

Variables aléatoires en Python (Part. 2)

Bibliothèque `numpy.random`

```
import numpy.random as rd
```

On a déjà vu que les instructions `rd.randint` et `rd.binomial` permettent de simuler des réalisations de variables aléatoires de loi uniforme et binomiale.

On va voir à présent comment construire d'une autre façon de telles variables aléatoires, en utilisant des instructions aléatoires !

Instructions aléatoires avec `rd.random`

La commande `rd.random()` génère un "nombre réel aléatoire" choisi dans l'intervalle $[0, 1]$. Ceci correspond à une réalisation d'une variable aléatoire U suivant la loi uniforme sur $[0, 1]$: $U \hookrightarrow \mathcal{U}([0, 1])$ (loi étudiée en détail l'année prochaine...)

`rd.random`

- `rd.random()` génère un nombre réel uniformément choisi dans $[0, 1]$.
- `rd.random(n)` génère un vecteur ligne de taille $1 \times n$ contenant de tels nombres aléatoires générés indépendamment.
- `rd.random([n,p])` génère un tableau de taille $n \times p$ contenant de tels nombres aléatoires générés indépendamment.

✎ Exercice 1

Après avoir importé la bibliothèque `numpy.random` :

1. Tester la commande `rd.random()` dans la console plusieurs fois d'affilée.

Quelques résultats obtenus :

2. Tester la commande `rd.random(5)` dans la console :

.....

On admet qu'une variable $U \hookrightarrow \mathcal{U}([0, 1])$ satisfait la propriété suivante :

$$\forall p \in [0, 1], \quad P(U \leq p) = P(U < p) = p.$$

Par exemple : • L'évènement $[U \leq 1]$ est réalisé avec probabilité 1 (logique!)

- $[U \leq \frac{1}{2}]$ est réalisé avec probabilité $\frac{1}{2}$, $[U \leq \frac{1}{3}]$ avec probabilité $\frac{1}{3}$ etc...

Ainsi, en quelque sorte :

$$\text{rd.random()} < p = \begin{cases} \text{True} & \text{avec probabilité } p \\ \text{False} & \text{avec probabilité } 1 - p \end{cases}$$

Effectuer une instruction "aléatoirement"

La syntaxe suivante permet de réaliser l'instruction 1 avec probabilité p ou bien l'instruction 2 avec probabilité $1 - p$:

```
if rd.random() < p :
    instruction1
else :
    instruction2
```

Construction "à la main" de la loi de Bernoulli

✎ Exercice 2

1. En s'inspirant de l'idée donnée dans l'encadré précédent, définir une fonction `bernoulli` qui prend en entrée un réel $p \in [0, 1]$ et simule une réalisation d'une variable aléatoire de loi $\mathcal{B}(p)$ (elle devra donc renvoyer 0 ou 1).

```
import numpy.random as rd
def bernoulli(p) :
```

2. Tester l'instruction `bernoulli(1/3)` 10 fois d'affilée dans la console.

Résultats obtenus :

Proportion de "1" obtenus sur ces 10 réalisations :

3. Après un grand nombre d'appels de l'instruction `bernoulli(1/3)`, on s'attend bien-sûr à ce que la proportion de "1" obtenus se rapproche de $1/3$... Compléter le programme pour afficher cette proportion sur $N = 1000$ appels.

```
N = 1000; S = ...


for k in range(N) :

    S = S + .....

print("Proportion de 1 :", .....
```

Proportion de "1" sur 1000 appels : Sur 10000 appels :

Construction "à la main" de la loi binomiale

 **Exercice 3**

Rappel : Une variable aléatoire de loi $\mathcal{B}(n, p)$ compte le nombre de succès lors de n répétitions indépendantes d'une épreuve de Bernoulli de paramètre p .

1. A partir de ce rappel, compléter la fonction `binomiale` pour qu'elle simule la réalisation d'une variable aléatoire de loi $\mathcal{B}(n, p)$.

```
def binomiale(n,p) :  
  
    x = 0  
    for k in range( ..... ) :  
  
        if ..... :  
  
            x = x + 1  
  
    return(x)
```

2. Pour un vecteur ou une liste `L` et un nombre `x`, on admet que :

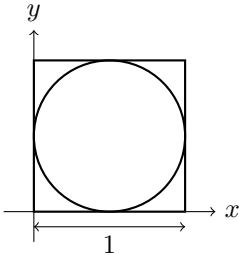
L'instruction `np.sum(L < x)` renvoie

.....

En déduire une façon plus rapide de simuler une loi $\mathcal{B}(n, p)$:

```
import numpy as np  
  
def binomiale(n,p) :  
  
    x = .....  
  
    return(x)
```

Approximation de π (méthode de Monte-Carlo)



On considère le carré plein de côté 1 donné par l'ensemble :

$$C = \{(x, y) \in \mathbb{R}^2 \mid 0 \leq x, y \leq 1\} = [0, 1] \times [0, 1] = [0, 1]^2$$

On considère le disque de centre $(\frac{1}{2}, \frac{1}{2})$ et de rayon $\frac{1}{2}$ inscrit dans ce carré :

$$D = \left\{ (x, y) \in \mathbb{R}^2 \mid \left(x - \frac{1}{2}\right)^2 + \left(y - \frac{1}{2}\right)^2 \leq \left(\frac{1}{2}\right)^2 \right\}$$

On dit qu'un point M est choisi uniformément au hasard dans le carré C si :

$M = (X, Y)$ avec $X \hookrightarrow \mathcal{U}([0, 1])$ et $Y \hookrightarrow \mathcal{U}([0, 1])$ (avec X et Y indépendantes).

On admet que la probabilité qu'un tel point M tombe dans le disque D est alors :

$$P(M \in D) = \frac{\text{Aire}(D)}{\text{Aire}(C)} = \dots\dots\dots$$

On propose de générer $N = 10^4$ points choisis uniformément dans le carré C , puis d'afficher la proportion de points qui appartiennent au disque. Cete proportion sera une approximation de la probabilité théorique $P(M \in D)$. (Loi des grands nombres)

```
N = 10**4 ; compteur = 0  
  
for k in range(N) :  
  
    # On genere un point M = (x,y) aleatoire  
  
    x = ..... ; y = .....  
  
    # Si le point est dans le disque, on augmente le compteur  
  
    if ..... :  
        compteur = compteur + 1  
  
proportion = ..... ; approx_pi = .....  
  
print(approx_pi)
```

- Approximation de π obtenue pour $N = 10^4$:
- Approximation de π obtenue pour $N = 10^5$:
- Approximation de π obtenue pour $N = 10^6$:
- Approximation de π obtenue pour $N = 10^7$: