Méthodes numériques

Modélisation d’expériences aléatoire et estimation de distribution de loi

Anicet E. T. Ebou, ediman.ebou@inphb.ci

Ce travail est soumis à une licence internationale Creative Commons Attribution 4.0.

01

Tirages dans une urne modélisée par un tableau

Modélisation d’une urne

Un moyen simple de modéliser une urne est d’utiliser une liste que l’on remplit comme l’urne. Par exemple, pour une urne qui contient p boules blanches et q boules noires, on peut créer une liste avec p chiffres 1, et q chiffres 0.

```
def model_urne(p, q):
    urne = p*[1] + q*[0]
    return urne
```

Modélisation d’une urne

On adaptera sa modélisation en fonction du contexte. Si on veut compter le nombre de boules blanches pour différents tirages, ce seront elles que l’on codera par 1. Si, au contraire, on compte plutôt les boules noires, alors on échangera le codage. Le plus important ici est de commenter son code pour expliquer à l’utilisateur la convention choisie.

Tirage avec remise

Méthode de base: Pour tirer un objet avec remise, il suffit de le désigner dans l'urne (sans y toucher). On pourra utiliser pour cela la commande `random.choice`.

On peut ensuite réiterer l'expérience aléatoire de nombreuses fois de façon indépendantes. Dans certains cas, on voudra conserver la liste des résultats successifs, ou simplement le nombre de succès au cours des n tirages (par exemple le nombre de fois que l'on tire le chiffre 1).

Tirage avec remise

```
from random import choice
# tirage avec remise
def tirage(urne):
     return choice(urne)
# tirages successifs avec remise, on enregistre les résultats dans une liste.
def tirage_successif(urne, nb):
     res = []
     for i in range(nb):
          res.append(choice(urne))
     return res
# tirages successifs avec remise. on compte le nombre
# de fois que l’on a obtenu la valeur k.
def tirage_nb_succes(urne, k, nb):
     succes = 0
     for i in range(nb):
          succes += (choice(urne) == k)
     return succes
```

Tirage avec remise

Méthode alternative: On peut également modéliser ce tirage en choisissant une position aléatoire dans l'urne. On utilise alors la commande **randrange**(i,j,k) qui tire un nombre aléatoire dans l'intervalle donné par **range**(i,j,k). Comme pour la commande **range**, on peut diminuer le nombre d'arguments.

Tirage avec remise

```
from random import randrange
# tirage avec remise utilisation de randrange
def tirage(urne):
     indice = randrange(len(urne))
     return(urne[indice])
# tirages successifs avec remise. on enregistre les résultats dans une liste.
def tirage_successif(urne, nb):
     res = []
     for i in range(nb):
     res.append(tirage(urne))
     return res
# tirages successifs avec remise. on compte le nombre de fois que
# l’on a obtenu la valeur k.
def tirage_nb_succes(urne, k, nb):
     succes = 0
     for i in range(nb):
     succes += (tirage(urne) == k)
     return succes
```

Tirage sans remise

Cette fois-ci, il faut modifier l'urne à chaque tirage. Pour cela nous proposons de fonctionner avec effets de bords. Cela signifie que la liste (variable supposée globale) qui représente l'urne est modifiée à chaque tirage.

Attention: Si on veut effectuer plusieurs fois l'expérience de façon indépendante, alors il faudra recréer une nouvelle urne à chaque fois (ou travailler avec des copies d'une urne originale, en veillant à ce que les copies ne pointent pas toutes vers le même objet mémoire).

Tirage sans remise

Méthode de base: La méthode la plus simple est sans doute de mélanger l'urne avec **shuffle** puis de tirer le dernier élément. Lorsque l'on veut réaliser plusieurs tirages, il suffit d'un seul `shuffle`, puis on tire les éléments les uns à la suite des autres avec `pop()`.

Tirage sans remise

```
# tirage sans remise
def tirage(urne):
     random.shuffle(urne)
     return(urne.pop())
# n tirages sans remise mélange l’urne, mais ne la modifie pas.
def tirage_successif(urne, nb):
     random.shuffle(urne)
     return(urne[:nb])
# tirage sans remise,on compte le nombre de fois que l’on obtient la valeur k
# au cours de nb tirages l’urne est modifiée à la fin
def tirage_nb_succes(urne, nb):
     succes = 0
     random.shuffle(urne)
     for i in range(nb):
          succes += (urne.pop() == 1)
     return succes
```

Tirage sans remise

```
# pour ne pas modifier l’urne initiale
def tirage_nb_succes(urne, nb):
     succes = 0
     urneLocale = urne[:]
     random.shuffle(urne_locale)
     for i in range(nb):
          succes += (urne_locale.pop() == 1)
     return succes
```

Tirage sans remise

Remarque : Ici, dans l'algorithme du tirage successif, on prend les premiers éléments de la liste (sans les enlever), alors que dans les autres fonctions, on prend les derniers éléments (et on les enlève avec pop()). Cela n'a pas d'influence sur l'aléatoire, mais nous avons simplement écrit le plus facile à programmer dans chaque situation.

