30 - Exemples d'algorithmes utilisant un générateur de nombres aléatoires.

I- Introduction

Définition d'un algorithme :

Un algorithme est une suite finie et non ambiguë d'opérations ou d'instructions permettant de résoudre un problème ou d'obtenir un résultat. Il est dit correct si, pour chaque instance du problème, il produit la bonne sortie.

Algorithme déterministe vs probabiliste :

Un algorithme déterministe est un algorithme qui, étant donné une entrée particulière, produira toujours la même sortie.

Un algorithme probabiliste est un algorithme qui utilise à la fois, les données du problème et des valeurs produites par un générateur pseudo-aléatoire. Cela veux dire que pour un même jeu de données, l'algorithme peut rendre deux résultats différents.

Pourquoi parle-t-on de pseudo-aléatoire :

Comme un générateur de nombres aléatoires est exécuté sur un ordinateur déterministe, il devient de facto un algorithme déterministe. Il n'est pas possible pour un ordinateur de créer son propre hasard en lançant un dé ou une pièce.

Il existe différents générateurs aléatoires proposés par la bibliothèque « random » python :

- random() :renvoie un nombre décimal entre [0,1[
- randint : renvoie un entier entre [a,b]

Elle existe d'autre générérateur tel que gauss, exponentielle, logarithmique, uniforme, normale, …

Pourquoi utiliser des algorithmes probabilistes ?

1er cas : on connaît un algorithme déterminisme qui résout le problème mais l'algorithme probabiliste est plus rapide et donne le bon résultat avec une probabilité très forte.

2ème cas : L'algorithme probabiliste est plus simple à comprendre et plus facile à même en œuvre. Il est tout aussi efficace et rapide que l'algorithme déterministe.

3ème cas : On ne connaît pas d'algorithme déterministe aussi efficace.

Parmi les algorithmes probabilistes, on distingue ceux dit de :

- Monte-Carlo
- Las Vegas
- Atlantic City (peu de doc)

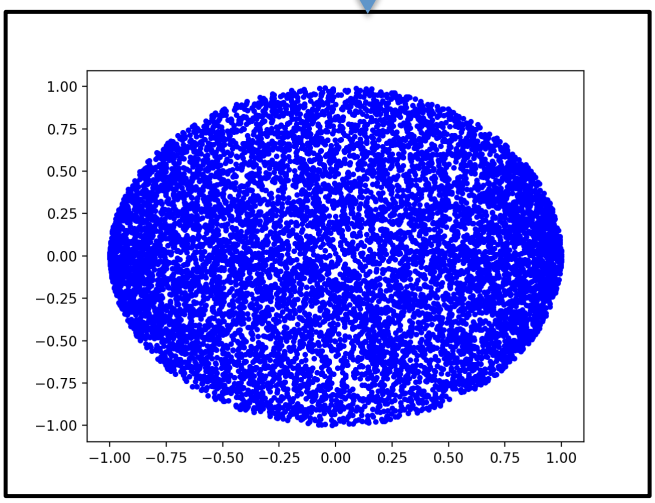
II- Monte-Carlo

Un algorithme de Monte-Carlo est un algorithme randomisé dont le temps d'exécution est déterministe mais dont le résultat peut-être incorrect avec une certaine probabilité.

Voyons comment répartir uniformément des points d'un ensemble A dans un sous-ensemble B (un disque de centre (0,0) et de rayon 1) :

