

**Simulation de variables aléatoires.**  
Fabienne CASTELL

**Quelques références.**

N. BOULEAU. *Probabilités de l'ingénieur, variables aléatoires et simulation*. Hermann, Paris, 1986.

L. DEVROYE. *Non-uniform random variate generation*. Springer, New York, 1986.

S. ROSS *Simulation*. 2nd edition. Academic Press, 1997.

## 1 Générateurs pseudo-aléatoires.

### 1.1 Introduction.

La plupart des langages fournissent un générateur de “nombres aléatoires”, i.e. une fonction (`rand`, `random` ...), qui lorsqu'on l'appelle, rend “un nombre au hasard entre 0 et 1”. Que recouvre cette fonction ? Dans quelle mesure rend-elle un nombre aléatoire ? Qu'entend-on par “simuler des nombres aléatoires” ? A quoi ressemble une suite de nombres aléatoires ? Autant de questions sur lesquelles la littérature est vaste, et auxquelles nous n'allons ici apporter des réponses que très partielles.

Par exemple, supposons que nous voulions simuler un jeu de pile ou face, i.e. une suite de 0 et de 1 de longueur  $N$ , 0 apparaissant avec proba  $1/2$ . Les suites de longueur  $N$

1. 0 0 0 0 ..... 0,
2. 0 1 0 1 .....,
3. 0 1 1 0 1 1 1 0 0 1 ,

ont toutes la même probabilité d'occurrence  $\frac{1}{2^N}$ . Pourtant si après  $N$  appels d'une fonction `Rand` sensée rendre 0 ou 1 avec proba  $1/2$ , on obtenait l'une des deux premières suites, on aurait du mal à croire au caractère aléatoire de cette fonction `Rand`. Intuitivement, est aléatoire ce qui est imprévisible, i.e. ce qui ne résulte pas d'un algorithme. Il y a donc une contradiction à vouloir essayer de simuler des nombres aléatoires sur un ordinateur. On sait pourtant qu'il existe des suites qui bien qu'issues d'algorithmes déterministes, ont un comportement “chaotique”. Les générateurs de nombres aléatoires sont des algorithmes qui génèrent de telles suites, dites **pseudo-aléatoires**. Se

pose alors la question de savoir ce qu'est un comportement "chaotique". En pratique, on demande à ces suites de nombres pseudo-aléatoires de reproduire **certains** des comportements théoriques des suites de nombres aléatoires. Par exemple, si  $(X_i)_{i \geq 1}$  est une suite de variables aléatoires i.i.d. de loi de Bernoulli de paramètre  $1/2$  ( $\mathcal{B}(1/2)$ ) (i.e. valant 1 ou 0 avec proba  $1/2$ ), la loi forte des grands nombres assure que presque toute réalisation de cette suite vérifie

$$\frac{1}{N} \sum_{i=1}^N X_i(\omega) \xrightarrow{N \rightarrow \infty} 1/2 .$$

On peut donc demander à notre générateur **Rand** de construire des suites vérifiant cette propriété. De façon plus générale, pour tout entier  $k$  et tout  $k$ -uplé  $(\varepsilon_1, \dots, \varepsilon_k) \in \{0, 1\}^k$ , la loi forte des grands nombres dit que p.s.,

$$\frac{1}{N} \sum_{i=1}^N \mathbb{I}_{(\varepsilon_1, \dots, \varepsilon_k)}(X_{(i-1)k+1}(\omega), \dots, X_{ik}(\omega)) \xrightarrow{N \rightarrow \infty} \frac{1}{2^k} .$$

**Définition 1.1** Une suite  $x = (x_n)$  de  $\{0, 1\}^{\mathbb{N}}$  est dite **k-uniforme**, ssi pour tout  $(\varepsilon_1, \dots, \varepsilon_k) \in \{0, 1\}^k$ ,

$$\frac{1}{N} \sum_{i=1}^N \mathbb{I}_{(\varepsilon_1, \dots, \varepsilon_k)}(x_{(i-1)k+1}, \dots, x_{ik}) \xrightarrow{N \rightarrow \infty} \frac{1}{2^k} .$$

On attend donc en pratique de notre générateur **Rand** qu'il produise des suites  $k$ -uniformes pour tout  $k$  (ou  $\infty$ -uniformes). Cela n'est évidemment qu'une conséquence du caractère aléatoire. Par exemple, on peut montrer que la suite n° 3 qui code les entiers en base 2 (suite de Champernowne), est  $\infty$ -uniforme. Elle n'a pourtant rien d'aléatoire.

