Autour des v.a.r. : Annales 2015 / 2016 / 2017 / 2018 / 2019

I. Simulation d'expériences aléatoires et de variables aléatoires

I.1. Simulation de variables aléatoires

EML – 2015

- On considère une v.a.r. U tel que : $U \hookrightarrow \mathcal{U}([0,1[)$.
- On démontre que la v.a.r. $V = -\dfrac{1}{\lambda}\ln(1-U)$ est telle que $V \hookrightarrow \mathcal{E}(\lambda)$.

a) Écrire une fonction en **Scilab** qui, étant donné un réel $\lambda > 0$, simule la loi exponentielle de paramètre λ.

Démonstration.

```
function v = simuExp(lambda)
    u = rand()
    v = -(1/lambda) * log(1-u)
endfunction
```

□

EDHEC – 2016

- On désigne par p un réel de $]0,1[$.
- On considère deux variables aléatoires indépendantes U et V, telles que U suit la loi uniforme sur $[-3,1]$, et V suit la loi uniforme sur $[-1,3]$.
- On considère également une variable aléatoire Z, indépendante de U et V, dont la loi est donnée par :

$$\mathbb{P}([Z=1]) = p \quad \text{et} \quad \mathbb{P}([Z=-1]) = 1-p$$

Enfin, on note X la v.a.r. définie par :

$$\forall \omega \in \Omega,\ X(\omega) = \begin{cases} U(\omega) & \text{si} \quad Z(\omega) = 1 \\ V(\omega) & \text{si} \quad Z(\omega) = -1 \end{cases}$$

a) Vérifier que l'on a :

$$X = U\,\frac{1+Z}{2} + V\,\frac{1-Z}{2}$$

Démonstration.
Soit $\omega \in \Omega$. Deux cas se présentent.

- Si $Z(\omega) = 1$ alors $X(\omega) = U(\omega)$ et :

$$U(\omega)\,\frac{1+Z(\omega)}{2} + V(\omega)\,\frac{1-Z(\omega)}{2} = U(\omega)\,\frac{1+1}{2} + V(\omega)\,\frac{1-1}{2} = U(\omega) = X(\omega)$$

- Si $Z(\omega) \neq 1$ alors $Z(\omega) = -1$, $X(\omega) = V(\omega)$ et :

$$U(\omega)\,\frac{1+Z(\omega)}{2} + V(\omega)\,\frac{1-Z(\omega)}{2} = U(\omega)\,\frac{1-1}{2} + V(\omega)\,\frac{1+1}{2} = V(\omega) = X(\omega)$$

Ainsi, pour tout $\omega \in \Omega$, $X(\omega) = U(\omega)\,\dfrac{1+Z(\omega)}{2} + V(\omega)\,\dfrac{1-Z(\omega)}{2}$.

□

b) Soit T une variable aléatoire suivant la loi de Bernoulli de paramètre p.
Déterminer la loi de $2T-1$.

Démonstration.

- Comme $T \hookrightarrow \mathcal{B}(p)$, $T(\Omega) = \{0,1\}$. On en déduit :

$$\begin{aligned}(2T-1)(\Omega) &= \{2u-1 \mid u \in T(\Omega)\}\\ &= \{2\times(-1)-1, 2\times -1\}\\ &= \{-1,1\}\end{aligned}$$

$$\boxed{(2T-1)(\Omega) = \{-1,1\}}$$

- De plus :

$$\mathbb{P}([2T-1=-1]) = \mathbb{P}([2T=0]) = \mathbb{P}([T=0]) = 1-p$$

$$\mathbb{P}([2T-1=1]) = \mathbb{P}([2T=2]) = \mathbb{P}([T=1]) = p$$

$$\boxed{\text{Ainsi, } 2T-1 \text{ et } Z \text{ suivent la même loi.}}$$

Commentaire

- Dire que deux v.a.r. discrètes Z_1 et Z_2 suivent la même loi ne signifie pas que $Z_1 = Z_2$.
  Cela ne signifie pas non plus que $\mathbb{P}([Z_1 = Z_2]) = 1$.
  Cela signifie simplement :
  - $Z_1(\Omega) = Z_2(\Omega)$,
  - $\forall z \in Z_1(\Omega)$, $\mathbb{P}([Z_1 = z]) = \mathbb{P}([Z_2 = z])$.
- Comme $Z(\Omega) = \{-1,1\} \neq \{0,1\}$, la v.a.r. Z ne suit pas une loi de Bernoulli.
  De manière générale, une v.a.r. qui ne prend que deux valeurs ne suit pas pour autant une loi de Bernoulli. Pour que ce soit le cas, il faut que ces deux valeurs soient 0 et 1.

□

c) On rappelle que `grand(1,1,'unf',a,b)` et `grand(1,1,'bin',p)`sont des commandes `Scilab` permettant de simuler respectivement une variable aléatoire à densité suivant la loi uniforme sur $[a,b]$ et une variable aléatoire suivant la loi de Bernoulli de paramètre p.
Écrire des commandes `Scilab` permettant de simuler U, V, Z, puis X.

Démonstration.

- Pour simuler les v.a.r. U, V et T, il suffit d'utiliser les instructions données.

```
1 U = grand(1,1,'unf',-3,1)
2 V = grand(1,1,'unf',-1,3)
3 T = grand(1,1,'bin',p)
```

- Pour les v.a.r. Z et X on utilise les questions ***3.a)*** et ***4.a)***.

```
4 Z = 2 * T - 1
5 X = U * (1+Z)/2 + V * (1-Z)/2
```

Commentaire

- Ces quelques lignes ne sont utilisables que si `p` est préalablement défini. Pour ce faire, on peut ajouter une ligne : `p = input('Entrer une valeur pour p : ')` en début de programme afin que l'utilisateur entre une valeur pour `p`.
  Une autre manière de régler ce problème est de transformer ce programme en une fonction de paramètre `p`.
- Dans le programme, on a écrit : `Z = 2 * T - 1`. Autrement dit, on a décidé de simuler la v.a.r. Z en lui donnant pour valeurs celles d'une v.a.r. qui suit la même loi. Au-delà de la v.a.r. , c'est ici la loi qu'on cherche à simuler. Ainsi, n'importe quelle v.a.r. qui suit cette loi fait l'affaire.
- Comme dit précédemment, la v.a.r. `Z` ne suit pas une loi de Bernoulli.
  Toutefois, on a vu en question précédente :

$$\mathbb{P}([Z=-1]) = \mathbb{P}([T=0]) = 1-p \quad \text{et} \quad \mathbb{P}([Z=1]) = \mathbb{P}([T=1]) = p$$

  On peut exploiter ce point pour écrire `Z` à l'aide d'une structure conditionnelle :

```
1  T = grand(1,1,'bin',p)
2  if T == 0 then
3      Z = -1
4  else
5      Z = 1
6  end
```

- Une fois `U`, `V` et `Z` codées, on peut aussi coder `X` en se servant de sa définition initiale.

```
5  if Z == 1 then
6      X = U
7  else
8      X = V
9  end
```

□

EDHEC – 2016

- On considère une suite $(X_n)_{n\in\mathbb{N}^*}$ de variables aléatoires, mutuellement indépendantes, suivant toutes la loi géométrique de paramètre x, et on pose $S_n = \sum_{k=1}^{n} X_k$ pour tout $n \in \mathbb{N}^*$.

a) On rappelle que la commande `grand(1,n,'geom',p)` permet à **Scilab** de simuler n variables aléatoires indépendantes suivant toutes la loi géométrique de paramètre p.
Compléter les commandes **Scilab** suivantes pour qu'elles simulent la variable aléatoire S_n.

```
1  n = input('entrez une valeur de n supérieure à 1 : ')
2  S = ---
3  disp(S)
```

Démonstration.

- On rappelle que $S_n = \sum_{k=1}^{n} X_k$ où les v.a.r. $X_1, \ldots, X_n$ sont indépendantes et suivent toutes la même loi $\mathcal{G}(x)$. L'instruction `grand(1,n,'geom',p)` permet de générer une matrice ligne $[x_1, \ldots, x_n]$ qui simule le n-échantillon $(X_1, \ldots, X_n)$.

- Pour simuler S_n, il suffit de faire la somme des coefficients de cette matrice ligne.

```
2  S = sum(grand(1,n,'geom',p))
```

Commentaire

Il est possible (mais moins élégant) de réaliser cette somme à l'aide d'une structure itérative. On obtient le programme suivant.

```
2  S = 0
3  tab = grand(1,n,'geom',p)
4  for i = 1:n
5      S = S + tab(i)
6  end
```

□

EDHEC – 2017

- Soit V une variable aléatoire suivant la loi exponentielle de paramètre 1, dont la fonction de répartition est la fonction F_V définie par : $F_V(x) = \begin{cases} 0 & \text{si } x \leqslant 0 \\ 1 - \mathrm{e}^{-x} & \text{si } x > 0 \end{cases}$.
  On pose $W = -\ln(V)$ et on admet que W est aussi une variable aléatoire dont le fonction de répartition est notée F_W. On dit que W suit une loi de Gumbel.
- On désigne par n un entier naturel non nul et par $X_1, \ldots, X_n$ des variables aléatoires définies sur le même espace probabilisé, indépendantes et suivant la loi $\mathcal{E}(1)$.
- On considère la variable aléatoire Y_n définie par $Y_n = \max(X_1, X_2, \ldots, X_n)$, c'est à dire que pour tout ω de Ω, on a : $Y_n(\omega) = \max(X_1(\omega), X_2(\omega), \ldots, X_n(\omega))$.
  On admet que Y_n est une variable aléatoire à densité.

a) On pose $Z_n = Y_n - \ln(n)$.
On rappelle que `grand(1,n,'exp',1)` simule n variables aléatoires indépendantes et suivant toutes la loi exponentielle de paramètre 1. Compléter la déclaration de fonction `Scilab` suivante afin qu'elle simule la variable aléatoire Z_n.

```
1  function Z = f(n)
2      x = grand(1,n,'exp',1)
3      Z = ---
4  endfunction
```

Démonstration.

```
3      Z = max(x) - log(n)
```

Commentaire

- On rappelle qu'il est inutile de recopier le programme en entier.
  Écrire la ligne contenant l'information manquante suffit.
- Il est tout à fait possible (et donc non sanctionné) aux concours d'utiliser plusieurs lignes, même si le concepteur a pensé à une réponse sur une seule ligne. Ici, on pouvait dans un premier temps simuler la v.a.r. Y_n puis la v.a.r. Z_n.

```
3        Y = max(x)
4        Z = Y - log(n)
```

□

b) Voici deux scripts :

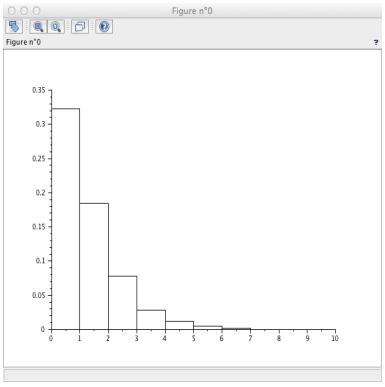
```
1  V = grand(1,10000,'exp',1)
2  W = -log(V)
3  s = linspace(0,10,11)
4  histplot(s,W)
```

Script (1)

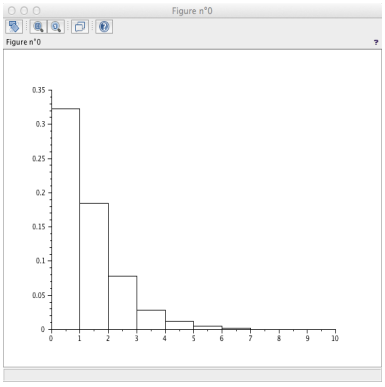
```
1  n = input('entrez la valeur de n : ')
2  Z = [] // la matrice-ligne Z est vide
3  for k = 1 :10000
4      Z = [Z,f(n)]
5  end
6  s = linspace(0,10,11)
7  histplot(s,Z)
```

Script (2)

Chacun des scripts simule 10000 v.a.r. indépendantes, regroupe les valeurs renvoyées en 10 classes qui sont les intervalles $[0,1]$, $]1,2]$, $]2,3]$, …, $]9,10]$ et trace l'histogramme correspondant (la largeur de chaque rectangle est égale à 1 et leur hauteur est proportionnelle à l'effectif de chaque classe). Le script (1) dans lequel les v.a.r. suivent la loi de Gumbel (loi suivie par W), renvoie l'histogramme (1) ci-dessous, alors que le script (2) dans lequel les v.a.r. suivent la même loi que Z_n, renvoie l'histogramme (2) ci-dessous, pour lequel on a choisi $n = 1000$.



