L2-S3 : UE Simulations numériques

SEANCE 9

Nombres aléatoires et méthode Monte-Carlo

13 avril 2022

Processus aléatoires et nombres aléatoires

- Certains processus en physique sont aléatoires, comme par exemple le phénomène de désintégration radioactive.
- D'autres processus ne sont pas strictement aléatoires mais sont vus comme tels à l'échelle macroscopique. Un exemple est le mouvement Brownien.
- L'utilisation de simulations par ordinateur permet d'étudier de tels phénomènes qui ne sont pas toujours accessibles par une approche analytique.
- On trouve des exemples de ces méthodes déjà à l'époque des premiers ordinateurs : des expériences numériques ont été effectuées au cours du projet Manhattan pour calculer le libre parcours moyen d'un neutron dans un matériau.

Processus aléatoires et nombres aléatoires

Comment générer des nombres aléatoires avec un algorithme informatique ?

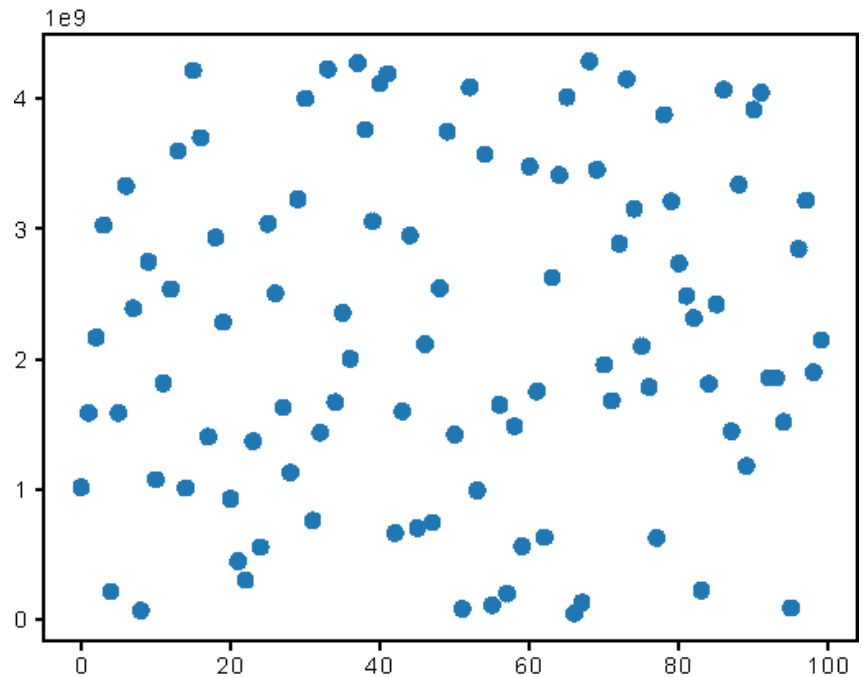
- Une suite aléatoire ne doit avoir aucune règle de prédiction identifiable.
- Il n'est donc pas possible d'écrire un programme qui génère une suite strictement aléatoire !
- Cependant il existe un grand nombre d'algorithmes déterministes qui permettent de générer des suites pseudo-aléatoires.
- Dans des simulations numériques nous pouvons les traiter comme des vraies suites aléatoires.

Processus aléatoires et nombres aléatoires

Exemple : Générateur congruentiel linéaire

$$x_{n+1} = (ax_n + c) \mod m$$

- A chaque itération n la valeur de x_{n+1} dépend de la valeur de x_n à l'itération précédente.
- La qualité du générateur dépend des paramètres a, c et m.
- La valeur initiale x est la graine (seed) de la série et définit la suite qu'on va obtenir.



Résultat d'une génération aléatoire de points (x_i, y_i).

Nombres aléatoires en python

Le module `numpy.random` permet de générer des nombres pseudo-aléatoires en utilisant des algorithmes déterministes.

La fonction `random()` génère des nombres pseudo-aléatoires uniformément répartis dans l'intervalle [0,1[.

In [1]:
```
1 import numpy as np
2
3 np.random.random()
```

Out[1]:
```
0.6314215736329736
```

La fonction prend aussi comme argument le nombre de valeurs qu'on veut générer.

In [2]:
```
1 np.random.random(3)
```

Out[2]:
```
array([ 0.74803697,  0.92541309,  0.42874785])
```

Nombres aléatoires en python

La fonction `seed()` définit la graine du générateur et ainsi d'une façon univoque la suite des nombres qui s'en suivent. L'utilisation d'une graine peut être extrêmement pratique en phase de développement.

