École Polytechnique
Année 2
Mathématiques Appliquées
TP2
MAP441
Modal SNA

Florent Benaych-Georges
Martin Bompaire
Stefano De Marco
Gersende Fort
Emmanuel Gobet
Igor Kortchemski

Simulation de variables aléatoires, loi des grands nombres, théorème central limite

1 Simulation de variables aléatoires discrètes

1.1 Un exemple simple de loi discrète

On considère la loi P sur l'ensemble $\{1,2,3\}$ donnée par les probabilités $P(\{1\}) = 0.3$, $P(\{2\}) = 0.6$, $P(\{3\}) = 0.1$. Simulez un grand nombre de v.a. i.i.d. de loi P, en faire l'histogramme.
Indication : `numpy.random.choice([a1,...,an],p=[p1,...,pn],size=(n,p))`: tirages indép. dans `[a1,...,an]` de loi `[p1,...,pn]` et `matplotib.pyplot.hist` trace un histogramme (spécifier `normed=True`). Deux options pour les colonnes: `bins=` *nombre de colonnes* ou `bins=` *abscisses des séparations des colonnes.*

1.2 Les lois discrètes classiques

1. On considère $B(n,p)$ la loi binomiale de paramètres n, p. Ecrire une procédure dans laquelle on simule un grand nombre de v.a. de loi $B(n,p)$ (avec la fonction `binomial` de `numpy.random`), faire l'histogramme de l'échantillon obtenu et comparer avec la représentation en bâtons, sur le même graphique, de la loi $B(n,p)$ (que l'on obtient à l'aide de la fonction `binom.pmf` de `scipy.stats`).

2. Faire de même avec une loi de Poisson (on utilisera la fonction `poisson` de `numpy.random` et la fonction `poisson.pmf` de `scipy.stats`).

3. Un calcul simple montre que lorsque n tend vers l'infini, la loi $B(n, \lambda/n)$ tend vers la loi de Poisson de paramètre λ. En pratique, on assimile $B(n,p)$ à la loi de Poisson de paramètre np dès que $np^2 < 0.1$. Illustrer cette *proximité de lois* en affichant, sur le même graphique, leurs histogrammes en bâtons (sans faire aucune simulation).

2 Simulation de variables aléatoires continues (1)

1. Simuler un grand nombre de variables aléatoires gaussiennes standard, représenter l'histogramme associé et comparer à la densité gaussienne. Faire de même avec des variables aléatoires de loi *gamma*. On utilisera les fonctions `randn, linspace, hist, plot` et la fonction `norm.pdf` de la librairie `scipy.stats`

2. Un des programmes donnés dans le tutoriel joint illustre la propriété d'*absence de mémoire* de la loi exponentielle : si X suit une telle loi, alors pour tout $t > 0$, la loi de $X - t$ sachant $X > t$ est la loi de X. Charger ce programme, l'exécuter, et faire la même expérience en remplaçant la loi exponentielle par des (autres) lois *gamma*. A-t-on encore absence de mémoire ? On utilisera la fonction `gamma` de `numpy.random` et la fonction `gamma.pdf` de `scipy.stats`

3 Programmation matricielle

3.1 Motivation

En `python`, la programmation matricielle proposée dans `numpy` est plus rapide que celle reposant sur des boucles. Le programme suivant compare le calcul d'une valeur approchée du nombre $\gamma \approx \sum_{i=1}^{n} \frac{1}{i} - \ln(n)$ pour $n \gg 1$ en programmation matricielle et via une boucle. Le récupérer à l'adresse `http://www.cmapx.polytechnique.fr/~benaych/MODAL_MAP441/` et le faire tourner. Quel est le facteur de temps de calcul entre les deux méthodes ?

```
import numpy as np
from time import time

n = int(1e6)
# Methode 1. Boucle for
print "-" * 40
print "Methode 1. Boucle for"
print "-" * 40
t1 = time()
print "gamma=", np.sum([1./i for i in xrange(1,n)]) - np.log(n)
t2 = time()
temps1 = t2 - t1
print "Cela a pris ", temps1, " secondes"
# Methode 2. Numpy
print "-" * 40
print "Methode 2. Numpy"
t1 = time()
print "gamma=", np.sum(1. / np.arange(1, n)) - np.log(n)
t2 = time()
temps2 = t2 - t1
print "Cela a pris ", temps2, " secondes"
print "-" * 40
print "Facteur de gain ", temps1/temps2
```

3.2 Exercices

Chacune des deux questions suivantes est à traiter sans boucle. On essayera de ne pas passer trop de temps sur ces questions (notamment, les probabilités seront, exceptionnellement, estimées sans intervalle de confiance).