Tirage sans remise

Lorsqu’il s’agit d’un tirage sans remise exhaustif : on tire tous les éléments de l’urne. Cela revient simplement à choisir un ordre aléatoire pour notre liste (c’est une permutation).

Il suffit alors de la seule commande: random.`shuffle(urne)`

Tirage sans remise

Méthode alternative: Comme pour le tirage avec remise, on peut aussi pointer directement vers un indice avec **randrange** et le supprimer avec **pop**. Dans les algorithmes suivants, l'urne est modifiée à chaque fois. Comme précédemment, on peut empêcher ce résultat en créant une urne locale.

Tirage avec remise

```
def tirage(urne):
     indice = random.randrange(len(urne))
     return(urne.pop(indice))
# tirages successifs sans remise. on enregistre les résultats dans une liste.
def tirage_successif(urne, nb):
     res = []
     for i in range(nb):
          res.append(tirage(urne))
     return res
# tirages successifs sans remise.
# on compte le nombre de fois que l’on a obtenu la valeur k.
def tirage_nb_succes(urne, k, nb):
     succes = 0
     for i in range(nb):
          succes += (tirage(urne) == k)
     return succes
```

02

Approximation numérique de la probabilité

Estimation de la distribution en loi

On admet que lorsque l'on réalise une expérience de nombreuses fois et de façon indépendante, la proportion de chaque événement tend vers sa probabilité. Dans le cas d'une variable aléatoire réelle, la moyenne des résultats tend vers l'espérance.

Ainsi, il suffit de calculer la proportion ou la moyenne obtenues lors de la répétition de l'expérience pour avoir une valeur approchée de la probabilité et de l'espérance. Ceci est très simple à mettre en œuvre.

Estimation de la distribution en loi

Pour estimer la distribution en loi d'une variable aléatoire X, on commence par effectuer un échantillon de grande taille N de résultats d'une épreuve aléatoire suivant une loi X de la même manière que précédemment pour l'estimation de l'espérance et de la variance.

Ensuite, on estime $\boldsymbol{P(X = k)}$ tout simplement pour chaque valeur k des issues possibles de la variable aléatoire X en calculant la proportion des résultats égaux à k parmi l'échantillon obtenu.

Estimation de la distribution de loi: un algorithme

Inventaire des données:

- On connaît le principe de l'expérience aléatoire et le nombre N de répétition de l'expérience aléatoire;

Résultats attendus:

- La distribution de loi estimée.

Estimation de la distribution de loi: un algorithme

Actions:

- Simuler N fois un résultat de X en utilisant la procédure de simulation et conserver les résultats;
- Compter le nombre d’occurence de chaque valeur observée;
- Retourner la liste des proportions ou la visualiser.

Estimation de la distribution en loi

Par exemple, on dispose d’une urne qui contient *p* boules blanches et *q* boules noires. On note X le nombre de boules blanches obtenus lors de nb tirages avec remise. On cherche une approximation numérique de ***P(X = k)***.

```
# p(X = k)
def proba(urne, nb, k, n = 1000):
     res = 0
     for i in range(n):
          res += (tirage(urne, nb) == k) # on ajoute 1 si le tirage donne k
     return res/n          # proportion
```

Estimation de l’espérance: un algorithme

Inventaire des données:

- On connaît le principe d’une expérience aléatoire;
- On a défini une variable aléatoire X dépendant des résultats de cette expérience;
- On a une procédure qui nous permet de calculer des valeurs de cette variable aléatoire en respectant sa loi de probabilité (Voir cours précédent sur la simulation des v.a.r. en Python).

