Chaînes de Markov (1)

Modélisation de phénomènes aléatoires - simulations de chaînes de Markov

Modélisation de phénomènes aléatoires - simulations de chaînes de Markov

1 Une puce se déplace sur la ligne suivante en sautant au hasard d'une case ou de deux cases vers la droite. Elle part de la case 1 (et elle va toujours vers la droite).
On observe sa position après 6 sauts.

1	2	3	4	5	6	7	8	9	10	11	12	13

1°) Quelles sont les cases qui peuvent être atteintes après 6 sauts ?

2°) Recopier et compléter le programme Python suivant pour faire afficher la position de la puce après 6 sauts :

```
from random import randint

def position():
    position=.....
    for i in range(..............):
        position=position + randint(1,2)
    return position
```

Exécuter le programme 10 fois et compléter le tableau :

Issue							
Fréquence							

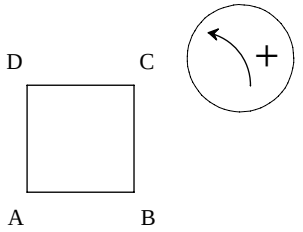
3°) Recopier et compléter le programme suivant pour qu'il répète l'expérience 100 fois et qu'il comptabilise les arrivées à la case 8.

```
from random import randint

def position():
    position=.....
    for _ in range(..............):
        position=position + randint(1,2)
    return position

def arrivee8():
    arrivee=0
    for i in range(100):
        if ..................................... :
            arrivee=arrivee + 1
    return arrivee
```

4°) Modifier la deuxième partie du programme pour pouvoir comptabiliser les arrivées dans n'importe quelle case saisie en paramètre (*n*) :

```
def arrivee(n):
    arrivee=0
    for i in range(100):
```

2 Soit ABCD un carré direct dans le plan orienté.
Une fourmi parcourt les côtés de ce carré en partant du sommet A et met une minute pour parcourir un côté. Arrivée à un sommet, elle choisit au hasard l'un ou l'autre des deux côtés issus de ce sommet pour poursuivre sa marche.



Quels sont les sommets qui peuvent être atteints après 1 minute ? 2 minutes ? 3 minutes ? 4 minutes ? Généraliser.

Pour simuler les déplacements de la fourmi, on affecte les chiffres 0, 1, 2, 3 au fait d'arriver respectivement aux sommets A, B, C et D.
Si la fourmi se déplace dans le sens direct, on ajoute 1 lors du déplacement. Sinon, on retranche 1 et on additionne au fur et à mesure des déplacements.

Recopier et compléter la fonction Python `position(n)` qui prend pour argument un entier naturel *n* supérieur ou égal à 1 et qui affiche la position de la fourmi après *n* minutes :

```
from random import choice

def position(n):
    s=.....
    for i in range(..............):
        x=choice([- 1,1])
        s=s+ .....
        s=s%4
    return s
```

Exécuter le programme.

Il est possible de généraliser à un déplacement quelconque sur un polygone régulier.

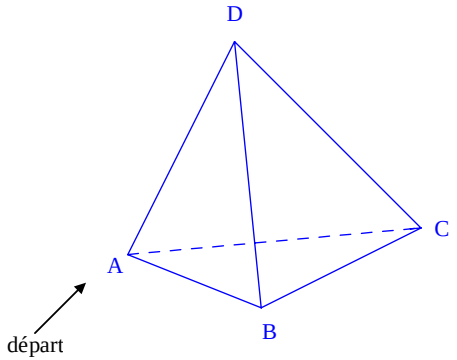
3 Un pion se déplace sur le damier suivant constitué de 9 cases. Au départ, il est sur la case 0.

– 4	– 3	– 2	– 1	0	1	2	3	4

Les déplacements du pion sont déterminés par le lancer d'une pièce de monnaie bien équilibrée.
Si la pièce tombe du côté face, le pion se déplace d'une case vers la droite, sinon il se déplace d'une case vers la gauche. Si le pion est sur la case 4 ou – 4, il y reste.
1°) On envisage un trajet qui est une succession de 4 déplacements.
Écrire une fonction Python qui simule un trajet, c'est-à-dire qui renvoie la position finale du pion.
2°) On envisage un trajet qui est une succession de *n* déplacements où *n* est un entier naturel supérieur ou égal à 1.
Même question ue précédemment.

4 On promène un pion sur les sommets d'un tétraèdre régulier ABCD. Toutes les minutes, on déplace le pion d'un sommet à un autre, en choisissant au hasard parmi les trois sommets possibles.
On suppose qu'au départ le pion se trouve en A.
On envisage *n* déplacements où *n* est un entier naturel supérieur ou égal à 1.
Écrire une fonction Python qui prend pour argument l'entier naturel *n* et qui renvoie la position où se trouve le pion au bout de *n* déplacements.

Figure :



5 Les urnes d'Ehrenfest à 2 boules

On considère deux urnes A et B. On dispose également d'une pièce de monnaie équilibrée.
Au départ,
- l'urne A contient deux boules indiscernables au toucher, numérotées 1 et 2 ;
- l'urne B est vide.

