CPES 2 – Probabilités approfondies 2015-2016

TP de Python

Igor Kortchemski – igor.kortchemski@cmap.polytechnique.fr

1 Simulation de lois

1.1 Loi uniforme

Comment Python simule-t-il une suite de variables aléatoires indépendantes et uniformes sur $[0,1]$?

En quelques mots, cela revient à générer des entiers aléatoires indépendants uniformes dans $E = \{1, 2, \dots, M\}$, puis à les diviser par M. Pour cela, Python génère une suite de nombres pseudo-aléatoires en partant d'un état $x_0 \in E$ (la graine, ou seed en anglais) et en appliquant successivement une même fonction $f : E \to E$ (par exemple $f(x) = ax + c$ modulo M avec a et c bien choisis). Cette transformation doit bien sûr vérifier plusieurs propriétés : on veut que la suite générée « ressemble » à une suite de variables aléatoires uniformes et indépendantes sur E. Que veut dire « ressemble » ? La suite doit réussir à passer toute une batterie de tests statistiques d'indépendances et d'adéquation de loi. En particulier, la période, c'est-à-dire le plus petit entier k tel que $f^{(k)}(x) = x$ (avec $f^{(k)}$ l'itérée k-ième), doit être (très (très)) grande.

Python utilise l'algorithme appelé « Mersenne Twister » (développé par Makoto Matsumoto et Takuji Nishimura en 1997) possédant une période de $2^{19937} - 1$ (qui est un nombre premier de Mersenne).

Ainsi : existe-t-il des vrais générateurs de nombre aléatoire ?

NON : Pas vraiment, tous les générateurs sont déterministes ! !

MAIS : ils sont construits de telle sorte à passer les tests statistiques.

Question 0. Exécuter plusieurs fois le Code 0.

Code 0 – Question0.py

```
import numpy.random as npr

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

On suppose ainsi qu'on a à notre disposition une « boîte noire » qui permet de simuler une suite de variables aléatoires indépendantes et uniformes sur $[0,1]$.

Pour générer d'autres aléas que ceux de loi uniforme, on fait subir des transformations astucieuses (algorithme de simulation).

Exemple (simulation d'une variable uniforme sur un segment quelconque). Si $a < b$ et U est une variable aléatoire uniforme sur $[0,1]$, on peut démontrer que $a + (b-a)U$ suit une loi uniforme sur $[a,b]$. Ainsi, pour simpuler une variable aléatoire uniforme sur $[a,b]$, on simule une variable aléatoire U uniforme sur $[0,1]$ et on renvoie $a + (b-a)U$.

1.2 Simulation par inversion de la fonction de répartition

On a vu en cours que si U est uniforme sur $[0,1]$ et $\lambda > 0$, alors $V = -\frac{1}{\lambda}\ln(U)$ est une loi exponentielle de paramètre λ. Plus généralement, on a le résultat suivant.

Théorème 1. *Soit X une variable aléatoire réelle. On suppose que sa fonction de répartition F est strictement croissante (F est donc bijective de $\mathbb{R}$ sur $]0,1[$ et on peut noter F^{-1} son inverse). Soit U une variable aléatoire uniforme sur $[0,1]$. Alors $F^{-1}(U)$ a la même loi que X.*

Démonstration. Soit $x \in \mathbb{R}$. On calcule $\mathbb{P}(F^{-1}(U) \leq x) = \mathbb{P}(U \leq F(x)) = F(x)$ (car U est uniforme sur $[0,1]$ et $0 \leq F(x) \leq 1$). Donc $F^{-1}(U)$ a la même fonction de répartition que X. □

Remarque 2. *Si F n'est pas strictement croissante (et donc pas injective), le théorème précédent reste vrai à condition de* définir *$F^{-1}(u)$ comme $F^{-1}(u) = \inf\{x \in \mathbb{R} : F(x) \geq u\}$ (F^{-1} est appelé l'inverse continu à droite de F).*

Question 1. Simuler une variable aléatoire de Cauchy de paramètre 1 dont une densité est $\frac{2}{\pi}\frac{1}{1+x^2}$ en complétant le Code 1.

Code 1 – Question1.py

