Sup Galilée
MACS1

janvier 2022

Initiation au calcul numérique – TP Proba. 1
Principes de la simulation aléatoire

Les exercices marqués d'un astérisque sont à réaliser en **priorité**.

Avant de commencer ce TP, créer un répertoire `PROJETS_NUMERIQUES`, puis un sous-répertoire `TP_Proba_1` dans lequel vous placerez vos fichiers.

L'objectif de ce TP est d'introduire la **simulation aléatoire** à l'aide d'Octave. La simulation aléatoire consiste à reproduire sur ordinateur une expérience aléatoire. Du fait du fonctionnement déterministe des ordinateurs, une telle simulation est approximative, basée sur l'utilisation d'un algorithme de génération de **nombres pseudo-aléatoires**. Ce TP va aborder le principe de la génération de nombres pseudo-aléatoires (et ses limites), et son application principale en calcul numérique, qui est la **méthode de Monte-Carlo** d'estimation de probabilités et d'espérances.

1 Nombres pseudo-aléatoires

De nombreux domaines d'application utilisent des nombres aléatoires : sécurité informatique (génération automatique d'identifiants, de clés secrètes), méthodes d'optimisation dans des espaces de grande dimension (recuit simulé, algorithmes génétiques), simulations numériques de systèmes complexes (physique, ingénierie, finance, assurance, météo...), jeux vidéos (paysages aléatoires, intelligence artificielle,...),...

A priori, vouloir utiliser un ordinateur pour obtenir des nombres aléatoires apparaît paradoxal, sinon impossible : par définition, un nombre aléatoire n'est pas prévisible, tandis que l'ordinateur ne peut appliquer qu'une formule ou un algorithme prédéfinis. Il existe heureusement des algorithmes fournissant des suites de nombres ayant **de façon approximative** certaines propriétés d'une suite de variables aléatoires indépendantes et de même loi. On parle de nombres **pseudo-aléatoires**.

On s'attend à ce qu'une suite pseudo-aléatoire suive la loi des grands nombres, et soit « bien mélangeante ». Une définition plus précise est délicate, et dépendrait en fait de l'application voulue : parfois, on souhaite simplement une suite de valeurs « bien répartie » dans un ensemble, sans s'inquiéter de la dépendance entre termes successifs, tandis que d'autres contextes exigent des contraintes d'indépendance plus importantes (par exemple en cryptographie). De plus, si on a besoin d'une grande quantité de nombres aléatoires, alors il faut un meilleur générateur, qui peut être plus coûteux en temps de calcul.

Dans Octave, le générateur de nombres aléatoires prend la forme de la "fonction" `rand` (sans paramètre), qui est particulière car son résultat est un nombre réel dans $]0,1[$ qui change a priori à chaque fois que l'on appelle cette fonction. On fera l'hypothèse suivante :

Les différents appels à `rand` renvoient une réalisation "générique" $U_1(\omega), U_2(\omega), \ldots$
d'une suite de variables aléatoires $(U_1, U_2, \ldots)$ indépendantes et de loi uniforme dans $[0,1]$.

Autrement dit, il existe une suite $(U_n)_{n\geq 1}$ de variables aléatoires (c'est-à-dire de fonctions mesurables $U_n : \Omega \to \mathbb{R}$ définies sur un espace de probabilités $(\Omega, \mathcal{F}, P)$) indépendantes et suivant chacune la loi uniforme sur $[0,1]$, et il existe un $\omega \in \Omega$ tels que `rand` renvoie $U_1(\omega)$, puis $U_2(\omega)$, etc.

La valeur ω est la **graine** du générateur : si on connaît ω, la suite $(U_n(\omega))_{n\geq 1}$ est entièrement connue ! Cela est utile pour reproduire une simulation avec les mêmes "nombres aléatoires". Dans l'hypothèse ci-dessus, on a précisé "générique", pour signifier que ω réalise ("en pratique") n'importe quel événement presque sûr. On pourra donc utiliser `rand` pour observer des propriétés vraies presque sûrement, notamment la loi forte des grands nombres.