In [3]:
```
1 np.random.seed(10)
2 np.random.random()
```

Out[3]:
```
0.771320643266746
```

In [4]:
```
1 np.random.random()
```

Out[4]:
```
0.020751949359401 5
```

In [5]:
```
1 np.random.seed(10)
2 np.random.random()
```

Out[5]:
```
0.771320643266746
```

In [6]:
```
1 np.random.random()
```

Out[6]:
```
0.020751949359401 5
```

Nombres aléatoires en python

Autres fonctions :

- *np.random.uniform(low,high,size=None)* : renvoie des flottants (réels) aléatoires dans l'intervalle [low, high[
- *np.random.randint(low,high,size=None)* : renvoie des entiers aléatoires dans l'intervalle `low` (inclus) à `high` (exclus).
- *np.random.choice(s,size=None)* : génère un échantillonnage aléatoire à partir d'une séquence `s` donnée (`s` doit être 1D)
- *np.random.permutation(s)* : permute aléatoirement un array ou renvoie une plage de nombres entiers permutés.

L'option `size` définit le nombre de valeurs qu'on veut générer. Le résultat correspond à un objet de type array de taille `size`.

Exemples d'utilisation des fonctions de np.random

Tirage de nombres entiers dans un intervalle :

In [7]:
```
1 np.random.randint(-5,10,size=4)
```

Out[7]:
```
array([7, 9, 4, 9])
```

Tirage dans un ensemble prédéfini

In [8]:
```
1 np.random.choice([12.2,3.5,15,13],size=3)
```

Out[8]:
```
array([ 12.2,  15. ,  12.2])
```

Exemples d'utilisation des fonctions de np.random

Permutations aléatoires dans un tableau :

In [9]:
```
1 np.random.permutation([12,23,45])
```

Out[9]:
```
array([45, 12, 23])
```

In [10]:
```
1 arr = np.arange(9).reshape((3, 3))
2 np.random.permutation(arr)
```

Out[10]:
```
array([[6, 7, 8],
       [0, 1, 2],
       [3, 4, 5]])
```

In [11]:
```
1 np.random.permutation(6)
```

Out[11]:
```
array([3, 1, 2, 4, 0, 5])
```

Méthode Monte-Carlo

Les méthodes Monte-Carlo visent à calculer une valeur numérique approchée en utilisant des procédés aléatoires.

Le nom fait allusion aux jeux de hasard pratiqués à Monte-Carlo.

Les méthodes suivent généralement les étapes suivantes :

- on définit un domaine de possibles configurations d'entrée ;
- on génère les configurations d'entrée en suivant une loi probabiliste ;
- on calcule chacune des configurations suivant une loi déterministe ;
- on agrège les résultats.

Méthode Monte-Carlo

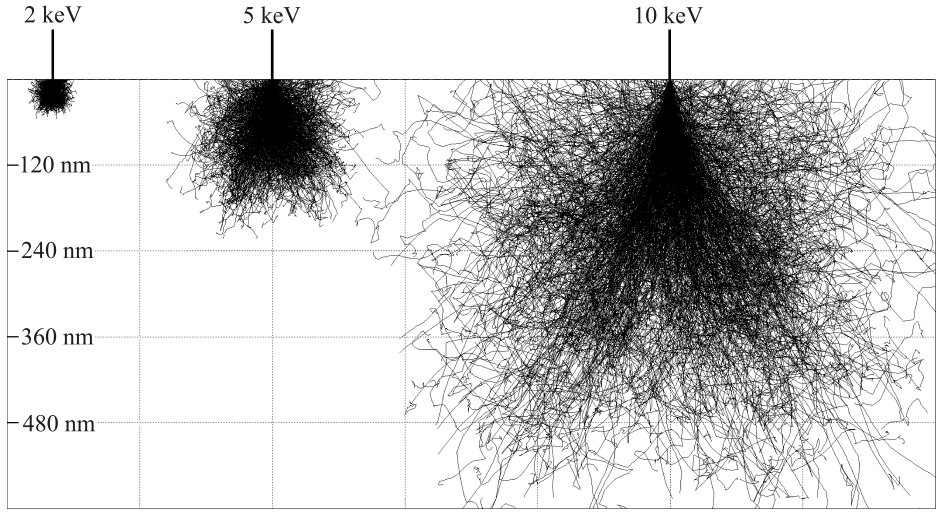
Exemple : quelle est la probabilité de toucher le bord d'un carreau en lançant une pièce de monnaie sur un sol carrelé ?

- On définit la taille du sol ainsi que les dimensions des carreaux et de la pièce de monnaie.
- On génère N configurations dans lesquelles la pièce de monnaie a été placée au hasard sur le sol.
- On vérifie si la pièce de monnaie a touché le bord d'un carreau.
- On dérive la fréquence des occurrences positives dans les N configurations. Pour N grand on peut estimer la probabilité souhaitée.

Méthode Monte-Carlo

Exemples de domaines d'utilisation :

- Diffusion de particules dans un solide irradié.
- Croissance cristalline.
- Dynamique de populations.
- Modélisation de trafic automobile.
- Évolution de portfolio d'investissement.
- Raytracing
- ...
- Calculs d'intégrales.