On effectue des lancers successifs indépendants de la pièce en notant à chaque fois le côté qu'elle présente.
Si elle présente le côté pile (P), on change d'urne la boule numéro 1.
Si elle présente le côté face (F), on change d'urne la boule numéro 2.
On s'intéresse à l'évolution du nombre de boules dans A.
On s'intéresse à l'évolution du nombre de boules de chaque urne.

1°) Dans cette question, on choisit de s'intéresser à une série de 7 lancers de la pièce.
On suppose que les 7 lancers ont donné les résultats suivants : P-P-F-P-F-F-P.
Originellement, les deux boules sont dans l'urne A.
Le premier lancer donne pile donc on change d'urne la boule 1. La boule 1 est donc placée dans l'urne B.

Recopier et compléter le tableau suivant :

		Urne A	Urne B	Nombre de boules dans A
Étape 0	Au départ	1-2		
Étape 1	Après le 1er lancer	2	1	
Étape 2	Après le 2e lancer	1-2		
Étape 3	Après le 3e lancer	1	2	
Étape 4	Après le 4e lancer		1-2	
Étape 5	Après le 5e lancer			
Étape 6	Après le 6e lancer			
Étape 7	Après le 7e lancer			

Représenter sur un graphique le nombre de boules dans A en fonction du numéro de l'étape (on peut parler de « trajectoire »).
2°) Démontrer qu'au bout de *n* étapes, les deux boules sont dans la même urne si *n* est pair et ne sont pas dans la même urne si *n* est impair.
3°) On considère le programme Python suivant :s : nombre de boules de l'urne A

```
from random import choice

def urne(n):
    s=2
    for i in range(n):
        if s==2:
           s= 1
        elif s==0:
           s=1
        else:
           x=choice([-1,1])
           s=s+x
        print (s)
```

Lire et comprendre ce programme puis le taper et le faire tourner.

1ère version :

```
from random import randint

def ehrenfest(n):
    A=[1,2]
    for _ in range(n):
        x=randint(1,2)
        if x in A:
           A.remove(x)
        else:
           A.append(x)
    return A # contenu de l'urne A au bout de n étapes
```

2e version :

```
from random import randint
import matplotlib.pyplot as plt

def ehrenfest(n):
    A=[1,2]
    X=[2]   #liste répertoriant le nombre de boules dans l'urne A
    for _ in range(n):
        b=randint(1,2)
        if b in A:
           A.remove(b)
        else:
           A.append(b)
        X.append(len(A))
    return X

def graphique(n):
    plt.xlim(0,n+1)
    plt.ylim(0,3)
    plt.grid(linestyle="-")
    x=[i for i in range(n+1)]
    plt.plot(x,ehrenfest(n),marker='o')
    plt.show()
```

Le 27-3-2023

Proposition Olivier Allard

```
def ehr(n) :
     x=2
    M=[x]
    for _ in range(n) :
         if x==2 :
            x=1
        elif x==1 :
             x=choice([0,2])
        else :
             x=1
      M.append(x)
     return M

def traj(n) :
    x=list(range(n+1))
    plt.plot(x,ehr(n))
    plt.show()
```

6 **Les urnes d'Ehrenfest à N boules**

On considère deux urnes A et B.
Au départ,
- l'urne A contient N boules indiscernables au toucher (N étant un entier naturel supérieur ou égal à 1) numérotées 1, 2, …, N ;
- l'urne B est vide.

À chaque étape on choisit un entier au hasard entre 1 et N.
La boule portant ce numéro est changée d'urne.

```
from random import randint
import matplotlib.pyplot as plt

def ehrenfest(N,n):
    A=[i for i in range(1,N+1)] ou list(range(1,N+1))
    X=[N]
    for i in range(n):
        b=randint(1,N)
        if b in A:
           A.remove(b)
        else:
           A.append(b)
        X.append(len(A))
    return X

def trajectoire(N,n):
    plt.xlim(0,n)
    plt.ylim(0,N+1)
    plt.grid(linestyle= "–" )
    x =[i for i in range (n+1)]
    plt.plot(x,ehrenfest(N,n),marker='o')
    plt.show()
```

7 Une urne A contient 2 boules blanches et une urne B contient 4 boules noires.
On procède au tirage aléatoire simultané d'une boule de chaque urne et la boule extraite est changée d'urne.
On répète cette opération.
On s'intéresse au contenu de l'urne A à chaque étape.
Il y a trois états possibles :
« L'urne A contient 2 boules blanches » (état 1),
« L'urne A contient 1 boule blanche et 1 boule noire » (état 2),
« L'urne A contient 2 boules noires » (état 3).
On notera que le nombre de boules est constant dans chaque urne.
Le contenu de l'urne B se déduisant à chaque fois de celui l'urne A, il n'est pas utile de le préciser pour définir les états.
Écrire une fonction Python `exp(n)` qui prend pour argument un entier naturel *n* supérieur ou égal à 1 et qui renvoie une liste donnant le nombre de boules blanches dans l'urne A au cours de *n* tirages successifs.
On utilisera des listes et les fonctions append et remove.

