L'essentiel du TD n°7

Simulation : approche fréquentielle

Simuler pour estimer, avec `random.random()` seul

1. Simuler avec `random.random()` : Bernoulli, dé
2. Estimer une probabilité : on compte
3. Quelle précision ?
4. Combiner des événements : `and`, `or`
5. Estimer une espérance : on fait la moyenne
6. Estimer une variance
7. Le piège : un tirage, un seul appel

1. Simuler avec `random.random()`

```
import random

random.random()          # 0.8710116635832919
random.random() < 0.3    # True avec proba 0.3

def bernoulli(p):
    if random.random() < p:
        return 1
    return 0

def de():
    return int(6 * random.random()) + 1

[de() for k in range(20)]   # [5, 2, 4, 5, ...]
```

À retenir

- `random.random()` renvoie un réel u de $[0, 1[$, et $P(u \in [a, b[) = b - a$;
- $u < p \iff u \in [0, p[$, intervalle de longueur p : probabilité p ;
- $\texttt{int}(6u) + 1 = k \iff u \in \left[\frac{k-1}{6}, \frac{k}{6}\right[$, de longueur $\frac{1}{6}$: le dé est équilibré ;
- deux appels donnent des résultats **indépendants**.

2. Estimer une probabilité : on compte

```
def frequence_six(n):
    k = 0
    for i in range(n):
        if de() == 6:
            k = k + 1
    return k / n

for n in [10, 100, 1000, 10000, 100000]:
    print(n, frequence_six(n))
# 10      0.0
# 100     0.25
# 1000    0.154
# 10000   0.172
# 100000  0.16756      (1/6 = 0.16666...)
```

À retenir

- **Loi des grands nombres** : on répète n fois une expérience, dans les mêmes conditions et de manière indépendante ; pour n grand, la **fréquence** de A est proche de $P(A)$;
- la fréquence est une valeur **aléatoire** : elle change à chaque exécution ;
- **à l'écrit** : donner la valeur exacte **et** citer la loi des grands nombres.

3. Quelle précision ?

```
for n in [100, 10000]:
    L = [frequence_six(n) for k in range(10)]
    print(n, min(L), max(L))
# 100     0.1     0.26
# 10000   0.1573  0.174
```

n	min et max (10 essais)	écart à $\frac{1}{6}$
100	0,1 et 0,26	$\approx$ 0,09
10 000	0,1573 et 0,174	$\approx$ 0,009

À retenir

- quand n est multiplié par 100, l'écart est divisé par environ 10 : il est de l'ordre de $\frac{1}{\sqrt{n}}$;
- gagner une décimale coûte 100 fois plus de calculs : on n'obtient pas beaucoup de décimales exactes.

Simulation ou calcul ?

Une simulation donne un ordre de grandeur ou **vérifie** un calcul. Elle ne remplace pas un calcul exact.

4. Combiner des événements : and, or

```
def somme_sept():
    return de() + de() == 7    # deux dés

def fonctionne(theta):
    c = []
    for k in range(3):     # c[0], c[1], c[2]
        c.append(random.random() < theta)
    return c[0] and (c[1] or c[2])

def frequence_appareil(theta, n):
    k = 0
    for i in range(n):
        if fonctionne(theta):
            k = k + 1
    return k / n

frequence_appareil(0.5, 100000)  # 0.37664
2 * 0.5**2 - 0.5**3              # 0.375
```

À retenir

- un test est déjà un **booléen** : pas besoin de `if ... return True` ;
- « C_1 et (C_2 ou C_3) » : $P\big(F_1 \cap (F_2 \cup F_3)\big) = 2\theta^2 - \theta^3$ (crible et indépendance).

Les parenthèses sont indispensables

`c[0] and c[1] or c[2]` se lit `(c[0] and c[1]) or c[2]` : on trouve 0,625 au lieu de 0,375.

Toujours comparer le résultat à la valeur exacte : un grand écart signale une erreur.

