Université Sorbonne Paris Nord, Institut Galilée
Année universitaire 2020–2021

MACS2 – Processus stochastiques
mallein@math.univ-paris13.fr

TP de Processus stochastiques – Simulations, vecteurs gaussiens

1 Simulations de variables aléatoires

On rappelle ici quelques méthodes utiles pour la simulation de variables aléatoires. Ces méthodes seront par la suite revues et étendues pour la simulation de processus stochastique, il est donc nécessaire de bien les connaître. On utilisera ici le langage de programmation *Python*, la bibliothèque *NumPy* pour les opérations mathématiques et simulations de variables aléatoires, et la bibliothèque *Matplotlib* pour la visualisation des résultats. Tous les programmes commenceront donc par les lignes de commande suivantes.

```
import numpy as np                # operations mathematiques
import numpy.random as npr        # simulation d'alea
import matplotlib.pyplot as plt   # plot
```

On commencera par étudier la simulation de variables aléatoires de loi uniforme.

Exercice 1 (Variables aléatoires uniformes)**.** 1. Exécuter plusieurs fois le code ci-dessous. Expliquer les résultats obtenus.

```
print(npr.rand())
npr.seed(seed=1)
print(npr.rand())
print(npr.rand())
npr.seed(seed=1)
print(npr.rand())
print(npr.rand())
```

2. La fonction `npr.rand` peut prendre des argument optionnels. Que renvoit `npr.rand(10)` ? `npr.rand(5,3)` ? `npr.rand(2,3,4)` ?
3. Exécuter le code suivant et commenter les résultats.

```
X = npr.rand(1000)
plt.hist(X, bins = 20, range = (0,1), density = True, color = 'blue')
plt.xlabel('Domaine')
plt.ylabel('Densité')
plt.title('Histogramme de X')
plt.show()
```

Remarque 1. La fonction `npr.rand` renvoie les valeurs successives d'une suite déterministe possédant les propriétés d'une suite de variables aléatoires i.i.d. de loi uniforme. Le choix du rang initial de la suite est fixé avec la fonction `seed`, par défaut ce nombre est lié à une quantité comme le nombre de micro-secondes écoulées depuis une date arbitraire. Pour plus d'informations, on pourra se renseigner sur cet algorithme, appelé "Mersenne Twister" (développé par Makoto Matsumoto et Takuji Nishimura en 1997).

Dans tout ce qui suivra, on traitera les appels à `npr.rand` comme des variables aléatoires indépendantes et identiquement distribuées de loi uniforme. On notera l'existence de générateurs certifiés de nombres aléatoires (c.f. random.org), qui peuvent avoir leur utilité sous certaines conditions.

On introduit deux méthodes principales permettant de simuler des échantillons de variables aléatoires. La première ne fonctionne que dans certains cas, et s'appelle la méthode de la fonction de répartition.

Exercice 2 (Méthode de la fonction de répartition)**.** Soit X une variable aléatoire à densité par rapport à la mesure de Lebesgue, on note F la fonction de densité de X, et F^{-1} la fonction inverse de F.

1. Montrer que la variable aléatoire $F(X)$ suit une loi uniforme sur $[0,1]$.
2. En déduire que si U est de loi uniforme, $F^{-1}(U)$ a même loi que X.
3. Rappeler la fonction de répartition de la loi exponentielle de paramètre λ. En déduire une méthode de génération de variables aléatoires de loi exponentielle.
4. Construire la fonction `loiExpo(n, lbda)` retournant un tableau NumPy de `n` variables aléatoires indépendantes de loi exponentielle de paramètre `lbda`. On pourra utiliser la fonction `np.log`.
5. Vérifier la correction de ce programme en exécutant le programme suivant :

```
X = loiExpo(1000,1)
plt.hist(X, bins = 20, range = (0,5), density = True, color = 'blue')
position = np.linspace(0,5,200)
densite = np.exp(-position)
plt.plot(postition,densite,color="red")
plt.xlabel('Domaine')
plt.ylabel('Densité')
plt.title('Histogramme de X')
plt.show()
```

6. ($\star$) Définir la fonction `loiCauchy(n,a,b)` renvoyant une variable aléatoire ayant pour densité $\frac{a}{\pi((x-b)^2+a^2)}$. Tester cette fonction grâce à la même méthode que précédemment.