Par ailleurs, l'infini n'existant pas pour un ordinateur, on ne peut pas en pratique vérifier qu'un générateur donné produit des suites  $\infty$ -uniformes. La seule conduite pratique que l'on peut raisonnablement tenir, est de vérifier que rien d'in vraisemblable ne se produit. La démarche est ici très empirique. On considère que le générateur dont on dispose est "bon", jusqu' à preuve du contraire; i.e. **le générateur Rand est jugé convenable, s'il passe avec succès une série de tests statistiques**, visant à tester la  $k$ -uniformité. Nous effectuerons en TPs quelques tests sur le générateur pseudo-aléatoire de Matlab.

## 1.2 A propos des générateurs de nombres pseudo-aléatoires.

Les générateurs de nombres pseudo-aléatoires les plus simples, sont les générateurs par **congruence**. Ils produisent des suites récurrentes du type

$$x_{n+1} = f(x_n) ,$$

où  $f$  est de la forme

$$f(x) = (Ax + B) \text{ modulo } M .$$

Ils rendent donc des nombres sur  $\{0, 1, \dots, M - 1\}$ . Si  $M$  est suffisamment grand, on les considère après renormalisation par  $M$ , comme des générateurs de nombres uniformes sur  $[0, 1]$ . Les valeurs  $A = 16807$ ,  $B = 0$ , et  $M = 2147483647$  ont été utilisées pendant de nombreuses années pour leur bonne conduite vis-à-vis de l' $\infty$ -uniformité.

Des générateurs plus performants les ont supplantés aujourd'hui. Toutefois, le principe général reste le même. La valeur  $x_{n+1}$  de la suite est obtenue comme une fonction de la ou des valeurs précédentes. Tout générateur demande donc une valeur initiale  $x_0$ . Pour une même valeur de  $x_0$ , on obtient la même suite de nombres pseudo-aléatoires. Dans Matlab, le générateur de nombres pseudo-aléatoires est appelé par la commande `rand`. Sa valeur initiale (`seed` pour Matlab 4, `state` pour Matlab 5) est remise à la même valeur au début de chaque session, si bien que chaque fois qu'on réinitialise Matlab, on obtient la même suite de nombres aléatoires. La commande

```
rand('state')
```

permet d'obtenir l'état actuel du générateur. C'est le vecteur (de dimension 35 dans Matlab 5) qui sera utilisé lors du prochain appel de `rand`. La commande

```
rand('state',0)
```

remet le générateur dans son état initial. La commande

```
rand('state',j)
```

met le générateur dans son  $j$ -ième état. Attention! Le  $j$ -ième état du générateur ne correspond pas à l'état du générateur après le  $j$ -ième appel de la fonction `rand`!

## 2 Génération de variables aléatoires à partir d'uniformes.

Jusqu'à présent, nous n'avons abordé que la génération de nombres uniformément répartis sur  $[0, 1]$  (en fait uniformément répartis sur  $\{0, \dots, M - 1\}$ ). En théorie, cela suffit pour simuler n'importe quel type de lois (cf lemme 2.1). Nous allons décrire ici deux méthodes générales qui permettent de simuler certaines lois à partir de lois uniformes sur  $[0, 1]$ . Dans tout ce paragraphe, on supposera donc qu'on dispose d'un générateur `Rand` de nombres aléatoires uniformes sur  $[0, 1]$ .

## 2.1 Inversion de la fonction de répartition.

**Principe de la méthode.** Soit  $X$  une variable à valeurs dans  $\mathbb{R}$  de fonction de répartition

$$F(x) \triangleq P[X \leq x] .$$

$F$  est clairement une fonction croissante, continue à droite, qui tend vers 0 en  $-\infty$ , et vers 1 en  $+\infty$ . On définit l'inverse généralisée de  $F$  (cf figure 2.1) par