Solutions des exercices

1 Une puce se déplace sur la ligne suivante…

Corrigé :

Après 6 sauts, la fourmi peut atteindre les cases 7, 8, 9, 10, 11, 12, 13.

1°)

```
from random import randint

def position():
    position=1
    for _ in range(1,7):
        position=position+randint(1,2)
    return position
```

À la place de `range(1,7)`, on peut aussi écrire `range(6)`.

On notera que la fonction position est une fonction sans argument.

```
from random import randint

def position():
    position=1
    for _ in range(1,7):
        position=position+randint(1,2)
    return position

def arrivee8():
    arrivee=0
    for i in range(100):
        if position()==8
           arrivee=arrivee + 1
    return arrivee
```

2°) Exécuter le programme 10 fois permet de siuler 10 séries de 6 sauts dans des conditions identiques indépendantes.

Les fréquences sont des nombres compris entre 0 et 1. La somme des fréquences est égale à 1.
Chacun a un tableau de fréquences différent.

Résultats de Maxime Guégan obtenus le mardi 23-4-2024

Issue	7	8	9	10	11	12	13
Fréquence	0,1	0,1	0,2	0,1	0,1	0,4	0

Comme on exécute le programme 10 fois, on divise les résultats par 10.

3°) `position()==n`

2 Soit ABCD un carré direct dans le plan orienté.
Une fourmi parcourt les côtés de ce carré en partant du sommet A…

Corrigé :

1 minute : sommets A et C
2 minutes : sommets B et D
3 minutes : sommets A et C
4 minutes : sommets B et D

Généralisation :
Au bout de *n* minutes (*n* étant un entier naturel supérieur ou égal à 1) :
La puce peut être en A ou C si *n* est pair.
La puce peut etre en B ou en D si *n* est impair.

```
from random import choice

def position(n):
    s=0
    for _ in range(1,n+1):
        x=choice([- 1,1])
        s=s+x
        s=s%4
    return s
```

On peut utiliser la fonction choice qui permet de choisir un élément au hasard dans une liste.

Variante possible : L'instruction « `2*randint(0, 1) - 1` » fournit un entier aléatoire égal à 1 ou à – 1.

```
from random import randint

def position(n):
    s=.....
    for i in range(..............):
        x = 2*randint(0, 1) - 1
        s=s + .....
        s=s%4
    return s
```

Exécuter le programme.

Il est possible de généraliser à un déplacement quelconque sur un polygone régulier.

3 Un pion se déplace sur le damier suivant constitué de 9 cases. Au départ, il est sur la case 0.

– 4	– 3	– 2	– 1	0	1	2	3	4

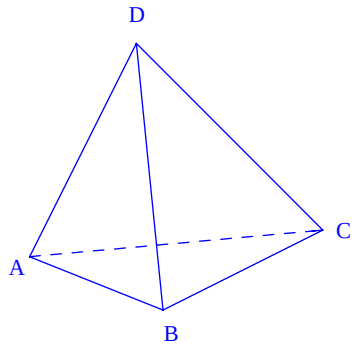
1°) On envisage un trajet qui est une succession de 4 déplacements.

```
from random import choice

s=0
for i in range(1,5):
    a=choice([-1,1])
    s=s+a
print (s)
```

2°) On envisage un trajet qui est une succession de *n* déplacements où *n* est un entier naturel supérieur ou égal à 1. Écrire une fonction Python qui permet de simuler un trajet.

4 On promène un pion sur les sommets d'un tétraèdre régulier ABCD. Toutes les minutes, on déplace le pion…

Solution :



Marche aléatoire sur un graphe connexe sur un graphe complet

On utilise 2 listes.

La variable *x* sert à noter la position du pion.

Programme avec toutes les positions :

```
from random import choice

def simul(n):
    x=1
    M=[x]
    for _ in range (1,n+1):
      L=[1,2,3,4]
      L.remove(x)
      x=choice(L)
      M.append(x)
    return M
```

On peut aussi utiliser une liste `L=['A','B','C','D']`.

Partie à enlever :

Indication : On pourra numéroter respectivement 0, 1, 2, 3 les sommets A, B, C, D.

Mon programme :
On ajoute un nombre aléatoire entre 0 et 3.
On réduit ensuite (division euclidienne par 4).

Le 20-4-2022

Programme Python fait par Vicente Seixas

```
from random import randint

L=['A','B','C','D']
p=0
for i in range(n):
    indice=p
    while indice==p:
          p=randint(0,len(L)-1)
    print (L[p])
```

Commentaires :

`L=['A', 'B', 'C', 'D']`	On initialise une liste.
`p=0` :	La fourmi commence en A.
`for i in range (n):`	On répète 10 fois.
`indice=p`	utile pour la suite
`while indice==p:`	On force la fourmi à se déplacer.