```
from math import *
import numpy.random as npr

def Cauchy():
    U=BLA #tire un reel uniforme sur [0,1]
    return tan(BLA) #On evalue l'inverse de la fonction de repartition en U

print Cauchy()
```

1.3 Loi géométrique

On va comparer plusieurs manières de simuler une variable géométrique de paramètre $p \in (0,1)$ à partir d'une variable aléatoire uniforme sur $[0,1]$ en se fondant sur les résultats théoriques suivants :

(1) De manière générale, pour simuler une variable aléatoire X à valeurs entières telle que $\mathbb{P}(X = i) = p_i$ pour tout $i \geq 0$, on tire une variable uniforme U sur $[0,1]$ et on renvoie l'entier k tel que $p_0 + \cdots + p_{k-1} < U < p_0 + \cdots + p_k$.

(2) On tire des variables aléatoires de Bernoulli de paramètre p (on renvoie 1 si une variable aléatoire uniforme U sur $[0,1]$ est plus petite que p, 0 sinon) et on s'arrête à la première fois qu'on tombe sur 1.

(3) On pose $\lambda = -\frac{1}{\ln(1-p)}$ et on renvoie $\lceil \lambda X \rceil$ ($\lceil x \rceil$ est le plus petit entier k tel que $k \geq x$, `ceil` en `Python`) où X est une variable aléatoire exponentielle de paramètre 1.

(4) Faire appel à une fonction intégrée de `Python`.

Question 2. Comparer l'efficacité de ces trois méthodes pour simuler N variables géométrique de paramètre p en complétant le Code 2. Commenter l'influence des paramètres.

Code 2 – Question2.py

```python
from __future__ import division
from math import *
import numpy as np
import numpy.random as npr
from time import time

p=0.1 #parametre de la geometrique

N=10000 #nombre de fois qu'on simule la geometrique

def methode1():
    k=1
    tmp=p
    U=npr.rand()
    while U>tmp:
        tmp=tmp+BLA
        k=k+1
    return BLA

def methode2():
    tmp=0
    k=0
    while tmp==0:
        k=k+1
        if BLA:
            tmp=1
    return k

def methode3():
    X=-log(npr.rand())
    return int(ceil(BLA))

def methode4():
    return npr.BLA(p)

t1 = time()
[methode1() for i in range(N)]
t2 = time()
temps1 = t2 - t1
print "La methode 1 a pris ", temps1, " secondes"

t1 = time()
[methode2() for i in range(N)]
t2 = time()
temps1 = t2 - t1
print "La methode 2 a pris ", temps1, " secondes"

t1 = time()
[methode3() for i in range(N)]
t2 = time()
temps1 = t2 - t1
print "La methode 3 a pris ", temps1, " secondes"
```

```
53
54
55 t1 = time()
56 [methode4() for i in range(N)]
57 t2 = time()
58 temps1 = t2 - t1
59 print "La methode 4 a pris ", temps1, " secondes"
```

2 Convergence de variables aléatoires

2.1 Approximation d'une loi de Poisson par une loi binomiale

Nous avons démontré le résultat suivant en cours.

Théorème 3. *Soit $\lambda > 0$. Soit X_n une variable aléatoire binomiale de paramètre $(n, \lambda/n)$. Alors X_n converge en loi vers une loi de Poisson de paramètre λ lorsque $n \to \infty$ (c'est-à-dire que pour tout $k \geq 0$, $\mathbb{P}(X_n = k) \to e^{-\lambda}\lambda^k/k!$ lorsque $n \to \infty$).*

Question 3. Illustrer ce théorème en complétant le Code 3.

Code 3 – Question3.py