$$\forall u \in [0, 1]; \quad F^{-1}(u) \triangleq \inf\{x, F(x) \geq u\} .$$

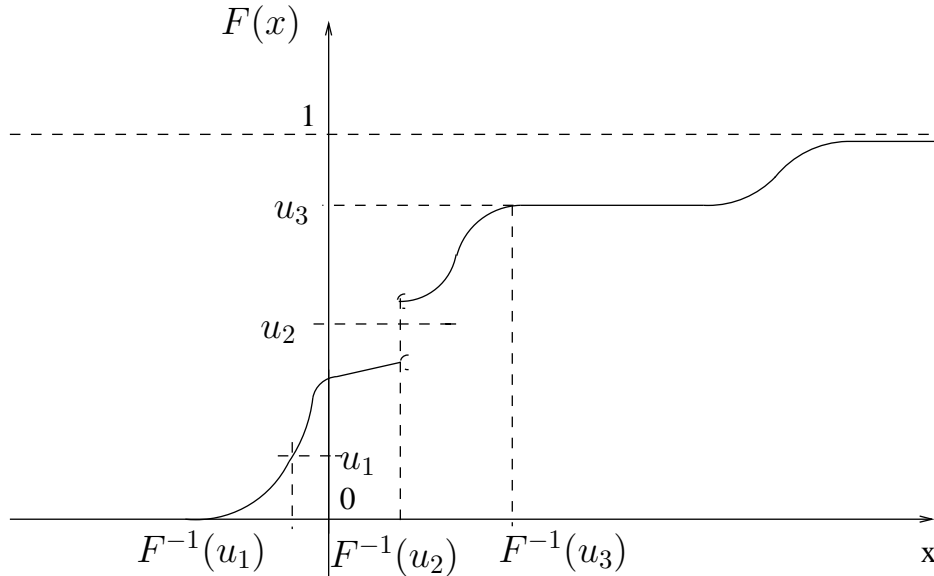


FIG. 1 – Représentation de l'inverse généralisée de  $F$ .

On a alors

**Lemme 2.1** Soit  $F$  une fonction de répartition sur  $\mathbb{R}$ , et  $U$  une variable de loi  $\mathcal{U}([0, 1])$ . La variable  $F^{-1}(U)$  admet  $F$  pour fonction de répartition.

Ce lemme permet en théorie de générer n'importe quel type de lois sur  $\mathbb{R}$ . En pratique, la méthode d'inversion demande la connaissance de la fonction  $F^{-1}$ . Par exemple, elle permet de générer très facilement des lois exponentielles. En effet, si  $X \sim \mathcal{E}(\lambda)$ ,

$$F_X(t) = \begin{cases} 0 & \text{pour } t \leq 0 \\ 1 - e^{-\lambda t} & \text{pour } t > 0 \end{cases} , \quad \text{et } F_X^{-1}(u) = -\frac{\log(1-u)}{\lambda} .$$

Par conséquent, si  $U \sim \mathcal{U}([0, 1])$ ,  $-\frac{\log(1-U)}{\lambda} \sim \mathcal{E}(\lambda)$ . Comme  $U$  et  $1-U$  ont même loi, on en déduit que si  $U \sim \mathcal{U}([0, 1])$ ,  $-\frac{\log(U)}{\lambda} \sim \mathcal{E}(\lambda)$ .

Si on ne connaît pas  $F^{-1}$  de façon explicite (ce qui est le cas pour la gaussienne par exemple), on peut essayer de l'approcher. C'est ce qu'on fait

lorsqu'on utilise les fameuses "tables" de la loi gaussienne. On commence par stocker une table des valeurs approchées de  $F$  en  $N$  points  $(x_i)_{1 \leq i \leq N}$ . On approche ensuite  $F$  par interpolation linéaire sur ces valeurs approchées. On approche alors  $F^{-1}$  en inversant cette interpolation linéaire; i.e, on calcule  $F^{-1}(u)$  par l'algorithme

$$\begin{aligned} & \text{Chercher } i \text{ tel que } F(x_i) \leq u < F(x_{i+1}). \\ & F^{-1}(u) \leftarrow x_i + \frac{u - F(x_i)}{F(x_{i+1}) - F(x_i)}(x_{i+1} - x_i) \end{aligned}$$

En plus de n'être qu'une méthode approchée, cette méthode demande

- de stocker les tables de la loi en question;
- de faire un algorithme de recherche.

