Niveau 2 - Méthodes Numériques
Cours 1

Jean-François Berger-Lefébure

17 Septembre 2024

Contents

1 Formules à connaître pour les lois de probabilité

1.1 Loi uniforme

La loi uniforme continue sur l'intervalle $[a, b]$, notée $U(a, b)$, a la fonction de densité suivante :

$$f_U(x) = \frac{1}{b-a} \quad \text{pour} \quad a \leq x \leq b.$$

La fonction de répartition associée est :

$$F_U(x) = \frac{x-a}{b-a} \quad \text{pour} \quad a \leq x \leq b.$$

Exemple d'utilisation pour générer une variable aléatoire :
Soit $U \sim \mathcal{U}(0, 1)$, une variable aléatoire uniforme sur l'intervalle $[0, 1]$, pour générer une variable X suivant la loi uniforme $U(a, b)$, il suffit d'appliquer la transformation :

$$X = a + (b-a) \cdot U$$

Cela permet de simuler X avec $U \sim \mathcal{U}(0, 1)$.

1.2 Loi exponentielle

La loi exponentielle avec un paramètre $\lambda > 0$, notée $X \sim \mathcal{E}(\lambda)$, a la fonction de densité suivante :

$$f_X(x) = \lambda e^{-\lambda x} \quad \text{pour} \quad x \geq 0.$$

La fonction de répartition associée est :

$$F_X(x) = 1 - e^{-\lambda x} \quad \text{pour} \quad x \geq 0.$$

Exemple d'utilisation pour générer une variable aléatoire :
Soit $X \sim \mathcal{E}(\lambda)$, pour générer une variable X suivant la loi exponentielle avec un paramètre λ, on applique la transformation suivante à une variable uniforme $U \sim \mathcal{U}(0, 1)$:

$$X = -\frac{1}{\lambda} \ln(1-U)$$

Cela permet de simuler X suivant la loi exponentielle.

1.3 Loi de Bernoulli

La loi de Bernoulli avec un paramètre $p \in [0, 1]$, notée $X \sim \mathcal{B}(p)$, a la fonction de densité suivante :

$$f_X(x) = p^x (1-p)^{1-x} \quad \text{pour} \quad x \in \{0, 1\}.$$

La fonction de répartition associée est :

$$F_X(x) = \begin{cases} 0, & \text{si} \quad x < 0, \\ 1-p, & \text{si} \quad 0 \leq x < 1, \\ 1, & \text{si} \quad x \geq 1. \end{cases}$$

Exemple d'utilisation pour générer une variable aléatoire :
Soit $X \sim \mathcal{B}(p)$, pour générer une variable X suivant la loi de Bernoulli avec un paramètre p, on applique la transformation suivante à une variable uniforme $U \sim \mathcal{U}(0, 1)$:

$$X = \begin{cases} 1, & \text{si} \quad U \leq p, \\ 0, & \text{si} \quad U > p. \end{cases}$$

Cela permet de simuler X suivant la loi de Bernoulli.

2 Transformation des variables aléatoires

Pour générer des variables aléatoires suivant des lois de probabilité spécifiques à partir de variables uniformes, on utilise la méthode d'inversion de la fonction de répartition. Ci-dessous, on montre comment effectuer cette transformation pour diverses lois de probabilité.

2.1 Loi uniforme

- Si $U \sim \mathcal{U}(0,1)$, alors une transformation simple est :

$$X = F_X^{-1}(U)$$

où F_X^{-1} est l'inverse de la fonction de répartition de la loi cible.

- Le code pour simuler une loi uniforme $U \sim \mathcal{U}(0,1)$ est :

```
U0 = np.random.random(n)  # Variable uniforme [0,1]
```

2.2 Loi exponentielle

- Si $X \sim \mathcal{E}(\lambda)$, la transformation est :

$$X = -\frac{1}{\lambda} \ln(1-U)$$

- Le code Python utilisé pour générer X suivant une loi exponentielle avec paramètre $\lambda = 1$ est :

```
X = -np.log(1 - U0)  # Loi exponentielle, lambda=1
```