Dans Octave, une commande pour fixer la graine égale à n (entier positif) est `rand("seed",n)`. On peut aussi vouloir laisser Octave choisir la graine de façon plus "aléatoire" (en fait la graine est choisie en utilisant l'heure, en millisecondes) avec `rand("seed","reset")`, auquel cas le résultat du premier appel à `rand` est à peu près imprévisible. C'est ce qui est fait au démarrage d'Octave.

Remarquons qu'en utilisant des nombres pseudo-aléatoires dans Octave, on combine donc deux approximations : l'approximation numérique (expliquée dans le TP Analyse Numérique 1) liée à la précision limitée, c'est-à-dire que `rand` ne donne pas n'importe quel nombre réel dans $]0,1[$, et l'approximation aléatoire liée à l'imitation imparfaite du hasard.

Exercice 1* : générateur et graine, utilisation de `rand`

1. Dans la fenêtre de commandes de Octave, exécuter `rand("seed",1)`, puis exécuter `rand` plusieurs fois et observer les résultats. Exécuter à nouveau `rand("seed",1)` et faire à nouveau appel à `rand` : que remarque-t-on ?

2. Recommencer en remplaçant `1` par `2`. Comparer les résultats.

3. Recommencer en remplaçant `1` par `"reset"`, et réessayer plusieurs fois : que remarque-t-on ? Est-ce que utiliser ensemble `rand("seed","reset")` et `rand` à chaque fois que l'on veut un nombre aléatoire (par exemple dans une boucle) est une bonne méthode pour obtenir une suite de nombres "vraiment" aléatoires ? (Réponse : non. Pourquoi ?)

Exercice 2* : simuler des variables aléatoires à loi discrète

Bien que Octave ne fournisse que des nombres qui suivent la loi uniforme dans $[0,1]$, on verra que cela permet d'obtenir des variables aléatoires réelles, ou dans $\mathbb{R}^d$, suivant n'importe quelle loi. On dira que l'on **simule** une loi de probabilité lorsque l'on fait produire à Octave un nombre aléatoire suivant cette loi.

On s'intéresse pour le moment à la simulation de quelques exemples de lois discrètes.

1. Écrire une fonction `x=signe_aleatoire(p)` qui renvoie un nombre aléatoire qui vaut 1 avec probabilité `p`, et -1 avec probabilité $1-$`p`.
Indication : si U suit la loi uniforme sur $[0,1]$, que vaut $\mathbb{P}(U \leq p)$? En déduire un programme avec une structure `if`.

2. Écrire une fonction `vx=signes_aleatoires(n,p)` qui renvoie un vecteur de n nombres aléatoires donnés par la fonction précédente.

3. Créer et exécuter un script `ex2.m` qui contient `N=1000;plot(1:N,cumsum(signes_aleatoires(N,0.5))`. Qu'a-t-on représenté ?

4. Une particule se déplace au hasard en sautant en diagonale sur une grille : elle part de $(0,0)$ et si sa position après n sauts est notée (X_n, Y_n), alors

$$X_{n+1} = X_n + U_{n+1} \text{ et } Y_{n+1} = Y_n + V_{n+1}$$

où $U_1, U_2, \ldots, V_1, V_2, \ldots$ sont indépendantes et de même loi $\frac{1}{2}\delta_1 + \frac{1}{2}\delta_{-1}$. Dans `ex2.m`, avec l'instruction `plot`, représenter une réalisation du chemin parcouru par cette particule jusqu'au temps 500. *On pourra s'inspirer de la question précédente.*
NB. La commande `axis equal;` permet d'avoir un repère orthonormé (même échelle sur les deux axes).

5. Écrire une fonction `x=uniforme(n)` qui renvoie un nombre aléatoire choisi de manière uniforme dans l'ensemble $\{1, 2, 3, \ldots, n\}$.
Indication : si U suit la loi uniforme sur $[0,1]$, montrer que $\lfloor nU \rfloor + 1$ suit la loi uniforme sur $\{1, 2, \ldots, n\}$, où $\lfloor x \rfloor$ est la partie entière (inférieure) d'un réel x (donnée par `floor` dans Octave). On n'utilisera pas la fonction `randi`, seulement `rand`.

6. Écrire une fonction `x=discrete012(p,q)` qui, étant donnés `p` et `q` tels que `p`≥ 0, `q`≥ 0 et `p+q`≤ 1 renvoie un nombre aléatoire X tel que

$$P(X=0) = p, \qquad P(X=1) = q, \qquad \text{et} \qquad P(X=2) = 1-p-q.$$

Indication : Utiliser `rand` une seule fois dans la fonction. On pourra commencer par penser au cas $p = q = \frac{1}{3}$.

Exercice 3* : simuler des variables aléatoires à loi uniforme continue

On s'intéresse pour le moment à la simulation de l'exemple le plus simple de lois continues : la loi uniforme sur $[a,b]$.

1. Écrire une fonction `x=uniforme_continue(a,b)` qui renvoie un nombre aléatoire dont la loi est uniforme sur $[a,b]$ (avec a, b réels et $a < b$).
Indication : On utilisera une transformation affine d'une variable de loi uniforme sur $[0,1]$.

2. (Dans `ex3.m`) Dans la fenêtre graphique n° 3 (utiliser `figure(3);`), représenter graphiquement une simulation de l'expérience aléatoire suivante : on choisit 1000 points aléatoirement, uniformément sur le cercle unité, que l'on les relie entre eux dans l'ordre de leurs tirages.

3. (Dans `ex3.m`) Dans la fenêtre graphique n° 4, représenter graphiquement une simulation de l'expérience aléatoire suivante : on choisit 1000 points aléatoirement, uniformément dans le rectangle $[0,2] \times [0,1]$, et on les représente par des points non reliés.

Exercice 4 : générateurs de nombres aléatoires par congruence linéaire

(Ne pas faire cet exercice avant d'avoir fait tous les exercices avec un $*$)

Pour bien comprendre comment `rand` fonctionne, écrivons notre propre générateur de nombres aléatoires, bien plus simple que celui de Octave. Les méthodes du type suivant ont été utilisées jusque dans les années 90. On définit la suite $(X_n)_{n\geq 0}$ par récurrence, par la donnée de $X_0 = \omega \in \{1, \ldots, 2^{31}-1\}$ (c'est la graine) puis

$$\text{pour tout } n \geq 0, \qquad X_{n+1} = 16807\ X_n \pmod{2^{31}-1}.$$

Alors les valeurs $(X_n)_{n\geq 1}$ se comportent approximativement comme des entiers aléatoires uniformément choisis entre 0 et $2^{31}-1 = 2147483647$, et donc $U_n = 2^{-31}X_n$ suit approximativement la loi uniforme sur $[0,1]$.

Par des propriétés arithmétiques, on peut montrer que la suite $X_0, X_1, \ldots, X_{N-1}$ parcourt une fois chacune des valeurs de $1, 2, \ldots, N-1$, où $N = 2^{31}-1$ (c'est un nombre premier), et $X_N = X_0$.

1. On souhaite écrire une fonction `u=alea()` qui renvoie un nombre aléatoire uniforme dans $[0,1]$ à l'aide de cette méthode, et une fonction `graine(n)` qui donne à la graine la valeur `n`. Reproduire les codes suivants et les tester quelques fois (les résultats ont-ils bien l'air aléatoires ?), et chercher comment il se fait que `alea()` renvoie des valeurs différentes alors qu'il n'y a pas de paramètre.

```
function u=alea()
    global X;
    X=mod(X*16807, 2147483647);
    u=X/2147483648;
end
```

```
function graine(n)
    global X;
    X=n
end
```

2. Si l'on a besoin de plus que $2^{31} (\simeq 2 \cdot 10^9)$ nombres aléatoires, peut-on utiliser ce générateur ?

3. Écrire une fonction `vecteur_alea(n)` qui renvoie un vecteur $[x_1, \ldots, x_n]$ de nombres réels tirés par la fonction `alea`. Pour $n = 10000$, représenter un histogramme des résultats, à l'aide de la fonction Octave `hist(x,k)` (où `x` est le vecteur des données, et `k` est le nombre de classes de l'histogramme) ; que met-on ainsi en évidence ? Comparer les histogrammes obtenus pour `k` égal à 10, 50, 100 et 500.

