Simulation de lois - Méthode d'inversion

1 Simulation de variables à densité

En Python, on a déjà vu que l'on pouvait simuler une loi de probabilité, notamment dans le cas de variables aléatoires discrètes. Dans ce qui suit, on va s'intéresser à la simulation de variables aléatoires à densité. A noter que l'on aura importé la bibliothèque `numpy.random` au préalable pour pouvoir réaliser des simulations. Commençons par rappeler la :

Définition 1.1 *La commande* `rd.random` *permet de créer un nombre réel aléatoire choisi dans* $[0,1]$.

En d'autres termes, la commande `rd.random` permet de simuler une variable aléatoire suivant la loi uniforme sur $[0,1]$, puisque pour tous réels $x, y \in [0,1]$ tels que $x < y$, la probabilité que le réel choisi par la commande `rd.random` appartienne à $[x,y]$ est égale à $y - x$. Plus généralement, pour simuler une variable aléatoire suivant la loi uniforme sur un intervalle $[a,b]$, on utilisera la commande suivante :

```
x=a+(b-a)*rd.random()
```

Pour obtenir p simulations (indépendantes) d'une variable uniforme sur $[a,b]$, on pourra se servir de la commande `x=a+(b-a)*rd.random(p)`. Plus généralement, si l'on veut obtenir une matrice de taille $p \times q$ dont les entrées sont des simulations d'une variable uniforme sur $[a,b]$, on utilisera la commande `x=a+(b-a)*rd.random([p,q])`. Enfin, on dispose aussi des commandes ci-dessous pour simuler d'autres lois continues classiques :

— La commande `rd.exponential(1/lambda)` donne une simulation d'une variable aléatoire qui suit la loi $\mathcal{E}(\lambda)$. A noter que le paramètre λ dans le système anglo-saxon (celui utilisé par Python) est l'inverse du paramètre λ dans le système français. De plus, la commande `rd.exponential(1/lambda,p)` donne une matrice ligne de longueur p dont les entrées suivent la loi $\mathcal{E}(\lambda)$, et la commande `rd.exponential(1/lambda,[p,q])` une matrice de taille $p \times q$ dont les entrées suivent la loi $\mathcal{E}(\lambda)$.

— La commande `rd.normal(m,s)` donne une simulation d'une variable aléatoire qui suit la loi $\mathcal{N}(m, s^2)$. A noter que le deuxième paramètre de cette loi dans le système anglo-saxon est l'écart-type s, alors que c'est la variance s^2 dans le système français. De plus, la commande `rd.normal(m,s,p)` donne une matrice ligne de longueur p dont les entrées suivent la loi $\mathcal{N}(m, s^2)$, et la commande `rd.normal(m,s,[p,q])` une matrice de taille $p \times q$ dont les entrées suivent la loi $\mathcal{N}(m, s^2)$.

— La commande `rd.gamma(nu)` donne une simulation d'une variable aléatoire qui suit la loi $\gamma(\nu)$. Plus généralement, la commande `rd.gamma(nu,1,p)` donne une matrice ligne de longueur p dont les entrées suivent la loi $\gamma(\nu)$, et la commande `rd.gamma(nu,1,[p,q])` donne une matrice de taille $p \times q$ dont les entrées suivent la loi $\gamma(n)$. Cette subtilité avec le 1 qui se rajoute dans les paramètres vient de ce que la loi $\gamma(\nu)$ est un cas particulier de la loi $\Gamma(b,\nu)$ (qui est hors-programme), au sens où $\Gamma(1,\nu) = \gamma(\nu)$.

Exemple 1.2 *Si l'on veut simuler une variable aléatoire de la forme* $Z = X + Y$*, où* X, Y *sont indépendantes et suivent toutes deux la loi uniforme sur* $[a,b]$*, on pourra utiliser la fonction suivante :*