Histogramme (1)



Histogramme (2) pour $n = 1000$

Quelle conjecture peut-on émettre quant au comportement de la suite des v.a.r. (Z_n) ?

Démonstration.
Commentons tout d'abord le script et l'histogramme (1).

- Les lignes 1 et 2 permettent d'obtenir des valeurs $(w_1, \ldots, w_{10000})$ qui correspondent à l'observation d'un 10000-échantillon $(W_1, \ldots, W_{10000})$ de la v.a.r. W qui suit la loi de Gumbel.
  (les v.a.r. W_i sont indépendantes et ont même loi que W)
- Les lignes 3 et 4 ont pour but de permettre de visualiser la répartition des 10000 valeurs $(w_1, \ldots, w_{10000})$ à l'aide d'un histogramme des fréquences :

- × l'instruction `linspace(0, 10, 11)` crée la matrice `[0, 1, 2, ..., 10]`.
- × l'instruction `histplot` crée les classes : $[0,1]$, $]1,2]$, $\ldots$, $]9,10]$.
  Elle permet aussi de récupérer l'effectif de chaque classe (*i.e.* le nombre de w_i dans chaque classe) et trace l'histogramme (1).

- Considérons par exemple la classe définie par l'intervalle $]2,3]$.
  La loi faible des grands nombres (LfGN) permet d'affirmer :

$$\text{fréquence de la classe }]2,3] = \frac{\text{effectif de la classe }]2,3]}{\text{taille de l'observation}} \simeq \mathbb{P}([2 < W \leqslant 3])$$

  Ici, on réalise bien un grand nombre d'observations ($N = 10000$) ce qui justifie cette formule. Ainsi, l'aire de la barre qui s'appuie sur l'intervalle $]2,3]$ est donc une approximation de $\mathbb{P}([2 < W \leqslant 3]) = F_W(3) - F_W(2)$.

Commentons maintenant l'histogramme (2).

- Les lignes 3, 4, et 5 permettent d'obtenir les valeurs $(u_1, \ldots, u_{10000})$ qui correspondent à l'observation d'un 10000-échantillon $(U_1, \ldots, U_{10000})$ de la variable Z_n (pour $n = 1000$).
  (les U_i sont indépendantes et ont même loi que Z_n)
- On trace alors l'histogramme de répartition de ces valeurs. Pour les raisons évoquées ci-dessus, l'aire de la barre du graphique (2) est une valeur approchée de :

$$\mathbb{P}([2 < Z_n \leqslant 3]) = F_{Z_n}(3) - F_{Z_n}(2)$$

Or, on constate que l'histogramme (2) est similaire à l'histogramme (1).
Cela signifie que les aires des barres de chacun de ces deux graphiques sont très proches. Ainsi :

$$F_{Z_n}(3) - F_{Z_n}(2) \simeq F_W(3) - F_W(2)$$

En considérant la première classe, on observe que : $F_{Z_n}(1) \simeq F_W(1)$.
On obtient alors, en considérant successivement toutes les classes :

$$\forall i \in [\![1,10]\!],\ F_{Z_n}(i) \simeq F_W(i)$$

Les fonctions de répartition F_{Z_n} et F_W coïncident en ces 10 points. En considérant des classes définies par d'autres points, on observerait que les fonctions coïncident en ces nouveaux points. Ainsi, lorsque $n = 1000$, les fonctions de répartition des v.a.r. W et Z_n sont très proches.

On conjecture que la suite de v.a.r. (Z_n) converge en loi vers la v.a.r. W.

Commentaire

- Ces deux histogrammes sont normalisés. De ce fait, ce n'est pas l'effectif de la classe qui est affiché en ordonnée mais un nombre qui, une fois multiplié par la largeur de la barre, fournit la fréquence de la classe. Autrement dit, dans un tel histogramme, la fréquence d'une classe c'est l'aire de la barre correspondante.
  Ici, chaque barre est de largeur 1. Ce sont donc les fréquences de chaque classe que l'on peut lire en ordonnée. Il ne faut pas oublier de prendre en compte ce coefficient multiplicatif lorsque l'on considère un nombre de barres plus grand (et donc des largeurs de barres différentes).
- Dans la démonstration, on a utilisé la loi faible des grands nombres (LfGN) afin de faire le lien entre fréquence de la classe $]2,3]$ et probabilité $\mathbb{P}([2<W\leqslant 3])$. Établissons ce lien de manière plus précise.
  Pour ce faire, on introduit la v.a.r. T suivante.

$$\begin{array}{rccl} T & : & \Omega & \to & \mathbb{R} \\ & & \omega & \mapsto & \left\{\begin{array}{ll} 1 & \text{si } W(\omega)\in]2,3] \\ 0 & \text{sinon}\end{array}\right.\end{array}$$

  Le N-échantillon d'observations $(w_1,\dots,w_N)$ (où N est un grand nombre) de la v.a.r. W fournit un N-échantillon d'observations $(t_1,\dots,t_N)$ de la v.a.r. T.
  La LfGN stipule :

$$\frac{1}{N}\sum_{i=1}^{N} t_i \simeq \mathbb{E}(T)$$

  Or T est une v.a.r. finie qui admet pour espérance :

$$\begin{array}{rcl}\mathbb{E}(T) & = & 1\times\mathbb{P}([2<W\leqslant 3])+\cancel{0\times\mathbb{P}([2<W\leqslant 3])} \\ & = & \mathbb{P}([2<W\leqslant 3]) = F_W(3)-F_W(2)\end{array}$$

  Par ailleurs, $\sum_{i=1}^{N} t_i$ permet de compter le nombre d'observations qui appartiennent à la classe $]2,3]$ (*i.e.* l'effectif de la classe $]2,3]$).
  Ainsi, $\frac{1}{N}\sum_{i=1}^{N} t_i$ est la fréquence de cette classe. Celle-ci est représentée graphiquement par la troisième barre de l'histogramme (1). D'après ce qui précède, l'aire de cette barre est une valeur approchée de $\mathbb{P}([2<W\leqslant 3])$.