**Application aux lois discrètes.** La méthode d'inversion peut bien sûr être utilisée pour simuler des lois discrètes. Supposons que l'on veuille simuler une v.a  $X$  à valeurs dans l'ensemble fini ou dénombrable  $\{x_i, 1 \leq i \leq N\}$  ( $N$  éventuellement infini,  $x_i < x_{i+1}$ ), et telle que

$$\forall i, \quad P[X = x_i] = p_i .$$

Soit  $F$  la fonction de répartition de  $X$

$$F(x) = p_1 + \dots + p_i, \quad \text{si } x_i \leq x < x_{i+1}, \quad 0 \leq i,$$

où  $x_0 = -\infty$  par convention. En notant  $F_i \triangleq p_1 + \dots + p_i$ , l'algorithme par inversion de la fonction de répartition est le suivant.

$X \leftarrow \text{Rand}$

Chercher  $i$  tel que  $F_i \leq X \leq F_{i+1}$ .  $X \leftarrow x_i$

Si le nombre d'éventualités  $N$  est petit, on peut choisir de stocker les valeurs de  $F$  pour ne pas les recalculer à chaque fois. Si  $N$  est grand, voire infini, ce stockage peut prendre trop de place mémoire, et il vaut peut-être mieux les calculer à chaque fois. Si  $F$  est telle que  $\exists k \ll N$  tel que  $F_k \simeq 1$ , on peut choisir de stocker les  $k$  premières valeurs de  $F_i$ , et de calculer les autres si besoin est; i.e si  $\text{Rand} > F_k$ , ce qui arrivera avec proba  $1 - F_k \simeq 0$ .

## 2.2 Méthode du rejet.

**Principe de la méthode.** La méthode du rejet repose sur le résultat suivant :

**Lemme 2.2** *Soit  $\mu$  une mesure positive sur  $\mathbb{R}$ , et soient  $f$  et  $g$  deux densités de probabilité par rapport à  $\mu$ , telles qu'il existe une constante  $C$  vérifiant pour tout  $x \in \mathbb{R}$ ,*

$$f(x) \leq Cg(x) .$$

*Soit  $(X_i)_{i \geq 1}$  une suite de v.a.i.i.d de densité  $g$ , et  $(U_i)_{i \geq 1}$  une suite de v.a.i.i.d de loi  $\mathcal{U}([0, 1])$  indépendante de la suite  $(X_i)_{i \geq 1}$ . Soit*

$$T \triangleq \inf\{i \geq 1, \text{ tel que } CU_i g(X_i) < f(X_i)\} .$$

*La variable  $X_T$  a pour densité  $f$ , par rapport à  $\mu$ .*

Ce lemme permet de simuler une densité  $f$  à partir d'une autre densité  $g$  facile à simuler. Supposons qu'on dispose d'une fonction **Randg** qui lorsqu'on l'appelle, "rend une réalisation d'une variable de densité  $g$ ". L'algorithme

```

X ← Randg
U ← Rand
While ( f(X) ≤ CUg(X) ),
    X ← Randg
    U ← Rand
EndWhile

```

rend une valeur  $X$  de loi de densité  $f$ .

Le nombre de passages dans la boucle est égal à l'instant où apparaît Face dans une suite de Pile ou Face, où Face sort avec la probabilité

$$p = P[f(X) > CUg(X)] = \int_{[0,1] \times \mathbb{R}} \mathbb{1}_{f(x) > Cug(x)} g(x) du \mu(dx) = \frac{1}{C}.$$

(on remarquera que  $C$  est nécessairement supérieur ou égal à 1, puisque  $1 = \int f(x) d\mu(x) \leq C \int g(x) d\mu(x) = C$ ). Ce nombre de passages est donc de loi géométrique de paramètre  $1/C$ , et en moyenne on passera  $C$  fois dans la boucle. On a donc intérêt à choisir  $g$  de façon à rendre  $C = \text{Max} \left( \frac{f(x)}{g(x)} \right)$  aussi petit que possible.

Par rapport à la méthode d'inversion, la méthode de rejet ne demande ni la connaissance de  $F$ , ni celle de  $F^{-1}$ . En particulier, elle s'étend telle quelle à des lois sur  $\mathbb{R}^d$ .

La méthode de rejet est souvent utilisée quand on veut simuler des lois à support compact,  $[0, 1]$  par exemple. On prendra alors  $\mu(dx) = \mathbb{1}_{[0,1]}(x) dx$ ,  $g(x) = 1$ ,  $f$  la densité à simuler, auquel cas  $C = \text{Max}_{[0,1]} f(x)$ . Dans ce cadre,

la méthode de rejet consiste à choisir un point  $(x, y)$  de façon uniforme sur  $[0, 1] \times [0, C]$ , et à regarder s'il tombe dans l'aire sous la courbe (cf figure). Si oui, le point est conservé, et la sortie de l'algorithme est l'abscisse de ce point. Si non, le point est rejeté, et on refait l'expérience.

