_circle, y_circle) et de rayon r, et **False** sinon.
7. Modifiez la fonction afin que par défaut, le cercle considéré soit de centre (0,0) et de rayon 1.

3 Calcul de la valeur de π

Maintenant que nous sommes capables de tirer des points aléatoirement et de savoir s'ils sont à l'intérieur d'un sous-domaine particulier, nous allons estimer la valeur de π.

8. Écrivez une fonction *nb_points_sub_domain(points)* qui prend en entrée une liste de points et qui renvoie le nombre de points à l'intérieur de notre sous-domaine.
9. Estimez la valeur de π en tirant successivement 10, 100, 1000, ..., points aléatoirement. Vous écrirez un nouveau script python et utiliserez les fonctions précédemment implémentées.
10. Quels sont les points forts / faibles de cette méthode ?

4 Affichage graphique

Pour ceux qui ont fini les questions précédentes, on pourra essayer de traiter les questions suivantes visant à afficher graphiquement les calculs réalisés précédemment.

11. Créez une fonction *show_points(x, y)* qui prend en entrée les coordonnées d'un point et qui l'affiche graphiquement.
12. Modifiez la fonction précédente pour qu'elle affiche une liste de points.
13. Ajoutez dans la même fenêtre la courbe représentant $\mathcal{C}$.
14. Coloriez les points de couleur différente suivant qu'ils soient dans $\mathcal{C}$ ou en dehors.

Pour cette série de question, on pourra se servir de la librairie *matplotlib* (Hunter, 2007) de la façon suivante pour les représentations graphiques :

- **import** : *import matplotlib as plt.*
- **affichage d'un point** : *plt.plot(x,y)*, il existe un troisième paramètre permettant de personnaliser cet affichage (forme et couleur).
- **affichage d'un graphique** : *plt.show()*.

Un exemple de résultat est montré en figure 2.

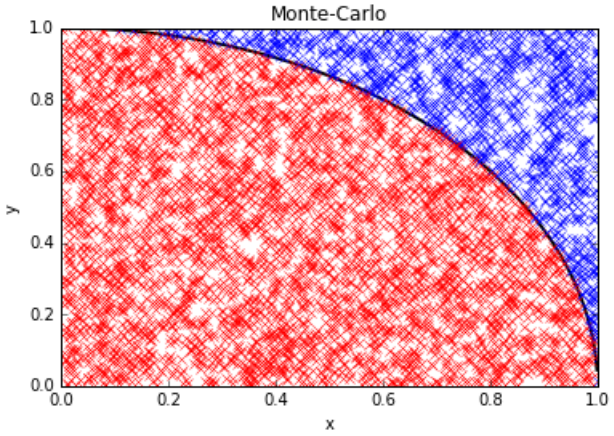

FIGURE 2 – Affichage graphique de la méthode de Monte-Carlo pour estimer π

5 Estimation de l'erreur

L'idée ici utilisée est d'exprimer une surface comme l'espérance d'une variable aléatoire X qui vaut 1 si un point tiré aléatoirement se trouve dans cette surface et 0 sinon. Ici, la surface est le quart de cercle trigonométrique.

Lorsque l'on fait un grand nombre de tirages, on peut supposer que les valeurs obtenues X_i suivent une loi gaussienne. On peut ensuite estimer l'erreur de la méthode avec une certaine probabilité (*ex. : il y a 80% de chances que π soit compris entre 3.1375 et 3.1498*).

Dans cette partie, on va s'attacher à l'estimation de l'erreur pour garantir que la valeur obtenue est située dans un intervalle $(x - \epsilon, x + \epsilon)$. On pourra trouver ϵ à partir de la formule suivante : $\epsilon = c\sqrt{\frac{\mathcal{V}}{n}}$ avec $\mathcal{V}$ la variance de la distribution X, n le nombre de tirages et c un nombre dépendant de la probabilité de trouver π dans l'intervalle obtenu.

15. Créez une fonction *variance(list_points)* qui prend en paramètre une liste de points et qui calcule leur variance.
16. Calculez l'intervalle dans lequel π se situe avec une probabilité de 99%, puis de 95%.
17. Relancez plusieurs fois l'expérience, que constatez-vous ?

Pour cette dernière série de questions, on pourra se servir de la fonction *sqrt* du module *math* de la façon suivante :

- **import** : import math
- **calcul de la racine carrée de x** : math.sqrt(x)

Voici quelques valeurs de c accompagnées des probabilités correspondantes. Il existe des tables qui référencent toutes les valeurs de c ainsi que leurs probabilités.

Valeur de c	Probabilité
1.28	80
1.65	90
1.96	95
2.58	99

Références

J. D. Hunter. Matplotlib : A 2d graphics environment. *Computing In Science & Engineering*, 9(3) :90–95, 2007.

N. Metropolis and S. Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247) :335–341, 1949.