4. Écrire une fonction `points2D(n)` qui représente, sur un graphique, n points (x_1, y_1), (x_2, y_2),..., (x_n, y_n), où $x_1, y_1, x_2, y_2, \ldots$ sont aléatoires, indépendants et uniformes dans $[0,1]$, obtenus grâce à la fonction `alea`. Apprécier visuellement la répartition des points.

5. Écrire une fonction `points3D(n)` qui représente, sur un graphique, n points (x_1, y_1, z_1), (x_2, y_2, z_2),..., (x_n, y_n, z_n), où $x_1, y_1, z_1, x_2, y_2, z_2, \ldots$ sont aléatoires, indépendants et uniformes dans $[0,1]$, obtenus dans l'ordre $x_1, y_1, z_1, x_2, y_2, z_2, \ldots$ grâce à la fonction `alea`. *On pourra utiliser `plot3(x,y,z,'.')` où `x`, `y` et `z` sont des vecteurs de même taille.* Apprécier visuellement la répartition des points pour $n = 1000$.

6. Entre les années 60 et les années 80, le générateur de nombres aléatoires de certaines bibliothèques FORTRAN très répandues, nommé RANDU, utilisait la récurrence $X_{n+1} = 65539 X_n \pmod{2^{31}}$, puis $U_n = 2^{-31}X_n$, en partant d'un $X_0 \in \{1, \ldots, 2^{31}-1\}$. Il est aujourd'hui considéré comme exemple caractéristique de *mauvais* générateur. Modifier le code précédent (renommer en `alea_RANDU` et `graine_RANDU`), et exécuter `points3D_RANDU(1000)`. Est-ce que les points semblent bien répartis dans le cube ? *Non, cela illustre des corrélations entre tirages successifs.*

2 Méthode de Monte-Carlo

Le résultat fondamental suivant a été démontré en cours de probabilités :

Theorem 2.1 (Loi des grands nombres). *Si $X_1, X_2, \ldots$ sont des variables aléatoires réelles indépendantes, qui suivent toutes la même loi qu'une variable aléatoire X intégrable, alors*

$$\text{presque sûrement,} \qquad \frac{X_1 + \cdots + X_N}{N} \underset{N\to\infty}{\longrightarrow} E[X].$$

En particulier, si $A_1, A_2, \ldots$ sont des événements indépendants, qui ont tous la même probabilité $P(A)$, alors

$$\text{presque sûrement,} \qquad \frac{\mathbf{1}_{A_1} + \cdots + \mathbf{1}_{A_N}}{N} = \frac{1}{N}\mathrm{Card}(\{1 \leq i \leq N \,|\, A_i \text{ est réalisé}\}) \underset{N\to\infty}{\longrightarrow} P(A).$$

En pratique, si on dispose d'une fonction qui fournit une réalisation d'une variable aléatoire X, alors il suffit de faire appel à cette fonction plusieurs fois pour obtenir des réalisations de variables aléatoires $X_1, X_2, \ldots$ indépendantes et de même loi que X. En effet, chaque appel utilise la fonction `rand`, dont les résultats sont indépendants et de même loi.

Ceci donne lieu à la **méthode de Monte-Carlo** (en référence au casino de Monaco) : pour obtenir une estimation de l'espérance $E[X]$, on simule un grand nombre de fois la variable X de façon indépendante, et on calcule la moyenne des résultats. De même, pour obtenir une estimation d'une probabilité $P(A)$, on simule un grand nombre de fois l'expérience (autrement dit, les variables $X, Y, \ldots$ dont dépend A) de façon indépendante, et on calcule la proportion des expériences dans lesquelles l'événement A a été réalisé. Autrement dit, on a les schémas suivants :

◇ Estimation d'une espérance $E[X]$:

```
N=10000;
SX=0;
for i=1:N
        (code pour simuler X)
        SX=SX+X;
end
Xmoy=SX/N;
```

◇ Estimation d'une probabilité $P(A)$:

```
N=10000;
NA=0;
for i=1:N
        (simulation)
        if(A est réalisé)
                NA=NA+1;
        end
end
P_est=NA/N;
```

Parfois, il pourra être utile de conserver les résultats de toutes les simulations dans un vecteur, afin de calculer ensuite des statistiques autres que la moyenne. On définira alors par exemple `vX=zeros(1,N);` puis `vX(i)=X;` à chaque itération.