La deuxième méthode, appelée méthode de rejet, permet d'éviter le parfois coûteux calcul de la fonction de répartition de la variables aléatoire, au prix d'un temps d'exécution parfois plus long.

Exercice 3 (Méthode de rejet)**.** On considère une loi de probabilité ayant une densité f par rapport à la mesure de Lebesgue, dont on ne sait pas calculer la fonction de répartition. On suppose qu'il existe une constante $c > 0$ et une fonction de densité g telle que $f \leq cg$, et telle qu'on sait simuler des variables aléatoires dont la loi a densité g.

1. Montrer que nécessairement, on a $c \geq 1$.
2. Soit $(Y_n, n \geq 1)$ une suite de v.a. i.i.d. ayant pour densité g, et $(U_n, n \geq 1)$ une suite de variables aléatoires i.i.d. de loi uniforme sur $[0,1]$. On pose $T = \inf\{n \geq 1 : U_n \leq f(Y_n)/cg(Y_n)\}$.
   (a) Déterminer la loi de T.
   (b) Montrer que la variable Y_T a pour densité la fonction f.
3. En déduire une méthode permettant de simuler des variables aléatoire ayant la densité f.
4. Application : construire une fonction `demicercle`, qui permet de simuler une variable aléatoire ayant pour densité la fonction $\frac{2}{\pi}\sqrt{1-x^2}\mathbf{1}_{\{x\in[-1,1]\}}$.

Exercice 4 (Mise en application ($\star$))**.** On définit la fonction

$$f(x) = \begin{cases} x+1 & \text{si } -1 < x < 0 \\ 1-x & \text{si } 0 \leq x < 1 \\ 0 & \text{sinon.} \end{cases}$$

1. Dessiner le graphe de la fonction f. Que vaut $\int_{\mathbb{R}} f(x)\mathrm{d}x$? Définissez la fonction `fonctionF`, qui associe à x la valeur de $f(x)$.
2. Construisez la fonction `methodeRejet` permettant de simuler des variables aléatoires de densité f grâce à la méthode de rejet.
3. Construisez la fonction `methodeFonctionRepartition` permettant de simuler des variables aléatoires de densité f grâce à la méthode de la fonction de répartition inverse.
4. Comparez la vitesse de ces deux algorithmes pour générer un échantillon de 100000 variables aléatoires ayant pour densité f.

2 Variables aléatoires de loi normale, vecteurs gaussiens

Il est fréquent de devoir créer des échantillons de variables aléatoires de loi normale, pour lesquels aucune des méthodes définies plus haut ne fonctionne très bien. Pour générer des variables de loi normale, on pourra utiliser la méthode de Box-Muller, définie ci-dessous. Cet exercice est un sujet d'entretient d'embauche très fréquent.

Exercice 5 (Loi normale, méthode de Box-Muller)**.** Soit (X, Y) un couple de variables i.i.d. de loi $\mathcal{N}(0, 1)$.

1. Déterminer la loi de $\sqrt{X^2 + Y^2}$.
2. Montrer que $\arctan(Y/X), \sqrt{X^2 + Y^2}$ forme une paire de variables indépendantes, dont on déterminera les lois.
3. En déduire une méthode permettant de simuler une variable aléatoire de loi normale centrée réduite.
4. Application : construire une fonction `gaussienne(mu,sigma2)`, qui permet de simuler une variable aléatoire de loi normale, dont on précise en paramètre la moyenne et la variance.
5. ($\star$) Quelle est la loi de $X^2 + Y^2$? La loi de Y/X ?

On passe ensuite à la simulation de vecteurs gaussiens, qui peut nécessiter quelques résultats d'algèbre linéaire sur lesquels on passera ici.

Exercice 6 (Vecteur gaussien, méthode de Cholesky)**.** 1. Tester la fonction `npr.normal`, quels sont ses paramètres ? Que retourne-t-elle ?