```
1 from __future__ import division
2 import numpy as np
3 import numpy.random as npr
4 import scipy.stats as sps
5 import matplotlib.pyplot as plt
6
7 param=3 #parametre
8 n=100
9 N=5000 #nombre de tirages effectue pour tracer l'histogramme en batons de la loi de
      X_n
10
11 X=npr.binomial(BLA)
12
13 BLA
14 BLA
15 BLA
16 BLA
17 BLA
18 BLA
```

2.2 Approximation d'une loi exponentielle par une loi géométrique

Nous avons démontré le résultat suivant en cours.

Théorème 4. *Soit $\lambda > 0$. Si X_n est une variable aléatoire de loi géométrique de paramètre λ/n, alors X_n converge en loi vers une variable exponentielle de paramètre λ lorsque $n \to \infty$ (c'est-à-dire que pour tout $x \geq 0$ on a $\mathbb{P}(X_n \leq x) \to \mathbb{P}(Z \leq x)$ où Z est une variable aléatoire exponentielle de paramètre λ).*

Question 4. Illustrer ce théorème en complétant le Code 4.

Code 4 – Question4.py

```
from __future__ import division
import numpy as np
import numpy.random as npr
import scipy.stats as sps
import matplotlib.pyplot as plt

param=2 #parametre
n=1000
N=5000 #nombre de tirages effectue pour tracer une densite de X_n

X=npr.geometric(BLA)

plt.hist(BLA, normed=True, label="Densite empirique", bins=int(sqrt(N)))
x = np.linspace(BLA, BLA, 100)
f_x = param*np.exp(-BLA)
plt.plot(x, f_x, "r", label="Densite theorique")
plt.legend()
```

2.3 Illustration d'un exercice de DM

Dans le DM facultatif, on démontrait que si $(X_n, n \geq 2)$ est une suite de variables aléatoires exponentielles de paramètre 1, définies sur le même espace de probabilité, alors presque sûrement la plus grande valeur d'adhérence de la suite $(X_n/\ln(n), n \geq 2)$ vaut 1.

Question 5. Illustrer ce théorème en complétant le Code 5.

Code 5 – Question5.py

```
from __future__ import division
import numpy as np
import numpy.random as npr
import scipy.stats as sps
import matplotlib.pyplot as plt

n=100000

X=np.arange(2,n+2)
Y=BLA

plt.plot(X, Y)
plt.plot(X, BLA, "r", label="Droite d'equation y=1")

plt.legend()
```

Question 6. Étudiant numériquement la convergence en loi de la suite $X_n/\ln(n)$ lorsque $n \to \infty$.

3 Approximation de π par la méthode de Monte-Carlo

Considérons un couple (X, Y) de variables aléatoires indépendantes telles que chacune soit uniforme sur $[0, 1]$. Soit $((X_i, Y_i))_{i\geq 1}$ une suite de vecteurs aléatoires indépendants et de même loi que (X, Y).

On admet que $\mathbb{P}(X^2 + Y^2 \leq 1) = \frac{\pi}{4}$. Ceci est intuitif au moins : (X, Y) représente un point "uniforme" dans le carré $[0, 1]^2$, et le probabilité qu'il appartienne à un quart de disque centré en 0 est l'aire de ce quart de disque (c'est-à-dire $\pi/4$) divisé par l'aire totale du carré, qui vaut 1.

Posons $Z_i = 1$ si $X_i^2 + Y_i^2 \leq 1$ et $Z_i = 0$ sinon. Ainsi, $(Z_i)_{i\geq 1}$ sont des variables aléatoires indépendantes de Bernoulli de paramètre $\pi/4$. En posant

$$S_n = \frac{4}{n} \cdot \sum_{i=1}^{n} Z_i,$$

on en déduit d'après la loi forte des grands nombres que S_n converge presque sûrement vers π.

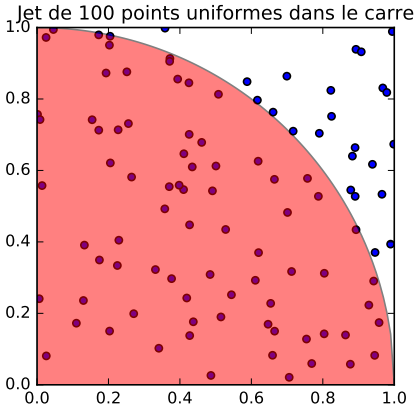

FIGURE 1 – Dans ce exemple, $S_{100} = \frac{4}{100} \cdot 77$ (il y a 77 points dans la région rouge).

On veut maintenant un intervalle de confiance sur la valeur de π. Pour cela, on remarque que

$$\mathbb{E}[S_n] = \pi, \qquad \mathsf{Var}(S_n) = \frac{16}{n^2} \sum_{i=1}^{n} \mathsf{Var}(X_i) = \frac{16}{n} \cdot \frac{\pi}{4} \cdot \left(1 - \frac{\pi}{4}\right) \leq \frac{4}{n}$$

car $x(1 - x) \leq 1/4$ sur $[0, 1]$.

D'après l'inégalité de Bienaymé-Tchebychev, on a donc

$$\mathbb{P}(|S_n - \pi| \geq \alpha) \leq \frac{4}{\alpha^2 n}. \tag{1}$$

Question 7. Simuler un intervalle de confiance de π d'amplitude au plus 10^{-2} à 99% en complétant le Code 7.

Code 7 – Question7.py