Estimation de l’espérance: un algorithme

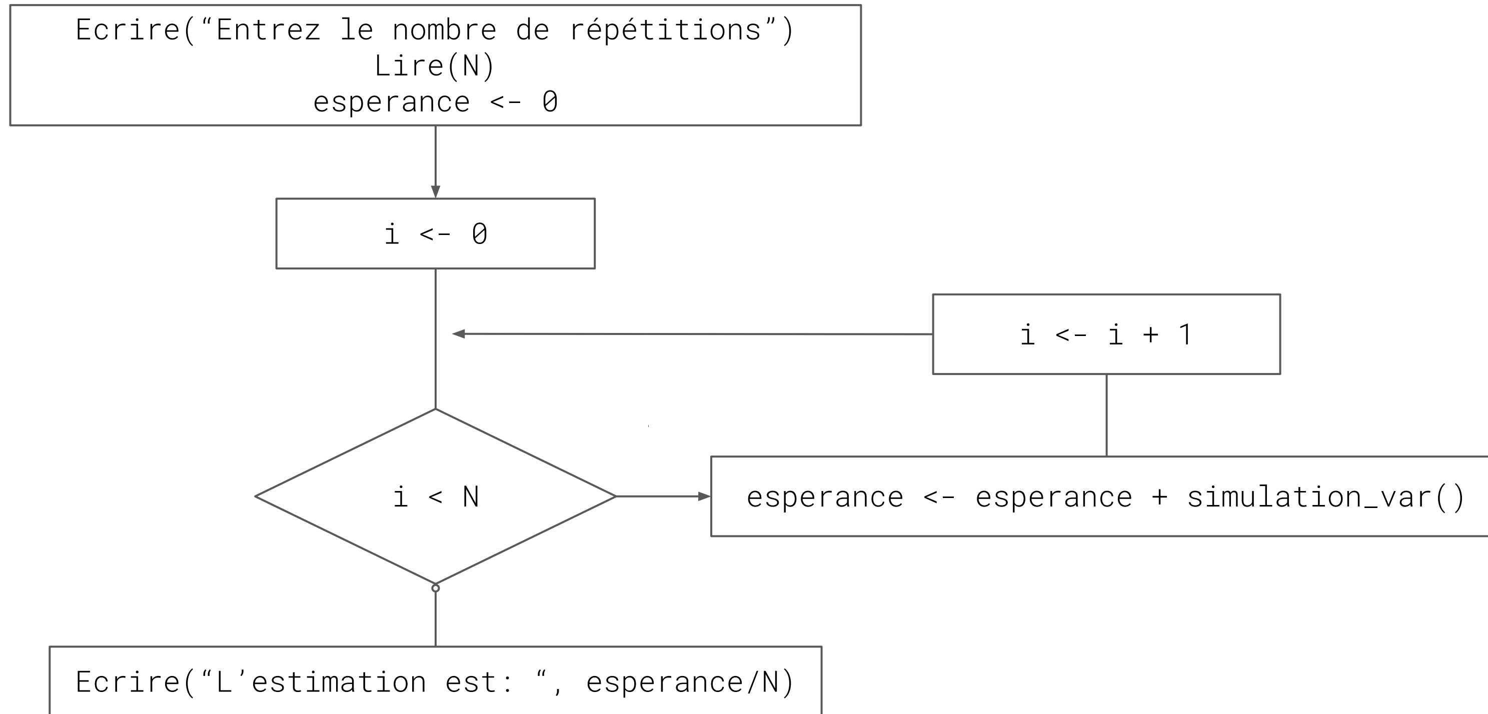
Résultats attendus:

- L’espérance estimée.

Actions:

- Simuler N fois un résultat de X en utilisant la procédure de simulation;
- Sommer toutes les valeurs obtenues;
- Diviser par le nombre N.

Estimation de l’espérance: un algorithme



Estimation de l’espérance: un algorithme

```
Algorithme Estimation_Esperance
# Cet algorithme permet d’estimer l'espérance d’une v.a.r suivant une loi
# simulée à l’aide de la fonction simulation_var
Var
    esperance: Réel
    N: Entier
Debut
    Ecrire(“Entrez le nombre de répétitions”)
    Lire(N)
    Pour i <- 1 à N faire
        esperance <- esperance + simulation_var()
    FinPour
    Ecrire(“L’espérance estimée est: “, esperance/N)
Fin
```

Estimation de l’espérance: un algorithme

```
def estimation_esperance(N):
     “””
     Fonction permettant l’estimation d’une espérance
     de v.a.r. a partir de la simulation de la v.a.r
     “””
     esperance = 0
     for i in range(N):
           esperance = esperance + simulation_var()
     return esperance/N
```

Estimation de la variance

La variance est l'espérance des écarts quadratique divisée par rapport à l'espérance. On peut donc à partir de la simulation de v.a.r. et de l'espérance estimée calculer la variance de la v.a.r.