**Application aux lois discrètes.** Si on prend pour  $\mu$  la mesure de comptage sur un ensemble discret, la méthode de rejet permet de simuler des variables discrètes. Par rapport à la méthode par inversion, elle a l'avantage de ne pas demander le calcul ou le stockage de  $F$ . Elle sera donc préférée à la méthode de rejet pour des lois discrètes ayant un grand nombre d'éventualités, et telles que toutes les éventualités ont à peu près la même probabilité (ce qui exclut la possibilité de trouver  $k$  petit tel que  $F_k \simeq 1$ ).

### 2.3 Simulation de lois gaussiennes.

Pour simuler des lois gaussiennes  $\mathcal{N}(0, 1)$ , on utilise le résultat suivant.

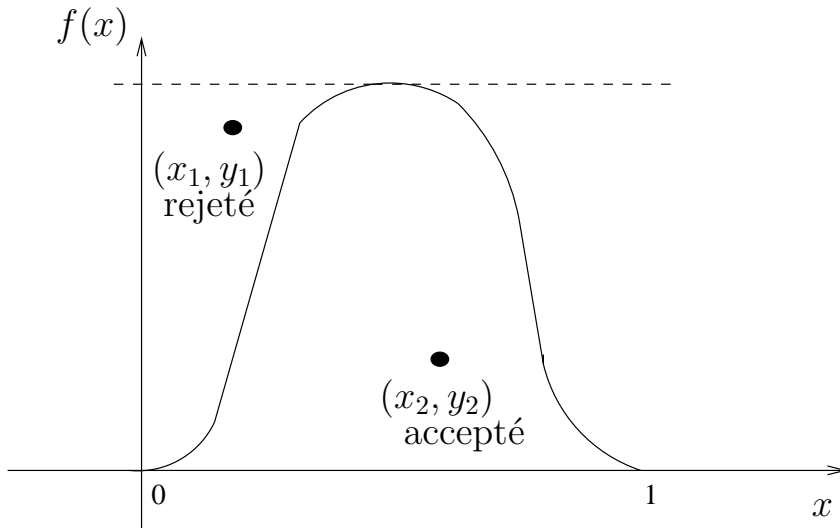


FIG. 2 – Algorithme du rejet.

**Lemme 2.3** *Méthode de Box-Muller.*

Si  $U_1$  et  $U_2$  sont indépendantes et de loi  $\mathcal{U}([0, 1])$ , alors

$$(\sqrt{-2\log(U_1)} \cos(2\pi U_2), \sqrt{-2\log(U_1)} \sin(2\pi U_2))$$

est un couple de variables indépendantes de loi  $\mathcal{N}(0, 1)$ .

A partir de la loi  $\mathcal{N}(0, 1)$  on peut évidemment passer à n'importe quel autre type de gaussiennes puisque si  $X \sim \mathcal{N}(0, 1)$ ,  $aX + b \sim \mathcal{N}(b, a^2)$ .

## 2.4 La simulation dans Matlab. Stibox.

Dans la version de base, les seuls générateurs sont les générateurs de loi uniforme  $\mathcal{U}([0, 1])$  (`rand`) et de loi  $\mathcal{N}(0, 1)$  (`randn`). On trouve aussi dans la version standard les fonctions `erf` et `erfinv` grâce auxquelles on obtient des approximations de la fonction de répartition  $\Phi$  de la loi  $\mathcal{N}(0, 1)$ , et de son inverse  $\Phi^{-1}$ .

$$\begin{aligned} \forall x \geq 0, \quad \operatorname{erf}(x) &\triangleq \frac{2}{\sqrt{\pi}} \int_0^x \exp(-y^2) dy, \\ &= 2 \int_0^{x\sqrt{2}} \exp\left(-\frac{t^2}{2}\right) \frac{dt}{\sqrt{2\pi}}, \\ &= P[|X| \leq x\sqrt{2}], \text{ où } X \sim \mathcal{N}(0, 1), \\ &= 2\Phi(x\sqrt{2}) - 1. \\ \forall x \leq 0, \quad \operatorname{erf}(x) &= -\operatorname{erf}(-x). \\ \forall y \in ]-1, +1[, \quad \operatorname{erfinv}(y) &= \operatorname{erf}^{-1}(y). \end{aligned}$$