□

HEC – 2017

- On note :
  - × a et b deux réels strictement positifs ;
  - × $(\Omega,\mathcal{A},\mathbb{P})$ un espace probabilisé sur lequel sont définies toutes les variables aléatoires du problème ;
  - × $G_{a,b}$ la fonction définie sur $\mathbb{R}_+$ par : $G_{a,b}(x)=\exp\left(-ax-\frac{b}{2}x^2\right)$.
- Pour tout $a>0$ et pour tout $b>0$, on pose : $f_{a,b}(x)=\left\{\begin{array}{ll}(a+bx)\ \exp\left(-ax-\frac{b}{2}x^2\right) & \text{si } x\geqslant 0 \\ 0 & \text{si } x<0\end{array}\right.$.
  On dit qu'une variable aléatoire suit la loi exponentielle linéaire de paramètres a et b, notée $\mathcal{E}_\ell(a,b)$, si elle admet $f_{a,b}$ pour densité.
- Soit Y une variable aléatoire suivant la loi exponentielle de paramètre 1.

On pose : $X = \dfrac{-a + \sqrt{a^2 + 2bY}}{b}$. On montre alors que X suit la loi $\mathcal{E}_\ell(a, b)$.

- On montre ensuite que, si U est une variable aléatoire suivant la loi uniforme sur $[0, 1[$, alors $Y = -\ln(1 - U)$ suit la loi $\mathcal{E}(1)$.

a) La fonction **Scilab** suivante génère des simulations de la loi exponentielle linéaire.

```
1  function x = grandlinexp(a,b,n)
2      u = rand(n,1)
3      y = ............
4      x = (-a + sqrt(a^2 + 2 * b * y)) / b
5  endfunction
```

Quelle est la signification de la ligne de code 2 ?

Démonstration.

L'instruction `rand(n,1)` renvoie un vecteur colonne de taille $\mathtt{n} \times 1$ contenant le résultat de la simulation de `n` v.a.r. aléatoires indépendantes qui suivent toutes la même loi $\mathcal{U}([0, 1])$. □

b) Compléter la ligne de code 3 pour que la fonction `grandlinexp` génère les simulations désirées.

Démonstration.
D'après la question ***4.c)***, il suffit d'écrire :

```
3      y = - log(1 - u)
```

□

c) De quel nombre réel peut-on penser que les six valeurs générées par la boucle **Scilab** suivante fourniront des valeurs approchées de plus en plus précises et pourquoi ?

```
1  for k = 1:6
2      mean(grandlinexp(0, 1, 10^k)
3  end
```

Démonstration.

- Soit X une v.a.r. telle que $X \hookrightarrow \mathcal{E}_\ell(0, 1)$.
  Pour $m \in \mathbb{N}^*$, on considère $(X_1, \ldots, X_m)$ un m-échantillon de la v.a.r. X.
  Autrement dit, on considère $(X_i)_{i \in \mathbb{N}^*}$ une suite de v.a.r. indépendantes et toutes de même loi $\mathcal{E}_\ell(0, 1)$. On note alors :
  $$\overline{X_m} = \frac{X_1 + \ldots + X_m}{m}$$
  la v.a.r. donnant la moyenne empirique associée à v.a.r. X.
- Pour chaque `k`, l'instruction `mean(grandlinexp(0, 1, 10^k))` permet de simuler la v.a.r. $\overline{X_{10^k}}$.
- En vertu de la loi faible des grands nombres, la v.a.r. $\overline{X_{10^k}}$ converge en probabilité vers la variable aléatoire constante égale à $\mathbb{E}(X)$.
- L'entier `k` prenant des valeurs de plus en plus grandes (on considère une simulation de $\overline{X_{10}}$, puis $\overline{X_{100}}$, …, puis $\overline{X_{1000000}}$), on peut penser que le résultat sera de plus en plus proche de $\mathbb{E}(X)$.

Les six valeurs générées par la boucle **Scilab** fourniront des valeurs de plus en plus précises de $\mathbb{E}(X)$. □

EDHEC – 2019

- Soit $\theta \in]0, \frac{1}{2}[$. On considère une v.a.r. X à valeurs strictement positves, de densité :

$$f : x \mapsto \begin{cases} \dfrac{1}{\theta\, x^{1+\frac{1}{\theta}}} & \text{si } x \geqslant 1 \\ 0 & \text{si } x < 1 \end{cases}$$

- On devait démontrer : $Y = \ln(X) \hookrightarrow \mathcal{E}\left(\frac{1}{\theta}\right)$.
- On rappelle qu'en **Scilab**, la commande `grand(1, 1, 'exp', 1/lambda)` simule une variable aléatoire suivant la loi exponentielle de paramètre λ.

Écrire des commandes **Scilab** utilisant `grand` et permettant de simuler X.

Démonstration.
D'après la question précédente, si une v.a.r. Y suit une loi $\mathcal{E}\left(\dfrac{1}{\theta}\right)$, alors la v.a.r. $\exp(Y)$ suit la même loi que la v.a.r. X. On propose donc le programme suivant :

```
1 theta = input('Entrez un paramètre theta :')
2 Y = grand(1, 1, 'exp', theta)
3 X = exp(X)
```

Commentaire

- Il n'est pas précisé par l'énoncé si le paramètre θ a déjà été fixé par l'utilisateur. C'est pourquoi on ajoute la ligne 1. Il n'est cependant pas certain que son oubli soit sanctionné.
- On prendra garde à syntaxe particulière de la fonction `grand` dans le cas d'une loi exponentielle (syntaxe précisée par l'énoncé). En effet, pour obtenir une simulation d'une loi $\mathcal{E}(\lambda)$, on entrera la commande `grand(1, 1, 'exp', 1/lambda)` (et non `grand(1, 1, 'exp', lambda)`).
- Ainsi, dans notre cas, pour simuler une loi $\mathcal{E}\left(\dfrac{1}{\theta}\right)$, on utilisera bien la commande `grand(1, 1, 'exp', theta)` (et non `grand(1, 1, 'exp', 1/theta)`).