```
from __future__ import division
import numpy as np
import numpy.random as npr
import scipy.stats as sps
import matplotlib.pyplot as plt

n=BLA

X=npr.rand(n)
Y=npr.rand(n)

Sn=4/n*np.sum(BLA) #sum compte le nombre d'elements non nuls (ou vrais) dans un
    tableau

print "Intervalle de confiance pour Pi au niveau 0.99 : ["+str(BLA)+","+str(BLA)+"]"
    #On utilise str pour convertir un entier en chaine de caracteres et on utilise le
     + pour concatener des chaines de caracteres
print "En vrai, Pi vaut "+str(np.pi)
```

4 Réglement de comptes à OK Corall

Soit $n \geq 1$ un entier. Deux groupes de bandits, chacun constitué de n personnes, se retrouvent à OK Corral pour un règlement de comptes. Tant qu'il reste au moins un bandit vivant dans chaque groupe, à chaque seconde un bandit (choisi uniformément au hasard parmi ceux encore vivants, indépendamment de tout ce qui s'est passé avant) abat un bandit de l'autre groupe. On note $V(n)$ le nombre (aléatoire) de survivants à l'issue de la fusillade.

Une étude théorique permet de montrer que $V(n)/n^{3/4}$ converge en loi lorsque $n \to \infty$ vers une variable aléatoire réelle à densité dont une densité est

$$\sqrt{\frac{3}{\pi}} \cdot x \cdot e^{-\frac{3x^4}{16}} \mathbb{1}_{x \geq 0}.$$

Question 8. Illustrer cette convergence en loi par des simulations.

5 Bonus

Voici le code utilisé pour générer la Figure 1.

Code 8 – illustration.py

```
from __future__ import division
from math import *
from pylab import *
import numpy.random as npr
import scipy.stats as sps

n=100

X=npr.rand(n)
```

```
10 Y=npr.rand(n)
11
12 x=linspace(0,1,100)
13 y=sqrt(1-x**2)
14 x=insert(x,0,0) #on insere 0 en 1ere position
15 y=insert(y,0,0) #on insere 0 en 1ere position (pour que la region rouge passe par le
       point (0,0))
16
17 scatter(X,Y) #On dessine les points
18 fill(x,y, 'r',alpha=0.5) #On dessine la region rouge
19 axis([0,1,0,1]) #On restreint les axes
20 title('Jet de '+str(n)+' points uniformes dans le carre') #On utilise str pour
       convertir un entier en chaine de caracteres et on utilise le + pour concatener
       des chaines de caracteres
21 gca().set_aspect('equal', adjustable='box') #On force les axes a etre de meme
       longueur
22
23 savefig('fig.pdf') #On enregistre la figure dans un fichier pdf
```