2. Soit $(X_1, \dots X_n)$ un échantillon de variables aléatoires i.i.d. de loi $\mathcal{N}(0, 1)$ et A une matrice $n \times n$. Quelle est la loi de $Y = AX$?
3. De façon réciproque, comment construire un vecteur gaussien de loi $\mathcal{N}(0, \Sigma)$, où Σ est une matrice symétrique positive ?
4. La factorisation de Cholesky permet de déterminer, pour toute matrice symétrique positive Σ une matrice triangulaire inférieure L tel que $LL^T = \Sigma$. Cette fonction est codée dans `np.linalg.cholesky`. Utiliser cette fonction pour construire la fonction `genererVecteur(Mu,Sigma,n)`, générant un échantillon de `n` vecteurs gaussiens de moyenne `Mu` et de matrice de covariance `Sigma`.
5. Utiliser cette fonction pour générer un échantillon de 1000 vecteurs aléatoires (X, Y) centrés tels que $\mathrm{Var}(X) = \mathrm{Var}(Y) = 1$ et $\mathrm{Cov}(X, Y) = 0.5$. Vérifier la justesse de cette fonction en calculant la covariance empirique de cette famille de vecteurs gaussiens.

On pourra enfin rappeler les méthodes d'estimations vues l'année dernière.

Exercice 7 (Estimation de paramètres, théorème de Cochran)**.** Soit $(X_1, \dots, X_n)$ un n-échantillon de variables aléatoires de loi $\mathcal{N}(\mu, \sigma^2)$, dont les deux paramètres μ et σ^2 sont inconnus.

1. Déterminer les estimateurs du maximum de vraisemblance $\hat{\mu}_n$ et $\hat{S^2}_n$ de (μ, σ^2), ainsi que l'estimateur non-biaisé $\hat{\sigma^2}_n$ de la variance.
2. Rappeler le théorème de Cochran. En déduire que $\hat{\mu}_n$ et $\hat{\sigma^2}_n$ sont indépendantes.
3. Exécutez le code suivant. Commenter les résultats.

```
X = npr.normal(size = (100,1000))
mu = np.zeros(100)
sigma2 = np.zeros(100)
for i in range(100):
    mu[i] = sum(X[i])/1000
    sigma2[i] = 1000/999*(sum(X[i]**2)/1000 - mu[i]**2)
plt.scatter(mu,sigma2)
plt.show()
```

4. Adaptez le code précédent pour étudier l'erreur statistique lors de l'estimation des paramètres d'une loi normale de paramètres $(100, 1)$, d'une loi normale de paramètre $(0.5, 50)$.

A Simulation de lois discrètes ($\star$)

On rappellera ici une méthode permettant de simuler des variables de loi discrète.

A.1 Loi de Bernoulli

La loi de probabilité la plus élémentaire à simuler est la loi de Bernoulli de paramètre p. On montre sur cet exemple comment utiliser les nombres pseudo-aléatoires, de loi uniforme, pour simuler ces variables.

Exercice 8 (Loi de Bernoulli)**.** Soit U une variable aléatoire de loi uniforme sur $[0,1]$ et $p \in [0,1]$.

1. Quelle est la loi de $X = \mathbf{1}_{\{U<p\}}$?
2. Complétez le code ci-dessous pour que la fonction `bernoulli(p)` retourne à chaque appel une variable aléatoire de loi Bernoulli de paramètre p.

```
def bernoulli(p):
    if npr.rand() < p:
        #BLA
    else:
        #BLA

print(bernoulli(p))
print(bernoulli(p))
print(bernoulli(p))
```

On peut modifier la définition de la fonction `bernoulli` pour qu'elle tienne sur une ligne, avec des fonctions `lambda`, mais nous ne nous plongerons pas dans ces méthodes de définition alternative.

Exercice 9 (Suite de variables i.i.d.)**.**

1. Que retourne la commande `npr.rand(10) < 0.5` ? Et `(npr.rand(10)<0.5)*1` ? Expliquez.
2. Modifiez la fonction `bernoulli` pour qu'elle prenne en arguments deux nombres $p \in (0,1)$ et $n \in \mathbb{N}$ et retourne un tableau de n variables de loi de Bernoulli de paramètre p.
3. Utiliser la fonction `plt.hist` pour vérifier la loi de ces variables.