□

ESSEC – I – 2017

- Soit $\alpha \in \mathbb{R}$ et $\beta > 0$. On dit qu'une variable aléatoire réelle à densité suit une loi de Laplace de paramètre (α, β), notée $\mathcal{L}(\alpha, \beta)$, si elle admet comme densité la fonction f donnée par :

$$\forall t \in \mathbb{R}, \quad f(t) = \frac{1}{2\beta} \exp\left(-\frac{|t-\alpha|}{\beta}\right)$$

- On suppose que X suit la loi $\mathcal{L}(0, 1)$.
  Montrer que $\beta X + \alpha$ suit la loi $\mathcal{L}(\alpha, \beta)$.
- Soit U une variable aléatoire qui suit la loi exponentielle de paramètre 1 et V une variable aléatoire qui suit la loi de Bernoulli de paramètre $\dfrac{1}{2}$ et indépendante de U.
  On montre que $X = (2V - 1)U$ suit la loi $\mathcal{L}(0, 1)$.

a) Compléter la définition Scilab ci-dessous pour que la fonction ainsi définie réalise la simulation d'une variable aléatoire qui suit la loi $\mathcal{L}(\alpha, \beta)$:

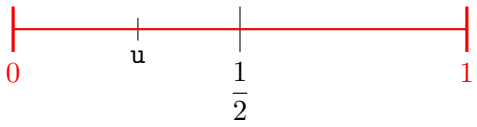
```
1  function r = Laplace(alpha,beta)
2      if ... <= 1/2
3          V = 1
4      else
5          V = 0
6      end
7      X = (2 * V - 1) * grand(1, 1, 'exp', 1)
8      r = ...
9  endfunction
```

L'énoncé donnait l'aide ***Scilab*** *suivante en fin de sujet :*

> **Aide Scilab.** La fonction Scilab `grand` permet de simuler, en particulier, les lois exponentielles et uniformes discrètes. Par exemple :
>
> × `grand(3,2,'exp',0.5)` renvoie une matrice aléatoire $(3,2)$ dont les coefficients sont des variables indépendantes qui suivent la loi exponentielle d'espérance $0,5$.
>
> × `grand(1,2,'uin',-1,3)` renvoie une matrice aléatoire $(1,2)$ dont les coefficients sont des variables indépendantes qui suivent la loi uniforme discrètes sur $\mathcal{U}([\![-1,3]\!])$.

Démonstration.

- La fonction `rand` permet de simuler une v.a.r. U telle que $U \hookrightarrow \mathcal{U}([0,1])$. Afin de simuler une v.a.r. V telle que $V \hookrightarrow \mathcal{B}\left(\frac{1}{2}\right)$ à l'aide de `rand`, on procède comme suit. L'appel `rand()` renvoie un réel `u` choisit aléatoirement dans $[0,1]$:



  Le réel `u` appartient à l'intervalle $[0,\frac{1}{2}]$ avec probabilité :

$$\mathbb{P}\left(\left[U \in [0,\tfrac{1}{2}]\right]\right) = \mathbb{P}\left(\left[U \leqslant \frac{1}{2}\right]\right) = \frac{1}{2} = \mathbb{P}([V=1])$$

  Le réel `u` appartient à l'intervalle $]\frac{1}{2},1]$ avec probabilité :

$$\mathbb{P}\left(\left[U \in]\tfrac{1}{2},1]\right]\right) = \mathbb{P}\left(\left[U > \frac{1}{2}\right]\right) = \frac{1}{2} = \mathbb{P}([V=0])$$

  C'est ce que réalise le programme demandé en complétant la ligne 2 comme suit :

```
2      if rand() <= 1/2
```

- L'instruction `grand(1, 1, 'exp', 1)` permet de simuler une v.a.r. U telle que $U \hookrightarrow \mathcal{E}(1)$.
- D'après la question qui précède, l'instruction `X = (2 * V - 1) * grand(1, 1, 'exp', 1)` permet de simuler une v.a.r. X telle que $X \hookrightarrow \mathcal{L}(0,1)$.
- Enfin, d'après la question ***3.a)***, la v.a.r. $\beta X + \alpha \hookrightarrow \mathcal{L}(\alpha,\beta)$.
  On en déduit la ligne 8 du programme à compléter :

```
8      r = beta * X + alpha
```

Commentaire

Afin de permettre une bonne compréhension des mécanismes en jeu dans cette question, on a détaillé sa réponse. Cependant, compléter correctement le programme **Scilab** démontre la bonne compréhension de la simulation demandée et permet certainement d'obtenir tous les points alloués à cette question. □

EDHEC – 2018

- Soit a un réel strictement positif et f la fonction définie par : $f(x) = \begin{cases} \frac{x}{a}\,\mathrm{e}^{-\frac{x^2}{2a}} & \text{si } x \geqslant 0 \\ 0 & \text{si } x < 0 \end{cases}$.

1. Montrer que la fonction f est une densité.
Dans la suite de l'exercice, on considère une variable aléatoire X de densité f.

Démonstration.

- Soit $x \in \mathbb{R}$. Deux cas se présentent.
  - Si $x \geqslant 0$: $\frac{x}{a} \geqslant 0$ car $a > 0$ et $\mathrm{e}^{-\frac{x^2}{2a}} > 0$. Ainsi, $f(x) = \frac{x}{a}\,\mathrm{e}^{-\frac{x^2}{2a}} \geqslant 0$. Donc : $f(x) \geqslant 0$.
  - Si $x < 0$: $f(x) = 0 \geqslant 0$.

D'où : $\forall x \in \mathbb{R},\ f(x) \geqslant 0$.

- La fonction f est continue sur $]-\infty, 0[$ car elle est constante (nulle) sur cet intervalle.
  La fonction f est continue sur $]0, +\infty[$ comme produit de fonctions continues sur cet intervalle.

Ainsi f est continue sur $\mathbb{R}$ sauf éventuellement en 0.

Commentaire

La continuité sur $\mathbb{R}$ sauf en un nombre fini de points suffit ici.
Mais on peut remarquer que f est continue en 0 puisque :

$$\lim_{x\to 0^-} f(x) = \lim_{x\to 0} 0 = 0, \qquad f(0) = 0, \qquad \lim_{x\to 0^+} f(x) = \lim_{x\to 0} \frac{x}{a}\,\mathrm{e}^{-\frac{x^2}{2a}} = 0$$

- Montrons que $\displaystyle\int_{-\infty}^{+\infty} f(t)\,dt$ converge et vaut 1.
  - Tout d'abord : $\displaystyle\int_{-\infty}^{+\infty} f(t)\,dt = \int_{0}^{+\infty} f(t)\,dt$, car f est nulle en dehors de $[0, +\infty[$.
  - La fonction f est de classe $\mathcal{C}^0$ sur $[0, +\infty[$.
    Soit $A \in [0, +\infty[$.

$$\begin{aligned}
\int_0^A f(t)\,dt &= \int_0^A \frac{t}{a}\,\mathrm{e}^{-\frac{t^2}{2a}}\,dt \\
&= -\int_0^A \frac{-t}{a}\,\mathrm{e}^{-\frac{t^2}{2a}}\,dt \\
&= -\left[\,\mathrm{e}^{-\frac{t^2}{2a}}\,\right]_0^A \\
&= -\left(\mathrm{e}^{-\frac{A^2}{2a}} - \mathrm{e}^0\right) \\
&= 1 - \mathrm{e}^{-\frac{A^2}{2a}} \underset{A\to+\infty}{\longrightarrow} 1
\end{aligned}$$

Ainsi : $\displaystyle\int_{-\infty}^{+\infty} f(t)\,dt$ converge et vaut 1. □

2. Déterminer la fonction de répartition F_X de X.

Démonstration.
Soit $x \in \mathbb{R}$. Deux cas se présentent.