Ainsi,

$$\begin{aligned} \forall x \in \mathbb{R}, \quad \Phi(x) &= \frac{1 + \operatorname{erf}(x/\sqrt{2})}{2}, \\ \forall y \in ]0, 1[, \quad \Phi^{-1}(y) &= \sqrt{2} \operatorname{erfinv}(2y - 1). \end{aligned}$$

Dans la boîte à outils `Stixbox`, on trouve les densités, les fonctions de répartition, les fonctions quantiles, et les générateurs aléatoires de la plupart des lois classiques. La plupart de ces générateurs utilisent la méthode par inversion de la fonction de répartition. Les exceptions sont les générateurs de préfixe `rj` (à savoir `rjbinom`, `rjpoiss`, `rjgamma`) qui utilisent la méthode de rejet, ainsi que les générateurs `rnorm`, `rchisq`, `rgeom`, `rpoiss`.

Le générateur `rnorm` utilise le générateur `randn` de Matlab, et se contente de passer d'une  $\mathcal{N}(0, 1)$  à une normale quelconque.

Le générateur `rchisq` utilise le générateur `rgamma`, la loi du  $\chi^2(d)$  étant en fait la loi Gamma  $\Gamma(d/2, 1/2)$ .

Le générateur `rpoiss` utilise la méthode de renouvellement. Soit  $(T_i)_{i \geq 1}$  une suite de v.a.i.i.d de loi  $\mathcal{E}(\lambda)$ . Soit  $S_i = \sum_{j=1}^i T_j$ . Soit  $N_t$  le nombre de  $S_i$  inférieurs à  $t$ ,

$$N_t \triangleq \sum_{i=1}^{\infty} \mathbb{1}_{S_i \leq t} .$$

$N_t$  suit une loi de Poisson de paramètre  $\lambda t$ .

Le générateur `rgeom` génère une variable  $X$  telle que

$$P(X = k) = p(1 - p)^k, \quad p \in [0, 1], \quad k \in \mathbb{N} . \quad (1)$$

Attention! Il y a un décalage de 1 par rapport à la définition classique de la loi géométrique. Le générateur `rgeom` utilise le fait que si  $X \sim \mathcal{E}(\lambda)$ ,  $[X]$  suit la loi donnée par (1) avec  $1 - p = e^{-\lambda}$ .

### Fonctions relatives aux lois de probabilités dans `Stixbox`.

Famille de lois	densité	Fonction de répartition	Fonction quantile	Générateur
Béta	<code>dbeta</code>	<code>pbeta</code>	<code>qbeta</code>	<code>rbeta</code>
Binomiale	<code>dbinom</code>	<code>pbinom</code>	<code>qbinom</code>	<code>rbinom</code> ou <code>rjbinom</code>
Chi-deux	<code>dchisq</code>	<code>pchisq</code>	<code>qchisq</code>	<code>rchisq</code>
Fisher	<code>df</code>	<code>pf</code>	<code>qf</code>	<code>rf</code>
Gamma	<code>dgamma</code>	<code>pgamma</code>	<code>qgamma</code>	<code>rgamma</code> ou <code>rjgamma</code>
Géométrique				<code>rgeom</code>
Hypergéométrique	<code>dhypg</code>	<code>phypg</code>	<code>qhypg</code>	<code>rhypg</code>
Kolmogorov-Smirnov		<code>pks</code>		
Normale	<code>dnorm</code>	<code>pnorm</code>	<code>qnorm</code>	<code>rnorm</code>
Poisson				<code>rpoiss</code> ou <code>rjpoiss</code>
Student	<code>dt</code>	<code>pt</code>	<code>qt</code>	<code>rt</code>

### 3 T.Ps

#### 3.1 Exercice 1. Génération d'une loi de Cauchy par la méthode d'inversion.

La loi de Cauchy est la loi de probabilité sur  $\mathbb{R}$ , qui admet pour densité par rapport à la mesure de Lebesgue  $\frac{1}{\pi(1+x^2)}$ . Une variable  $X$  de Cauchy a donc pour fonction de répartition  $F_X(t) = \frac{1}{\pi} \arctan t + \frac{1}{2}$ , et pour fonction de répartition inverse  $F_X^{-1}(u) = \tan(\pi(u - \frac{1}{2}))$ .