Estimation de la variance: un algorithme

Inventaire des données:

- On sait comment estimer l'espérance;
- On connaît le nombre N de répétition de l'expérience aléatoire.

Résultats attendus:

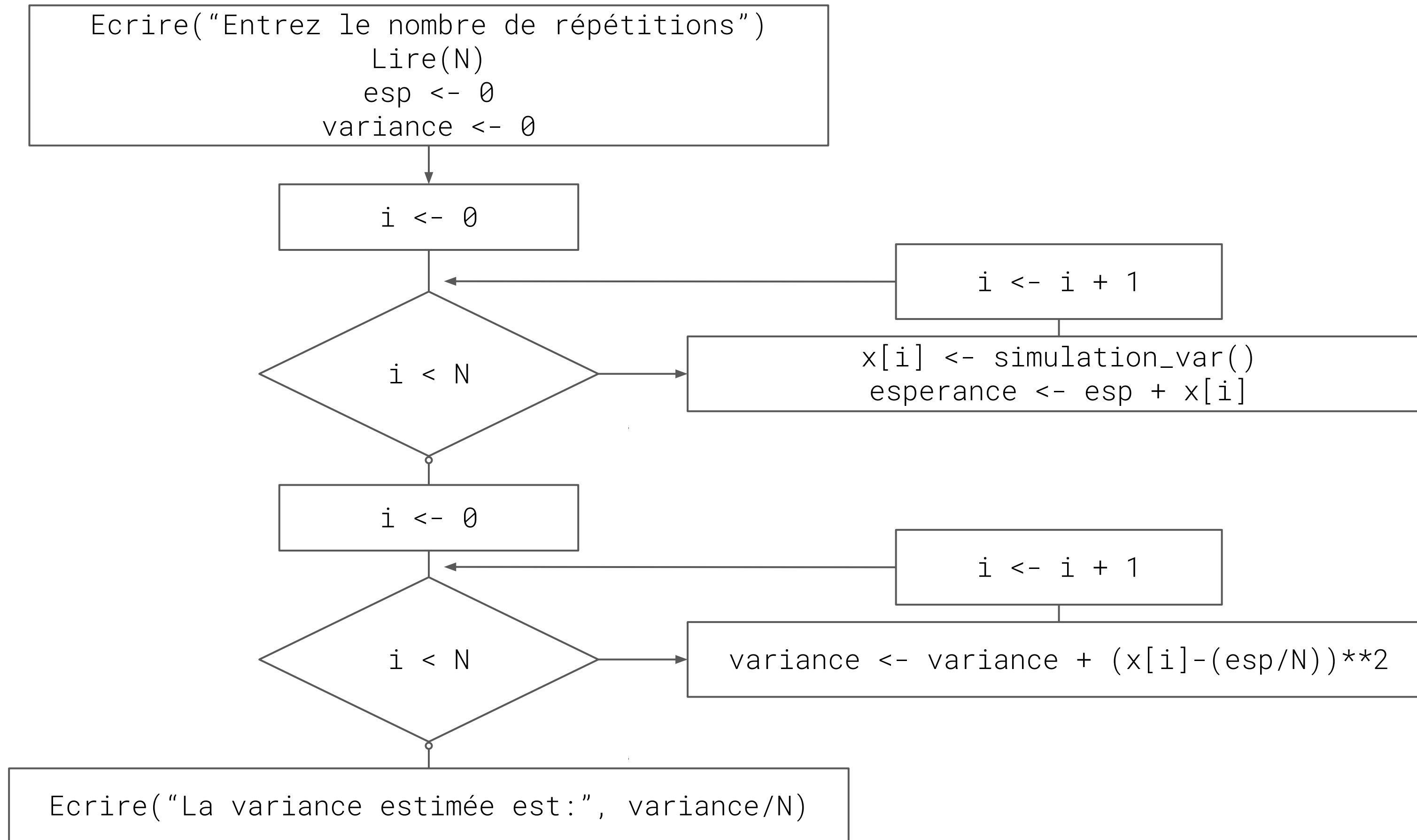
- La variance estimée.

Estimation de la variance: un algorithme

Actions:

- Simuler N fois un résultat de X en utilisant la procédure de simulation et conserver les résultats;
- Estimer l’espérance;
- Calcule de l’écart quadratique de la valeur de X avec l’espérance estimée;
- Sommer tous les écarts et les rapporter le tout à N.

Estimation de la variance: un algorithme



Travaux Pratiques

Proposer un schéma de résolution de l’algorithme précédent, ainsi que son expression littérale en pseudo-code et en Python.