A.2 Lois à support fini

Exercice 10 (Loi binomiale)**.** On cherche à modéliser une variable aléatoire de loi binomiale de paramètres n et p.

1. Proposer une première fonction `binomiale1` basée sur la méthode de la fonction de répartition pour simuler une variable de loi binomiale de paramètres `n` et `p`.
2. En utilisant qu'une binomiale est une somme de variables de Bernoulli i.i.d. proposer une deuxième fonction `binomiale2` pour simuler cette loi.
3. Comparer la vitesse d'exécution de ces deux fonctions en utilisant la fonction `time` du module `time`.
4. En utilisant le théorème central limite, proposer une troisième façon de simuler une variable de loi binomiale, en utilisant la fonction `npr.randn` qui simule des variables aléatoires de loi normale centrée réduite, fonctionnant pour de grandes valeurs de n.

Exercice 11 (Loi uniforme discrète)**.** Pour construire une variable aléatoire de loi uniforme sur $\{1,2,\dots,n\}$, on vérifiera que le code suivant fonctionne, et on expliquera son fonctionnement.

```
def loiUnifDiscrete(nb,N):
    return 1+np.floor(npr.rand(nb)*N)
```

A.3 Loi à support infini

Dans les cas de lois discrète à support infini, on peut utiliser dans certain cas des propriétés ad-hoc de la loi pour permettre de la simuler simplement.

Exercice 12 (Loi géométrique)**.** On cherche à modéliser la loi géométrique de paramètre p.

1. En utilisant qu'une variable aléatoire de loi géométrique est le temps de première réussite d'une expérience ayant probabilité p de réussir, construire une première fonction `geometrique1` qui simule une variable aléatoire de loi géométrique.
2. Proposer une seconde méthode utilisant le calcul de la fonction de répartition de la loi géométrique.

Exercice 13 (Loi de Poisson)**.** Soit $(U_n, n \geq 0)$ une suite de variables aléatoires indépendantes de loi uniforme sur $[0,1]$ et $\lambda > 0$. On pose

$$T = \inf\{n \geq 0 : (-\log(U_0)) + \cdots + (-\log U_n) \geq \lambda\}.$$

1. Déterminer la loi de T.
2. Proposer une méthode de simulation d'une loi de Poisson de paramètre λ.

Exercice 14 (Loi générique)**.** Soit U une variable aléatoire de loi uniforme sur $[0,1]$ et μ une distribution de probabilité sur $\{0, 1, \dots N\}$.

1. Construisez une fonction mesurable f telle que $f(U)$ est une variable aléatoire de loi μ.
2. Créez une fonction `simulerVA` qui prend en entrée une liste `mu`, et qui retourne en sortie une variable aléatoire Z telle que $\mathbf{P}(Z = i) =$`mu[i]`.
3. Modifiez votre fonction pour que `simulerVA` permette de simuler un tableau de copies indépendantes de Z.
4. Vérifiez que votre fonction fonctionne correctement en utilisant la fonction `plt.hist`.
5. Exécutez le code suivant. Quel théorème du cours illustre-t-il ?

```
mu = [0.1,0.2,0.2.0.4,0.1]
N=10000

X = simulerVA(mu,N)
print(sum(X)/N)
```

6. Exécutez le code suivant. Quel théorème du cours illustre-t-il ? Essayez de modifier les paramètres utilisés.

```
mu = [0.1 ,0.2 ,0.2,0.4 ,0.1]
esp = sum([i*mu[i] for i in range(len(mu))])
# L ' espérance d'une variable de loi mu est esp
N =10000
n=1000

Y = np.zeros(N)
for j in range(N):
    X = simulerVA(mu,n)
    Y[j] = (sum(X) - n*esp)/sqrt(n)

plt.hist (Y, bins = 20, density = True)
plt.xlabel ('Domaine')
plt.ylabel ('Densité')
plt.show ()
```

7. Définir la fonction `simuVA` simulant une variable aléatoire Y à valeurs dans $\mathbb{N}^*$ telle que $\mathbf{P}(Y = k) = \frac{1}{k(k+1)}$ pour tout $k \in \mathbb{N}^*$.