Écrire un générateur de la loi de Cauchy. Simuler un  $N$ -échantillon de cette loi. Tracer sur un même graphe un histogramme de cet échantillon et la densité de la loi de Cauchy.

Comment évolue la moyenne empirique de l'échantillon, quand  $N$  tend vers l'infini ? Expliquez pourquoi.

#### 3.2 Exercice 2. Génération de la loi uniforme sur le disque.

Écrire un générateur de la loi uniforme sur  $D = \{(x, y) \in \mathbb{R}^2; x^2 + y^2 \leq 1\}$ , par la méthode de rejet.

#### 3.3 Exercice 3. Différentes méthodes de génération de la loi normale.

1. Écrire un générateur de la loi  $\mathcal{N}(0, 1)$  utilisant la fonction `erfinv` de Matlab.
2. Écrire un générateur de la loi  $\mathcal{N}(0, 1)$  utilisant l'inégalité

$$\forall x \in \mathbb{R}, \frac{1}{\sqrt{2\pi}} \exp^{-\frac{x^2}{2}} \leq \sqrt{\frac{2\pi}{e}} \frac{1}{\pi} \frac{1}{1+x^2}.$$

3. Écrire un générateur de la loi  $\mathcal{N}(0, 1)$  utilisant l'inégalité

$$\forall x \in \mathbb{R}, \frac{1}{\sqrt{2\pi}} \exp^{-\frac{x^2}{2}} \leq 2\sqrt{\frac{e}{2\pi}} \frac{1}{2} \exp^{-|x|}.$$

4. Écrire un générateur de la loi  $\mathcal{N}(0, 1)$  par la méthode de Box-Muller.
5. Écrire un générateur de la loi  $\mathcal{N}(0, 1)$ , utilisant le théorème central-limite.
6. Comparer ces différents générateurs.

#### 3.4 Exercice 4. Comparaison de la méthode par inversion et la méthode du rejet sur la loi de Poisson.

Écrire un générateur `ripoiss` de la loi de Poisson de paramètre  $\lambda$ , par la méthode d'inversion. Pour différentes valeurs de  $\lambda$ , comparer ce générateur

aux générateurs `Stibox rjpoiss` (méthode de rejet), et `rpoiss` (méthode de renouvellement).

### 3.5 Exercice 5. Comparaison de la méthode d'inversion et de rejet. Suite.

Soit  $X$  la v.a. à valeurs dans  $\{1, \dots, n\}$ , de loi donnée par

$$\begin{aligned} P(X = 1) &= \frac{1}{2^{n-1}}, \\ P(X = k) &= \frac{2}{2^{n-1}}, k \in \{2, \dots, n\}. \end{aligned}$$

Ecrire un générateur de  $X$  par la méthode d'inversion et par la méthode de rejet. Les comparer pour différentes valeurs de  $n$ . Voyez-vous une autre méthode pour générer  $X$  ?

### 3.6 Exercice 6. Méthode de décomposition.

Cette méthode s'applique dès qu'on peut décomposer la densité de probabilité  $f$  sous la forme

$$f(x) = \sum_{k=1}^n p_k f_k,$$

où les  $f_k$  sont des densités de probabilité, et  $(p_k; 1 \leq k \leq n)$  est une probabilité sur  $\{1, \dots, n\}$ . Elle consiste à générer une variable  $Z$  de loi  $p$  et si  $Z = k$ , à générer une variable de densité  $f_k$ . L'algorithme

```

U ← Rand
If  $p_1 + \dots + p_{k-1} < U < p_1 + \dots + p_k$ 

    X ← Rand $f_k$ 

```

```

EndIf

```

rend une variable  $X$  de densité  $f$ . Cette méthode demande d'avoir à sa disposition des générateurs des densités  $f_k$ .

Écrire un générateur de la loi de densité

$$\frac{1}{2} \mathbb{I}_{0 \leq x \leq y \leq 1} + \frac{3}{2} \mathbb{I}_{0 \leq y \leq x \leq 1}.$$

### 3.7 Exercice 7. Génération de maxima.

Cette méthode s'applique lorsque la fonction de répartition  $F$  de la loi à générer peut s'écrire sous la forme

$$F(x) = \prod_{i=1}^N F_i(x),$$

où les  $F_i$  sont des fonctions de répartition. Elle s'appuie sur la remarque suivante : si les  $X_i$  ( $1 \leq i \leq N$ ) sont des variables indépendantes de fonction de répartition respective  $F_i$ ,  $X \triangleq \max(X_i)$  a pour fonction de répartition  $F$ .