1. Soit, pour $N = 100$, $X_{(1)} < \cdots < X_{(N)}$ le réordonnement croissant d'une famille $X_1, \ldots, X_N$ de v.a. i.i.d. de loi gaussienne standard. Estimer la probabilité que $\cos(X_{(\lfloor 0.8N \rfloor)}) < 0.6$. On pourra simuler $M \times N$ v.a. gaussiennes, utiliser la fonction `numpy.sort(., axis=1)` puis utiliser le fait que si `V` est une matrice, pour tout `j`, `V[:, j]` désigne la `j`ème colonne de `V` (la première est la colonne numéro 0).
2. Soit $(X_i)_{i \geq 1}$ des v.a. i.i.d. de loi uniforme sur $[0, 1]$. Estimer n_0, la plus petite valeur de l'entier n telle que
$$\mathbb{P}[X_1 + \cdots + X_n < 5] \leq 0.2$$
L'inégalité de Markov nous permet d'affirmer que $n_0 \leq 20$. On pourra donc simuler $M \times 20$ v.a., utiliser les fonctions `numpy.cumsum(., axis=1)` et `numpy.sum(., axis=0)`, ainsi que la fonction `numpy.argwhere`.

4 Simulation de variables aléatoires continues (2) : inversion de la fonction de répartition

La *loi de Cauchy de paramètre a* est la loi de densité $f_a(x) = \frac{1}{\pi} \frac{a}{x^2+a^2}$ sur $\mathbb{R}$. Simuler un grand nombre de variables aléatoires de loi de Cauchy de paramètre a, représenter l'histogramme associé et le comparer à la densité. Que se passe-t-il lorsque a s'approche de 0 ?

Rappel : On rappelle que si U est une v.a. uniforme sur $]0, 1[$ et F la fonction de répartition d'une loi μ, alors $F^{-1}(U)$ est distribuée selon μ.

5 Loi des grands nombres

On rappelle que si (X_n) est une suite de variables aléatoires indepéndantes identiquement distribuées et ayant une espérance m, alors la suite de variables aléatoires
$$\bar{X}_n = \frac{X_1 + \cdots + X_n}{n}$$
converge presque sûrement vers m lorsque n tend vers l'infini. Simuler un grand nombre N de variables aléatoires de loi uniforme sur $[0, 1]$, tracer sur un même graphe la représentation de la suite $\bar{X}_n$ pour n variant de 1 à N et

la courbe d'équation $y = m$, et observer cette convergence. On pourra utiliser la fonction `axhline(y)` qui trace une droite horizontale à la hauteur `y`.

6 Théorème central limite

6.1 Le théorème

On rappelle que si (X_n) est une suite de variables aléatoires indepéndantes identiquement distribuées et de carré intégrable, d'espérance et écart-type communs notés respectivement m et s, alors en définissant

$$\bar{X}_n = \frac{X_1 + \cdots + X_n}{n},$$

la suite

$$\sqrt{n}(\bar{X}_n - m)/s$$

converge en loi vers la loi gaussienne standard lorsque n tend vers l'infini.

1. Lorsque la loi des X_i est la loi uniforme sur $[-\sqrt{3}, \sqrt{3}]$, que valent m et s ? Visualiser, pour $n \gg 1$, la proximité de la loi de $\sqrt{n}(\bar{X}_n - m)/s$ avec la loi gaussienne standard (à l'aide d'un histogramme, en simulant un grand nombre de réalisations indépendantes de $\sqrt{n}(\bar{X}_n - m)/s$).
2. Afin d'illustrer la différence de nature profonde entre la convergence presque sûre et la convergence en loi, tracer la représentation d'une réalisation de la suite $\sqrt{n}(\bar{X}_n - m)/s$, pour n variant de 1 à N. Cette suite semble-t-elle converger ?

6.2 Intervalles de confiance

Le théorème central limite explique que pour n assez grand, $\bar{X}_n - m$ est approximativement distribuée comme une variable aléatoire normale standard multipliée par $s/\sqrt{n}$. Ainsi, puisque pour G une variable aléatoire normale standard, on a

$$\mathbb{P}[-1.96 \leq G \leq 1.96] = 0.95,$$

on sait que pour n assez grand, on a approximativement

$$\mathbb{P}\Big[\bar{X}_n - \frac{1.96s}{\sqrt{n}} \leq m \leq \bar{X}_n + \frac{1.96s}{\sqrt{n}}\Big] \approx 0.95.$$

On dit alors que l'intervalle

$$\Big[\bar{X}_n - \frac{1.96s}{\sqrt{n}}, \bar{X}_n + \frac{1.96s}{\sqrt{n}}\Big]$$

est un **intervalle de confiance asymptotique au niveau 95%** pour m, c'est à dire qu'avec une probabilité asymptotiquement égale à 0.95, le paramètre m sera bien entre les bornes (aléatoires) de cet intervalle. Bien souvent, l'écart-type s n'est pas plus connu que l'espérance m. On peut alors le remplacer par son estimateur

$$\widehat{s} = \Big(\frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X}_n)^2\Big)^{1/2}.$$

On suppose encore que les X_i sont des v.a. i.i.d. de loi uniforme sur $[0, 1]$. Afin d'illustrer clairement la notion d'*intervalle de confiance à 95%*, fixer des valeurs élevées pour n et M, simuler M fois $\bar{X}_n$, et vérifier que dans 95% des cas, m se situe bien entre $\bar{X}_n - \frac{1.96\hat{s}}{\sqrt{n}}$ et $\bar{X}_n + \frac{1.96\hat{s}}{\sqrt{n}}$.