Simulation Monte-Carlo de la scintillation d'un cristal YAG suite à l'impact d'électrons de différentes énergies.

Intégration par la méthode de Monte-Carlo

Il s'agit d'évaluer numériquement la valeur de l'intégrale d'une fonction f définie dans un espace $\mathbb{R}^d$

$$I = \int_{\Omega} f(\mathbf{x}) d\mathbf{x}$$

La vitesse de convergence est un indicateur de l'efficacité de la méthode utilisée pour évaluer l'intégrale.

- Méthode des trapèzes : $n^{-2/d}$
- Méthode de Simpson : $n^{-4/d}$

Pour ces méthodes la vitesse de convergence diminue quand d augmente.

Intégration par la méthode de Monte-Carlo

Alors que d'autres algorithmes évaluent habituellement l'intégrande sur une grille régulière, Monte Carlo choisit aléatoirement les points auxquels l'intégrande est évaluée.
On choisi N points aléatoires dans le domaine d'intégration Ω

$$\mathbf{x_1} \cdots \mathbf{x_N} \in \Omega$$

Le volume du domaine d'intégration est

$$V = \int_{\Omega} d\mathbf{x}$$

L'intégrale de la fonction f peut être approximée par :

$$I \approx Q_N \equiv \frac{V}{N} \sum_{i=1}^{N} f(\mathbf{x_i})$$

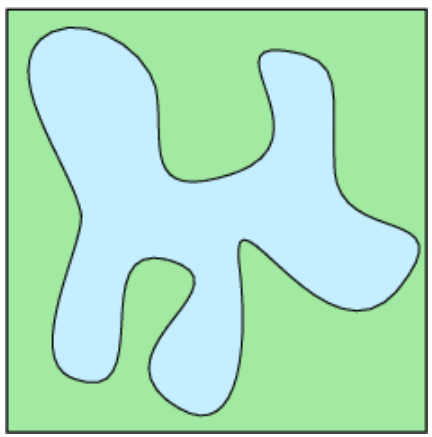
Intégration par la méthode de Monte-Carlo

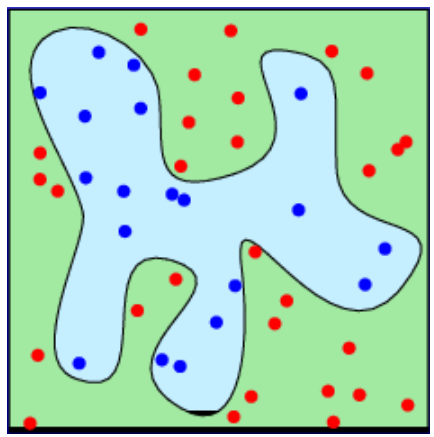
- La vitesse de convergence est $n^{-1/2}$ indépendamment de la dimension du problème.
- La méthode Monte Carlo est ainsi particulièrement utile pour les intégrales à plusieurs dimensions.
- L'échantillonnage du domaine d'intégration est normalement homogène mais des lois de probabilité plus efficaces peuvent être choisies en fonction de la situation (cas par exemple de fonctions divergentes).

Intégration par la méthode de Monte-Carlo

- Il s'agit de trouver *manu militari* la valeur de la surface S_L d'un lac à l'intérieur d'un terrain de surface connue S_T.
- On tire N boulets de canon sur le terrain d'une façon aléatoire et homogène et on compte le nombre de boulets M qui sont tombés dans le lac.
- Pour un nombre N très grand on peut estimer la surface du lac comme :

$$S_L = \frac{M}{N} S_T$$





Comment calculer π par cette méthode ?

DM2

Le DM2 est disponible aujourd'hui et à rendre avant le

mercredi 27 avril, heure du cours

1. Ouvrir un terminal :
   - soit sur https ://jupyterhub.ijclab.in2p3.fr
   - soit sur un ordinateur du 336
2. Télécharger le DM2

```
methnum fetch L1/DM2 TONGROUPE
```

   en remplaçant TONGROUPE par ton numéro de groupe.
3. Pour soumettre le DM2, dans le terminal taper :

```
methnum submit L1/DM2 TONGROUPE
```

À vos TPs !

1. Ouvrir un terminal :
   - soit sur https ://jupyterhub.ijclab.in2p3.fr
   - soit sur un ordinateur du 336
2. Télécharger la séance d'aujourd'hui :

```
methnum fetch L1/Seance9 TONGROUPE
```

   en remplaçant TONGROUPE par ton numéro de groupe.
3. Sur un ordinateur du bâtiment 336 uniquement, lancer le jupyter :

```
methnum jupyter notebook
```

4. Pour soumettre la séance, dans le terminal taper :

```
methnum submit L1/Seance9 TONGROUPE
```

À vos TPs !

Rappel : votre gitlab personnel sert de sauvegarde pour passer vos documents d'une plateforme à l'autre via les commandes fetch/submit.