2.3 Loi Bernoulli

- Pour générer une variable suivant une loi Bernoulli $U \sim$ Bernoulli(p), on transforme une variable uniforme $U_0 \sim \mathcal{U}(0,1)$ par :

$$U = \begin{cases} 1, & \text{si } U_0 < p \\ 0, & \text{sinon} \end{cases}$$

Ce code permet de générer une variable Bernoulli :

```
U = (U0 < p).astype(int)  # Transformation en variable Bernoulli
```

2.4 Loi normale

- Si $Z \sim \mathcal{N}(0,1)$, la méthode d'inversion implique l'utilisation de la fonction de répartition inverse d'une normale standard $Z = F_Z^{-1}(U)$, où $F_Z^{-1}(U)$ peut être obtenu par des fonctions comme `ppf` dans `scipy.stats`.

- Le code Python pour simuler une loi normale standard est :

```
Z = stats.norm.ppf(U0)  # Simulation d'une loi normale
```

2.5 Loi binomiale

- Pour une loi binomiale $B(n,p)$, la transformation en utilisant une méthode d'inversion (approximée) consiste à générer un certain nombre de succès en comparant les valeurs uniformes à la probabilité p.

- Le code Python pour générer une loi binomiale est :

```
X = np.random.binomial(n, p, size=n)
```

3 Exercice 2

Exercice 2. *Estimer via Monte Carlo les quantités suivantes :*

— $\mathbb{E}[\sin(U)]$ *où* $U \sim \mathcal{U}([0,\pi])$

— $\mathbb{E}[\log(X)]$ *où* $X \sim \mathcal{E}(1)$

— $\mathbb{E}[(e^{-\frac{\sigma^2}{2}+\sigma Z}-1)^+]$ *où* $Z \sim \mathcal{N}(0,1)$ *et* $\sigma = 0.2$

On n'utilisera que des simulations de loi uniforme pour cet exercice. On utilisera $n = 10^6$ *et on donnera un intervalle de confiance pour chacune des estimations.*

Pour simuler une loi $\mathcal{U}([0,1])$*, dans Python on utilisera la fonction* ***random.random(n)*** *du package* ***numpy****, dans R on utilisera la fonction* ***runif(n)*** *où* n *est le nombre de simulations.*

On pourra afficher les histogrammes de U*,* X *et* Z*.*

3.1 Codes

```
###Exercice 2 - Méthode d'inversion de la f.d.r.

n = 10**6   # Nombre de simulations (1 million)
#quantile pour l'erreur / intervalle de confiance au niveau de confiance 95%
q = stats.norm.ppf(0.975)
#Génère 'n' valeurs aléatoires uniformément distribuées dans l'intervalle [0, 1]
U0 = np.random.random(n)

#Estimation de IE(sin(U)) avec U ~ U([0, Ï€])
U = np.pi*U0    # Transformation des valeurs de U pour les amener dans l'intervalle [0, π]
sinU = np.sin(U)    # Calcul du sinus de chaque valeur de U.
e1 = np.mean(sinU)    # Estimation de l'espérance de sin(U).
sd1 = np.std(sinU)    # Estimation de l'écart-type de sin(U).
print("Estimateur MC : %s Â± %s" % (e1, q*sd1/np.sqrt(n)))
```

```
Estimateur MC : 0.6365644509966977 Â± 0.0006028052719652003
```

```
#Estimation de IE(log(X)) avec X ~ Exp(1)
X = -np.log(U0)
logX = np.log(X)
e2 = np.mean(logX)
sd2 = np.std(logX)
print("Estimateur MC : %s Â± %s" % (e2, q*sd2/np.sqrt(n)))
```

```
Estimateur MC : -0.5773647694475849 Â± 0.002510726750230508
```

```
#Estimation de IE((exp(-s^2/2 + sZ)-1)+) avec Z ~ N(0,1)
sigma = 0.2
Z = stats.norm.ppf(U0)
Y = np.exp(-sigma*sigma*0.5 + sigma*Z) - 1.
Y[Y < 0.] = 0.

e3 = np.mean(Y) # 2.*stats.norm.cdf(0.5*sigma) - 1.
sd3 = np.std(Y)
print("Estimateur MC : %s Â± %s" % (e3, q*sd2/np.sqrt(n)))
```

```
Estimateur MC : 0.07963403773737804 Â± 0.002515114096438687
```