5. Estimer une espérance : on fait la moyenne

```
def attente_six():
    nb = 1                  # 1er lancer dans le test
    while de() != 6:
        nb = nb + 1
    return nb

def moyenne_attente(n):
    s = 0
    for i in range(n):
        s = s + attente_six()
    return s / n

moyenne_attente(100000)    # 6.04398   E(X) = 6
```

À retenir

- **Loi des grands nombres** : pour n grand, la **moyenne** de n valeurs de X simulées indépendamment est proche de $E(X)$;
- on ajoute la **valeur simulée**, pas 1 ;
- X : nombre de lancers pour obtenir 6. On a $P(X = k) = \left(\frac{5}{6}\right)^{k-1} \frac{1}{6}$ par indépendance ;
- nombre de lancers inconnu à l'avance : boucle `while` ; `nb` commence à 1.

6. Estimer une variance

```
def nombre_succes(m, p):
    s = 0
    for k in range(m):
        s = s + bernoulli(p)
    return s

def moyenne_variance(m, p, n):
    s1 = 0              # somme des X_i
    s2 = 0              # somme des X_i ** 2
    for i in range(n):
        x = nombre_succes(m, p)   # UNE fois
        s1 = s1 + x
        s2 = s2 + x**2
    moy = s1 / n
    return moy, s2 / n - moy**2

moyenne_variance(10, 0.3, 100000)
# (2.9937, 2.0925203099999994)
```

À retenir

- $V(X) = E(X^2) - E(X)^2$: on estime $E(X^2)$ par la moyenne des carrés ;
- le nombre de succès de m épreuves de Bernoulli suit la loi $\mathcal{B}(m, p)$: $E(X) = mp$ et $V(X) = mp(1-p)$. Ici : 3 et 2,1.

Une seule valeur de X par tour

On garde le résultat dans `x`. Avec deux appels à `nombre_succes`, on utiliserait deux valeurs différentes de X.

7. Le piège : un tirage, un seul appel

On veut $P(Y=1)=\frac{1}{2}$, $P(Y=2)=\frac{1}{3}$, $P(Y=3)=\frac{1}{6}$.

Faux

```
def tirage_faux():
    if random.random() < 1/2:      # u1
        return 1
    elif random.random() < 5/6:    # u2
        return 2
    else:
        return 3
```

À retenir : deux appels, deux nombres

- chaque appel donne un **nouveau** nombre : u_2 est indépendant de u_1 ;
- $Y=2$ si $u_1 \geqslant \frac{1}{2}$ **et** $u_2 < \frac{5}{6}$: $P(Y=2)=\frac{1}{2}\times\frac{5}{6}=\frac{5}{12}$;
- $Y=3$ si $u_1 \geqslant \frac{1}{2}$ **et** $u_2 \geqslant \frac{5}{6}$: $P(Y=3)=\frac{1}{2}\times\frac{1}{6}=\frac{1}{12}$.

Correct

```
def tirage():
    u = random.random()    # u
    if u < 1/2:            # [0, 1/2[
        return 1
    elif u < 5/6:          # [1/2, 5/6[
        return 2
    else:                  # [5/6, 1[
        return 3
```

À retenir : un appel, un seul nombre

- le **même** u est comparé aux deux seuils : on découpe $[0,1[$ en trois intervalles ;
- $Y=2$ si $u \in \left[\frac{1}{2}, \frac{5}{6}\right[$, de longueur $\frac{1}{3}$: $P(Y=2)=\frac{1}{3}$;
- $Y=3$ si $u \in \left[\frac{5}{6}, 1\right[$, de longueur $\frac{1}{6}$: $P(Y=3)=\frac{1}{6}$.

Le résumé : les deux schémas à connaître

Estimer $P(A)$: on compte.

```
def frequence(n):
    k = 0
    for i in range(n):
        if experience():   # True si A
            k = k + 1
    return k / n
```

Estimer $E(X)$: on fait la moyenne.

```
def moyenne(n):
    s = 0
    for i in range(n):
        s = s + simule_X()
    return s / n
```

Simuler avec `random.random()`

$P(u < p) = p$ · `int(6 * u) + 1` : dé équilibré · loi discrète : découper $[0, 1[$ en intervalles de longueurs les probabilités.

Les trois réflexes

Une expérience, **un seul** appel · **comparer** à la valeur exacte · à l'écrit : valeur exacte **et** loi des grands nombres.