7 Calcul de probabilités d'événements rares : approximation gaussienne vs poissonienne

Soient $X_1, X_2, \cdots$ des v.a. i.i.d. de loi de Bernoulli de paramètre p et $S_n = X_1 + \cdots + X_n$. Par ce qui précède, pour de grandes valeurs de n, la loi de S_n peut être approximée par une loi de Poisson ou par une loi gaussienne, selon que p soit faible ou pas. Illustrer ces deux paradigmes en affichant, sur une même figure, les histogrammes en bâtons de la loi de S_n et de la loi de Poisson correspondante, ainsi que la densité de la loi gaussienne correspondante.

8 Application du TCL : intervalles de confiance pour la méthode de Monte-Carlo

1. On veut calculer l'intégrale, par rapport à la mesure de Lebesgue sur $[0,1]^{10}$, de la fonction $f(x) = \pi^2(x_1+\cdots+x_{10})^2\sqrt{x_1^2+\cdots+x_{10}^2}$. Soit I cette intégrale.

   (a) Peut-on espérer résoudre ce problème avec des sommes de Riemann, comme on le fait pour les intégrales de fonctions suffisamment régulières sur $\mathbb{R}$?

   (b) Donner, en utilisant les fonctions `numpy.mean(., axis=1)` (moyenne des lignes), `numpy.linalg.norm(., axis=1)` (norme euclidienne des lignes) et `numpy.std` (ecart-type), des valeurs approchées de l'espérance et la variance de $f(U)$, avec U variable aléatoire de loi uniforme sur $[0,1]^{10}$. En déduire un intervalle de confiance à 95% pour I. Donner une estimation du nombre de simulations nécessaires pour que l'intervalle de confiance ait une amplitude au plus égale à 2% de I.

2. Calculer la probabilité (dont on donnera une estimation avec un intervalle de confiance) qu'un vecteur aléatoire gaussien centré de taille 9 et de matrice de covariance $\mathrm{diag}([1,2,3,4,5,4,3,2,1]) + M$, avec

$$M = \left[\frac{1}{10(i+j)}\right]_{1\le i,j\le 9}$$

   se trouve dans l'hypercube $[-5,3]^9$. L'amplitude de l'intervalle pourra atteindre 4% de la probabilité estimée. On pourra utiliser la fonction `multivariate_normal` de `numpy.random` pour simuler des vecteurs gaussiens.

3. Soit d un entier positif (on le fixera par exemple à 10). Pour $x = (x_1, \cdots, x_d)$ un vecteur réel, on appelle *record* de x tout indice i dans $1, \cdots, d$ tel que $x_i = \max(x_1, \cdots, x_i)$. On considère ici le cas où x est un vecteur aléatoire à coordonnées indépendantes uniformes sur $[0,1]$.

   (a) Donner, en appliquant la même méthode que au dessus, le nombre moyen d'indices i dans $\{2, \cdots, d\}$ tels que i et $i-1$ sont tous les deux des records. L'amplitude de l'intervalle pourra atteindre 4% du nombre estimé. On pourra par exemple utiliser la fonction `numpy.maximum.accumulate(., axis=1)` qui permet de calculer le maximum cumulé des lignes d'un tableau.

   (b) Donner la loi du nombre d'indices i dans $\{2, \cdots, d\}$ tels que i et $i-1$ sont tous les deux des records (en affichant l'histogramme).

   (c) Essayer de faire la même chose avec un vecteur aléatoire x dont les coordonnées ne sont pas des v.a. i.i.d. (par exemple un vecteur gaussien d'espérance et de matrice de covariance non standard).

9 Simulation de variables aléatoires continues (3) : méthode de rejet

La *loi de Wigner* est la loi de support $[-2,2]$ et de densité $\frac{1}{2\pi}\sqrt{4-x^2}$. Simuler un grand nombre de variables aléatoires de loi de Wigner, représenter l'histogramme associé et le comparer à la densité.

Rappel : Pour simuler une v.a. de densité f de support $[a,b]$ et telle que $0 \le f \le M$, on utilise une suite $(U_n, V_n)_{n\ge1}$ de couples indépendants de variables aléatoires telles que pour tout n, U_n, V_n sont indépendantes et de lois uniformes sur respectivement $[a,b]$, $[0,M]$, on pose

$$\tau = \min\{n \ge 1\,;\, V_n \le f(U_n)\},$$

et la variable aléatoire U_τ suit alors la loi de densité f. C'est la méthode de simulation par rejet. Notons que plus M est petit, plus cette méthode est rapide, on a donc intérêt à choisir $M = \|f\|_\infty$.