– Si $x \leqslant 0$, alors :

$$F_X(x) \;=\; \mathbb{P}\big([X \leqslant x]\big) \;=\; \int_{-\infty}^{x} f(t)\,dt \;=\; \int_{-\infty}^{x} 0\,dt \;=\; 0$$

car f est nulle sur $]-\infty, 0[$.

– Si $x \geqslant 0$, alors :

$$\begin{aligned} F_X(x) \;&=\; \mathbb{P}\big([X \leqslant x]\big) \;=\; \int_{-\infty}^{x} f(t)\,dt \\ &=\; \int_{0}^{x} f(t)\,dt && \textit{(car } f \textit{ est nulle en dehors de } [0,+\infty[\textit{)} \\ &=\; \int_{0}^{x} \frac{t}{a}\,\mathrm{e}^{-\frac{t^2}{2a}}\,dt && \textit{(par définition de } f \textit{ sur } [0,x]\textit{)} \\ &=\; 1-\mathrm{e}^{-\frac{x^2}{2a}} && \textit{(d'après le calcul de la question précédente)} \end{aligned}$$

Ainsi : $F_X : x \mapsto \begin{cases} 1-\mathrm{e}^{-\frac{x^2}{2a}} & \text{si } x \geqslant 0 \\ 0 & \text{si } x < 0 \end{cases}$ □

3. On considère la variable aléatoire Y définie par : $Y = \dfrac{X^2}{2a}$.

a) Montrer que Y suit la loi exponentielle de paramètre 1.

Démonstration.

- Notons $\varphi : x \mapsto \dfrac{x^2}{2a}$ de sorte que $Y = \varphi(X)$.

$$Y(\Omega) = (\varphi(X))(\Omega) = \varphi\big(X(\Omega)\big) \subset [0,+\infty[$$

En effet, $X(\Omega) \subset \mathbb{R}$ et $\varphi(\mathbb{R}) = [0,+\infty[$ (φ ne prend que des valeurs positives).

- Soit $x \in \mathbb{R}$. Deux cas se présentent.
  – Si $x \leqslant 0$, alors $[Y \leqslant x] = \varnothing$ car $Y(\Omega) \subset [0,+\infty[$. Ainsi :

$$F_Y(x) \;=\; \mathbb{P}\big([Y \leqslant x]\big) \;=\; \mathbb{P}(\varnothing) \;=\; 0$$

– Si $x \geqslant 0$, alors :

$$\begin{aligned}
F_Y(x) &= \mathbb{P}\big([Y \leqslant x]\big) = \mathbb{P}\left(\left[\tfrac{X^2}{2a} \leqslant x\right]\right) \\
&= \mathbb{P}\left(\left[X^2 \leqslant 2a\ x\right]\right) && \textit{(car } a > 0\textit{)} \\
&= \mathbb{P}\left(\left[\sqrt{X^2} \leqslant \sqrt{2a\ x}\right]\right) && \textit{(car la fonction } \sqrt{.} \textit{ est strictement croissante sur } [0, +\infty[\textit{)} \\
&= \mathbb{P}\left(\left[\ |X| \leqslant \sqrt{2a\ x}\right]\right) \\
&= \mathbb{P}\left(\left[-\sqrt{2a\ x} \leqslant X \leqslant \sqrt{2a\ x}\right]\right) \\
&= F_X\big(\sqrt{2a\ x}\big) - F_X\big(-\sqrt{2a\ x}\big) && \textit{(car } X \textit{ est une v.a.r. à densité)} \\
&= F_X\big(\sqrt{2a\ x}\big) = 1 - \exp\left(-\frac{\big(\sqrt{2a\ x}\big)^2}{2a}\right) = 1 - \exp\left(-\frac{\cancel{2a}\ x}{\cancel{2a}}\right)
\end{aligned}$$

- On en conclut que X admet pour fonction de répartition :

$$F_Y : x \mapsto \begin{cases} 1 - \mathrm{e}^{-x} & \text{si } x \geqslant 0 \\ 0 & \text{si } x < 0 \end{cases}$$

On a donc bien : $Y \hookrightarrow \mathcal{E}(1)$. □

b) On rappelle qu'en **Scilab** la commande `grand(1, 1, 'exp', 1/lambda)` simule une variable aléatoire suivant la loi exponentielle de paramètre λ. Écrire un script **Scilab** demandant la valeur de a à l'utilisateur et permettant de simuler la v.a.r. X.

Démonstration.

- Dans cette question, on considère : $X(\Omega) \subset [0, +\infty[$.

Commentaire

Ce premier point amène une remarque sur la notation $X(\Omega)$ lorsque X est une v.a.r.

- Rappelons qu'une v.a.r. X est une application $X : \Omega \to \mathbb{R}$.
  Comme la notation le suggère, $X(\Omega)$ est l'image de Ω par l'application X.
  Ainsi, $X(\Omega)$ n'est rien d'autre que l'ensemble des valeurs prises par la v.a.r. X :

$$\begin{aligned} X(\Omega) &= \{X(\omega) \mid \omega \in \Omega\} \\ &= \{x \in \mathbb{R} \mid \exists \omega \in \Omega,\ X(\omega) = x\} \end{aligned}$$

- Il faut bien noter que, dans la définition de $X(\Omega)$, aucune application probabilité $\mathbb{P}$ n'apparaît. Il y a donc une différence fondamentale entre les valeurs que peut prendre X et les valeurs de F_X ou f_X dont la définition dépend d'une probabilité $\mathbb{P}$. Par exemple, si l'on sait que f_X est nulle en dehors de $]0, +\infty[$ alors : :

$$\mathbb{P}\big([X \leqslant 0]\big) = \int_{-\infty}^{0} f(t)\ dt = \int_{-\infty}^{0} 0\ dt = 0$$

  Cela ne signifie pas que X ne prend pas de valeurs négatives mais simplement que cela se produit avec probabilité nulle.