Lors d'un prochain TP, on s'intéressera au choix de `N` et aux marges d'erreur. Pour aujourd'hui, on se contentera de prendre `N=` 10000 ou plus (pour les estimations de probabilités, cela assure une précision de 1%, dans 95% des tirages). On mettra **toujours** cette valeur dans une variable (plutôt que directement dans l'instruction `for`), pour pouvoir la modifier facilement.

Exercice 5* : Illustration graphique de la loi des grands nombres

Répondre aux questions suivantes dans un script `LGN.m`. On fera afficher les graphes de chaque question dans des fenêtres différentes.

1. Simuler un vecteur de $N = 10000$ variables aléatoires $X_1, X_2, \dots, X_N$ indépendantes et de loi uniforme sur $[0, 1]$. Représenter graphiquement la suite $(\frac{X_1+\cdots+X_n}{n})_{1\leq n\leq N}$ *(On pourra utiliser la fonction `cumsum`)*. Expliquer le résultat. Représenter sur le même graphe une droite horizontale qui vous semble pertinente.

2. On définit $Y_n = (X_n)^{-1}$ pour $n = 1, \dots, N$ (où $X_1, \dots, X_N$ viennent de la question précédente). Représenter graphiquement la suite $(\frac{Y_1+\cdots+Y_n}{n})_{1\leq n\leq N}$. Expliquer le résultat. *(Peut-on appliquer la LGN ? Que vaut l'espérance de Y ?)* Il pourra être intéressant de lancer la simulation plusieurs fois.

3. On définit $Z_n = (X_n)^{-0.9}$ pour $n = 1, \dots, N$ (où $X_1, \dots, X_N$ viennent de la question précédente). Représenter graphiquement la suite $(\frac{Z_1+\cdots+Z_n}{n})_{1\leq n\leq N}$. Comment expliquer ce résultat ? *(Même quand la LGN s'applique, la convergence peut être plus ou moins rapide...)*

Exercice 6* : Estimations par méthode de Monte-Carlo

Répondre aux questions suivantes dans un script `estimation.m` qui affichera les résultats (utiliser `disp("Texte")` et `disp(x)` pour indiquer les questions et réponses).

1. Estimer $E\left[\frac{4}{1+X^2}\right]$, où X suit la loi uniforme sur $[0, 1]$.

2. Estimer $E[\sqrt{X^2+Y^2}]$, où X, Y suivent la loi uniforme sur $[0, 1]$ et sont indépendantes.

3. Estimer $E[\sqrt{X^2+Y^2}]$, où X, Y suivent la loi uniforme sur $[0, 1]$ et sont indépendantes.

4. Estimer $P(X + Y + Z \geq 5)$, où X, Y, Z sont indépendantes et de loi uniforme sur $\{1, 2, 3, 4\}$.

5. Estimer $E[N]$, où $N = \inf\{k \geq 1 \,|\, X_1 + \cdots + X_k > 20\}$, où $X_1, X_2, \dots$ sont des variables aléatoires de loi uniforme sur $\{1, 2, 3\}$. *On pourra d'abord écrire une fonction `simule_N` qui simule N.*

Exercice 7 : Collectionneur de vignettes

Dans chaque tablette de chocolat se trouve une vignette à collectionner. Il y a $V = 20$ vignettes différentes, on suppose que les vignettes présentes dans les tablettes sont indépendantes et uniformément choisies parmi les V vignettes existantes.

1. Écrire une fonction `collection(V)` qui renvoie une simulation du nombre de tablettes à ouvrir pour avoir toute la collection. *On pourra utiliser un vecteur `vignettes` de longueur V qui mémorise les vignettes déjà obtenues.*

2. Estimer le nombre moyen de tablettes à ouvrir pour avoir toute la collection.