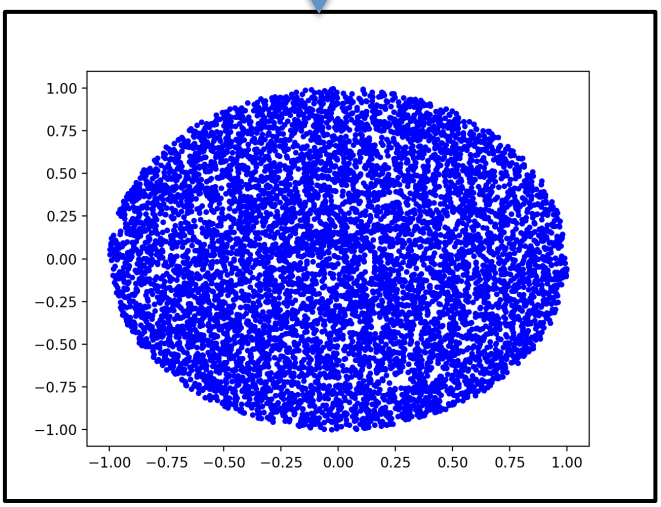
Algorithme 1 :

Générer x sur A
Répéter :
    Générer y sur B
Tant que (x,y) ∉ B



Algorithme 2 :

Générer x sur A
Générer y sur A
Si (x,y) ∉ B :
    Conserver (x,y)
Sinon :
    Ignorer (x,y)



Comme on peut le voir, l'algorithme 1 ne donne pas une répartition uniforme des points. En effet, la concentration de points est plus importante sur les extrémités qu'au centre. Cela vient du fait que peut importe le x, on cherche un y qui vérifie l'équation $x^2+y^2<1$, or sur les côtés, l'intervalle est plus petit et donc les points se superposent.
Pour l'algorithme 1, x et y ne sont pas indépendants.

Développement : Algorithme de Monte-Carlo pour l'approximation de l'aire de pi

Généralisation pour l'intégration d'une fonction (aire sous la courbe)

Développement : SCMA/CD with Binary exponential backoff

C'est un algorithme qui est utilisé pour gérer les conflits de communication sur un réseau informatique.

Des machines sont connecté entre elles par un câble et peuvent s'envoyer des messages, or il arrive que des messages soit envoyer en même, se rencontrent et bloquent la ligne. Si cette collision arrive trop de fois, les deux machines vont tirer aléatoirement un nombre entre 0 et 2^k-1 avec k entier $\notin$ [a,b].
Ce nombre représente le temps d'attente, en unité de temps quelconque, avant d'envoyer le message. Si le nombre tiré est le même, alors on incrémente k de 1 et on fait retirer les nombres. On diminue alors la probabilité de tirer les mêmes nombres.

Le temps d'exécution est déterministe mais le résultat peut être que les deux machines tirent le même nombre à chaque fois et donc le message n'est jamais envoyé. Cette probabilité reste très faible pour un k très grand.

III- Las Vegas

C'est un algorithme qui, bien qu'utilisant de l'aléatoire, produit un résultat correct, donc qui produit exactement le même résultat qu'un algorithme qui n'utilise pas de probabilité du tout

Exemple : QuickSort avec choix du pivot aléatoire.

Développement : Le jeu des huit reines

Principe : Placer reines sur un échiquier sans qu'elles ne puissent s'attaquer.

Développement : Shuffle

Comment fonction le shuffle de python pour renvoyer une liste mélanger uniformément ?

IV- Ouverture : un exemple de générateur de nombres pseudo-aléatoires

Le Linear Feedback Shift Register (LFSR) est un générateur pseudo-aléatoire fonctionnait par rétroaction sur un mot binaire.

Le principe est simple, on choisit un mot binaire de k bits (graine/seed), ensuite renvoie le bit de poids fort en décalant de 1 à gauche les autre bit et en ajoutant un 0 à droite. Si le bit de poids fort est 1, alors on inverse les bits de position fixée à l'avance (avec XOR de python). On obtient ainsi un nouveau mot binaire et on répète l'opération jusqu'à avoir retrouver la graine

Comme tout les générateurs pseudo-aléatoire, il est cyclique mais en fonction de la longueur du mot de départ et des positions des bits à inverser, on peux générer un très grand échantillon de nombre pseudo aléatoire.

Développement : code en python

LFSR.py