- En toute rigueur, on ne peut donc pas confondre $X(\Omega)$ et l'ensemble sur lequel f_X ne s'annule pas (cela n'a pas beaucoup de sens puisque f_X est définie à un nombre fini de points près). Il est donc fréquent que les énoncés précisent, lors de l'introduction de la v.a.r. X, son ensemble image :

  On considère un variable aléatoire X, **à valeurs positives**, *de densité f*

  C'est ce qu'on se permet de faire dans cette question.

- Par définition : $Y = \dfrac{X^2}{2a}$. Ainsi : $X^2 = 2a\ Y$ et $\sqrt{X^2} = \sqrt{2a\ Y}$.
  Enfin, comme on a supposé que X ne prend que des valeurs positives, on obtient :

$$X = \sqrt{2a\ Y}$$

- D'après la question précédente : $Y \hookrightarrow \mathcal{E}(1)$. On en déduit le script suivant :

```
a = input("Prière d''entrer une valeur strictement positive")
y = grand(1, 1, 'exp', 1)
x = sqrt(2*a*y)
disp(x)
```

□

ESSEC − I − 2018

On définit deux variables aléatoires Y_n et Z_n de la façon suivante.
Pour tout $\omega \in \Omega$:

- $Y_n(\omega) = \max(X_1(\omega), \ldots, X_n(\omega))$ est le plus grand des réels $X_1(\omega)$, …, $X_n(\omega)$;
  on remarque que Y_n est définie également lorsque n vaut 1, de sorte que dans la suite du sujet on pourra considérer Y_{n-1}.
- $Z_n(\omega)$ est le « deuxième plus grand » des nombres $X_1(\omega)$, …, $X_n(\omega)$, autrement dit, une fois que ces n réels sont ordonnés dans l'ordre croissant, Z_n est l'avant-dernière valeur. On note que lorsque la plus grande valeur est présente plusieurs fois, $Z_n(\omega)$ et $Y_n(\omega)$ sont égaux.

1. On suppose que l'on a défini une fonction **Scilab** d'entête `function x = simulX(n)` qui retourne une simulation d'un échantillon de taille n de la loi de X sous la forme d'un vecteur de longueur n. Compléter la fonction qui suit pour qu'elle retourne le couple $(Y_n(\omega), Z_n(\omega))$ associé à l'échantillon simulé par l'instruction `X = simulX(n)` :

```
1  function [y, z] = DeuxPlusGrands(n)
2    X = simulX(n)
3    if ...
4      y = X(1) ; z = X(2)
5    else
6      ...
7    end
8    for k = 3:n
9      if X(k) > y
10       z = ... ; y = ...
11     else
12       if ...
13         z = ...
14       end
15     end
16   end
17 endfunction
```

Démonstration.

```
3    if X(1) > X(2)
4      y = X(1) ; z = X(2)
5    else
6      y = X(2) ; z = X(1)
7    end
8    for k = 3:n
9      if X(k) > y
10       z = y ; y = X(k)
11     else
12       if X(k) > z
13         z = X(k)
14       end
15     end
16   end
```

Détaillons l'obtention de ce programme.

- Comme précisé par l'énoncé, `X = simulX(n)` est un vecteur de longueur n, contenant n réalisations de la v.a.r. X : $X_1(\omega)$, …, $X_n(\omega)$.
- La variable `y` doit contenir le plus grand élément parmi $X_1(\omega)$, …, $X_n(\omega)$. Elle sera alors la réalisation $Y_n(\omega)$.
- De même, la variable `z` doit contenir le second plus grand élément parmi $X_1(\omega)$, …, $X_n(\omega)$. Elle sera alors la réalisation $Z_n(\omega)$.
- L'idée derrière ce script est de parcourir le vecteur `X` et de mettre à jour les variables `y` et `z` au fur et à mesure.
  - × On commence donc par comparer `X(1)` et `X(2)`
    La plus grande valeur est alors stockée dans `y` et la seconde dans `z`. Autrement dit :
    - si `X(1)` $>$ `X(2)`, alors `y` $=$ `X(1)` et `z` $=$ `X(2)`

```
3    if X(1) > X(2)
4      y = X(1) ; z = X(2)
```

    - si `X(1)` $\leqslant$ `X(2)`, alors `y` $=$ `X(2)` et `z` $=$ `X(1)`

```
5    else
6      y = X(2) ; z = X(1)
```

  - × On compare ensuite chaque nouvel élément `X(k)` du vecteur `X` à `y`.
    - Si `X(k)` $>$ `y`, alors :
      - `X(k)` est le maximum de `X(1)`, …, `X(k)`,
      - `y` est donc le deuxième plus grand élément parmi `X(1)`, …, `X(k)` (puisque c'était le maximum de `X(1)`, …, `X(k-1)`)
      - la variable `z` prend donc la valeur de `y`, et la variable `y` celle de `X(k)`.

    On obtient donc :

```
9        if X(k) > y
10         z = y ; y = X(k)
```

    (La mise à jour de la variable `z` *avant celle de la variable* `y` *a permis de ne pas écraser le contenu précédent de* `y`*)*
    - Si `X(k)` $\leqslant$ `y`, alors :
      - `y` est toujours le maximum de `X(1)`, …, `X(k)`.
        Il est donc inutile de mettre cette variable à jour.
      - on compare alors `X(k)` et `z`.
        Si `X(k)` $>$ `z`, alors `X(k)` est le deuxième plus grand élément parmi `X(1)`, …, `X(k)`.
        On met donc à jour la variable `z` :

```
11       else
12         if X(k) > z
13           z = X(k)
```

        Sinon, `z` reste la deuxième plus grande valeur de `X(1)`, …, `X(k)`.
        On ne la met donc pas à jour.
  - × En réitérant ce procédé pour toutes les coordonnées de `X`, on obtient que `(y,z)` est bien une réalisation de (Y_n, Z_n).

Commentaire

- Afin de permettre une bonne compréhension des mécanismes en jeu, on a détaillé la réponse à cette question. Cependant, écrire correctement la fonction **Scilab** démontre la bonne compréhension et permet certainement d'obtenir tous les points alloués.
- On pourrait avoir envie d'écrire :

```
9       if X(k) > y
10        y = X(k) ; z = y
```

Mais attention : en écrivant cela, on effectue les calculs suivants :

$$\texttt{y} = \texttt{X(k)} \quad \longleftrightarrow \quad \texttt{y} \text{ contient } \texttt{X(k)}$$

$$\texttt{z} = \texttt{y} \quad \longleftrightarrow \quad \begin{array}{l}\texttt{z} \text{ contient } \texttt{y} \text{ donc } \texttt{X(k)} \\ \textit{(et non la valeur précédente de } \texttt{y}\textit{)}\end{array}$$

D'où la mise à jour de `z` avant celle de `y`.