Écrire deux générateurs de la loi de densité

$$(9t^2 - 8t^3) \mathbb{1}_{t \in [0,1]},$$

1. l'un utilisant le générateur `rbeta` de Stixbox ;
2. l'autre utilisant l'inégalité

$$(9t^2 - 8t^3) \mathbb{1}_{t \in [0,1]} \leq 9t^2 \mathbb{1}_{t \in [0,1]}.$$

Les comparer.

### 3.8 Exercice 8. Test du générateur d'uniformes.

**Description du test.** Soient  $U_i$  une suite de v.a. indépendantes et de même loi  $\mathcal{U}([0, 1])$ . Pour tout entier  $k$ , et tout  $k$ -uplé  $t = (t_1, \dots, t_k) \in [0, 1]^k$ , la loi des grands nombres affirme que

$$P(N, t, U) \triangleq \frac{1}{N} \sum_{i=1}^N \mathbb{1}_{[0, t_1] \times \dots \times [0, t_k]}(U_{(i-1)k+1}, \dots, U_{ik}) \xrightarrow[N \rightarrow +\infty]{\text{p.s.}} t_1 \cdots t_k. \quad (2)$$

De plus le théorème central limite dit que

$$\frac{\sqrt{N} (P(N, t, U) - t_1 \cdots t_k)}{\sqrt{(t_1 \cdots t_k)(1 - t_1 \cdots t_k)}} \xrightarrow[N \rightarrow +\infty]{\text{loi}} Z, \text{ où } Z \sim \mathcal{N}(0, 1). \quad (3)$$

Autrement dit, pour tout  $z \in \mathbb{R}^+$ , et pour  $N$  grand

$$P \left[ \left| \frac{\sqrt{N} (P(N, t, U) - t_1 \cdots t_k)}{\sqrt{(t_1 \cdots t_k)(1 - t_1 \cdots t_k)}} \right| \leq z \right] \simeq P[|Z| \leq z] = \text{erf}(\sqrt{2}z).$$

Pour  $\alpha$  petit (5 % par exemple), et pour  $N$  grand, on a donc

$$P \left[ \left| \frac{\sqrt{N} (P(N, t, u) - t_1 \cdots t_k)}{\sqrt{(t_1 \cdots t_k)(1 - t_1 \cdots t_k)}} \right| \leq \frac{\text{erfinv}(1 - \alpha)}{\sqrt{2}} \right] \simeq 1 - \alpha. \quad (4)$$

Si le générateur `rand` produit une suite  $(u_i)$  pour laquelle on peut trouver  $N$  grand,  $k \in \mathbb{N}$ , et  $t \in [0, 1]^k$  tels que

$$\left| \frac{\sqrt{N} (P(N, t, u) - t_1 \cdots t_k)}{\sqrt{(t_1 \cdots t_k)(1 - t_1 \cdots t_k)}} \right| > \frac{\text{erfinv}(1 - \alpha)}{\sqrt{2}},$$

on peut conclure avec  $\alpha$  chances de se tromper, que  $(u_i)$  n'est pas une réalisation d'une suite de v.a. indépendantes de loi  $\mathcal{U}([0, 1])$ .

**Implémentation.**

1/ Ecrire une fonction  $p = \text{proportion}(U, t)$ , qui prend comme argument une matrice  $U$  de dimension  $nU \times nSim$ , un vecteur  $t$  de dimension  $nt$ , et rend un vecteur  $p$  de dimension  $nU$ , dont la coordonnée n°  $k$  est  $p(k) = P(nU, t, U(:, k))$  (où  $P(N, t, U)$  est défini dans (2)).

2/ Pour différentes valeurs de  $nSim$ ,  $nU$ ,  $t$ ,  $\alpha$ , générer une matrice  $U$  de taille  $nU \times nSim$  d'uniformes sur  $[0,1]$  par la fonction **rand** de Matlab. Calculer  $p = \text{proportion}(U, t)$  et les bornes de l'intervalle de confiance donné par (4). Qu'en concluez-vous quant à la qualité de **rand** ?