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

def exemple1(a,b):
    x=a+(b-a)*rd.random(2)
    y=np.sum(x)
    return y
```

2 La méthode d'inversion

Dans ce paragraphe, on va voir une méthode permettant de simuler une variable aléatoire X ne suivant pas a priori une loi classique, si l'on connaît au préalable sa fonction de répartition F. Ce procédé est mieux connu sous le nom de *méthode d'inversion*. Pour une variable aléatoire discrète, cette méthode repose sur le résultat suivant :

Théorème 2.1 *Soit X une variable aléatoire discrète de support $X(\Omega) = \{x_i\}_{i\in I}$, où I est une partie non vide (finie ou infinie) de $\mathbb{N}$ et où les x_i sont classés dans l'ordre croissant. Si F est la fonction de répartition de X et si U est une variable aléatoire suivant la loi uniforme sur $[0,1]$, alors :*

$$P([X = x_1]) = P([U \leq F(x_1)]) \quad et \ : \ \forall k \geq 2, \ P([X = x_k]) = P([F(x_{k-1}) < U \leq F(x_k)]).$$

Dans le cas discret, le principe de la méthode d'inversion consiste alors à simuler la variable aléatoire U (à l'aide de la commande `rd.random`), puis à déterminer avec un test `if` ou une boucle `while` l'unique entier k tel que $F(x_{k-1}) < U \leq F(x_k)$. Une fois l'entier k trouvé, il suffit de renvoyer la valeur x_k pour simuler la variable aléatoire X.

Exemple 2.2 *Supposons que l'on veuille simuler une variable aléatoire X suivant la loi de Bernoulli de paramètre p donné au départ. Par définition, on voit que $F(0) = 1 - p$ et $F(1) = 1$, où F est la fonction de répartition de X. Dès lors, la simulation de X pourra passer par la fonction suivante :*

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

def exemple2(p):
    r=rd.random()
    if r>1-p:
        return 1
    else:
        return 0
```

Exemple 2.3 *Soit p un élément de $]0, \frac{1}{2}[$. Supposons que l'on veuille simuler une variable aléatoire X à valeurs dans $\{1,2,3\}$, dont la loi est donnée par $P([X = 1]) = 1 - 2p$, $P([X = 2]) = p$ et $P([X = 3]) = p$. Si F est la fonction de répartition de X, alors on trouve que $F(1) = 1 - 2p$, $F(2) = 1 - p$ et $F(3) = 1$. Dès lors, la simulation de X passera par la fonction suivante :*

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

def exemple3(p):
    r=rd.random()
    if r>1-p:
        return 3
    elif r>1-2p:
        return 2
    else:
        return 1
```

Pour une variable aléatoire à densité, la méthode d'inversion repose sur le résultat suivant :

Théorème 2.4 *Soit X une variable aléatoire à densité, de fonction de répartition F. Si U est une variable aléatoire suivant la loi uniforme sur $[0,1]$, alors on a pour tout $x \in \mathbb{R}$:*

$$F(x) = P([U \leq F(x)]).$$

Dans le cas d'une variable aléatoire à densité, le principe de la méthode d'inversion consiste d'abord à calculer la fonction de répartition F de X, puis à exprimer la probabilité $P([U \leq F(x)])$ sous la forme $P([G(U) \leq x])$, où G est une fonction à déterminer. Dès lors, la variable aléatoire $G(U)$ suivra la même loi que X, et on obtiendra une simulation de X d'abord en simulant la variable aléatoire U (à l'aide de la commande `rd.random`), puis en lui appliquant la fonction G.

Exemple 2.5 *Supposons que l'on veuille simuler une variable aléatoire X suivant la loi exponentielle de paramètre λ donné au départ. Pour ce faire, on commence par rappeler que la fonction de répartition F de X est donnée par :*

$$F(x) = \begin{cases} 0 & si \quad x < 0 \\ 1 - e^{-\lambda x} & si \quad x \geq 0 \end{cases} .$$

Si $x \geq 0$, alors on voit avec la méthode d'inversion que :

$$\begin{aligned} F(x) &= P([U \leq 1 - e^{-\lambda x}]) \\ &= P([e^{-\lambda x} \leq 1 - U]) \\ &= P([-\lambda x \leq \ln(1-U)]) \\ &= P\left(\left[-\frac{\ln(1-U)}{\lambda} \leq x\right]\right). \end{aligned}$$

De plus, comme la variable aléatoire $-\frac{\ln(1-U)}{\lambda}$ ne prend que des valeurs ≥ 0, on constate que, si $x < 0$:

$$F(x) = 0 = P\left(\left[-\frac{\ln(1-U)}{\lambda} \leq x\right]\right).$$

En résumé, la variable aléatoire $-\frac{\ln(1-U)}{\lambda}$ suit la même loi que X. Dès lors, pour réaliser une simulation de la loi exponentielle de paramètre λ, on pourra utiliser la fonction suivante :

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

def exemple5(lambda):
    u=rd.random()
    x=-np.log(1-u)/lambda
    return x
```