□

`ECRICOME – 2019`

- Soit D une variable aléatoire prenant les valeurs -1 et 1 avec équiprobabilité.
- Soit U une variable aléatoire suivant la loi uniforme sur $]0,1[$ et V la variable aléatoire définie par : $V = \dfrac{1}{\sqrt{1-U}}$.

a) Écrire une fonction en langage **Scilab**, d'en-tête `function a = D(n)`, qui prend un entier $n \geqslant 1$ en entrée, et renvoie une matrice ligne contenant n réalisations de la v.a.r. D.

Démonstration.

```
1  function a = D(n)
2    a = zeros(1,n)
3    for i = 1:n
4      r = rand()
5      if r < 1/2 then
6        a(i) = -1
7      else
8        a(i) = 1
9      end
10   end
11 endfuntion
```

- **Début de la fonction**
  On commence par initialiser la variable `a` qui doit contenir, d'après l'énoncé, une matrice ligne à `n` colonnes.

```
2     a = zeros(1,n)
```

- **Structure itérative**
  On met ensuite en place une structure itérative (boucle for) pour affecter à chaque coefficient de la matrice `a` une réalisation de la v.a.r. D.

```
3     for i = 1:n
```

On cherche maintenant à simuler la v.a.r. D.

× D'après l'énoncé : $D \hookrightarrow \mathcal{U}(\{-1,1\})$.
Ainsi, chaque coefficient de la variable `a` doit :

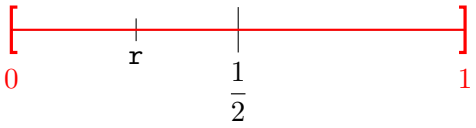
- prendre la valeur -1 avec probabilité $\mathbb{P}([D=-1]) = \dfrac{1}{2}$.
- prendre la valeur 1 avec probabilité $\mathbb{P}([D=1]) = \dfrac{1}{2}$.

× Pour cela, on utilise la commande suivante :

```
5    r = rand()
```

L'instruction `rand()` renvoie un réel choisi aléatoirement dans $]0,1[$.
Plus formellement, il s'agit de simuler une v.a.r. U telle que $U \hookrightarrow \mathcal{U}([0,1])$.

× Cette valeur `r` choisie aléatoirement dans $]0,1[$ permet d'obtenir une simulation de D.



Deux cas se présentent :

- Si `r` $< \dfrac{1}{2}$: alors on affecte à `a(i)` (la $i^{\text{ème}}$ coordonnée de `a`) la valeur -1.
Ce cas se produit avec la probabilité attendue :

$$\mathbb{P}\left(\left[0 < U < \frac{1}{2}\right]\right) = \mathbb{P}\left(\left[U < \frac{1}{2}\right]\right) = \frac{1}{2} = \mathbb{P}([D=-1])$$

- Si `r` $\geqslant \dfrac{1}{2}$: alors on affecte à `a(i)` la valeur 1.
Ce cas se produit avec la probabilité attendue :

$$\mathbb{P}\left(\left[\frac{1}{2} < U < 1\right]\right) = \mathbb{P}\left(\left[\frac{1}{2} < U\right]\right) = \frac{1}{2} = \mathbb{P}([D=1])$$

On obtient la suite du programme :

```
6    if r < 1/2 then
7      a(i) = -1
8    else
9      a(i) = 1
10   end
```

Commentaire

Afin de permettre une bonne compréhension des mécanismes en jeu, on a détaillé la réponse à cette question. Cependant, fournir la fonction **Scilab** démontre la bonne compréhension de la simulation demandée et permet certainement d'obtenir la totalité des points alloués à cette question. On procèdera de même dans la question suivante. □

b) On considère le script suivant :

```
1  n = input('entrer n')
2  a = D(n)
3  b = rand(1,n)
4  c = a / sqrt(1-b)
5  disp( sum(c) / n)
```

De quelle variable aléatoire les coefficients du vecteur `c` sont-ils une simulation ?
Pour `n` assez grand, quelle sera la valeur affichée ? Justifier votre réponse.

Démonstration.

- On commence par demander à l'utilisateur d'entrer une valeur pour l'entier `n`.

```
1  n = input('entrer n')
```

- D'après la question précédente, on affecte ensuite à la variable `a` une matrice ligne contenant `n` réalisations de la v.a.r. D. Plus précisément, la variable `a` est un `n`-uplet $(d_1, \dots, d_{\mathtt{n}})$ qui correspond à l'observation d'un `n`-échantillon $(D_1, \dots, D_{\mathtt{n}})$ de la v.a.r. D.
(cela signifie que les v.a.r. D_1, ..., $D_{\mathtt{n}}$ sont indépendantes et de même loi que D)

```
2  a = D(n)
```

- On continue en affectant à la variable `b` une matrice ligne contenant `n` réalisations d'une loi uniforme sur $]0, 1[$. Autrement dit, la variable `b` est un `n`-uplet $(u_1, \dots, u_{\mathtt{n}})$ qui correspond à l'observation d'un `n`-échantillon $(U_1, \dots, U_{\mathtt{n}})$ de la v.a.r. U.

```
3  b = rand(1,n)
```

- La ligne 4 permet de définir une nouvelle variable `c` :

```
4  c = a ./ sqrt(1-b)
```

  × On sait déjà que la variable `a` contient une observation du `n`-échantillon $(D_1, \dots, D_{\mathtt{n}})$.

  × On rappelle de plus que la variable `b` contient une observation du `n`-échantillon $(U_1, \dots, U_{\mathtt{n}})$. Ainsi, la variable `1 ./ sqrt(1-b)` contient une observation du `n`-échantillon $\left(\frac{1}{\sqrt{1-U_1}}, \dots, \frac{1}{\sqrt{1-U_{\mathtt{n}}}}\right)$.
  D'après ce qui précède, cela correspond à un n-échantillon $(V_1, \dots, V_{\mathtt{n}})$ de la v.a.r. V.
  Or, d'après la question ***7.b)***, les v.a.r. V et Y ont même loi.
  Finalement, on construit ainsi par étape un `n`-échantillon $(Y_1, \dots, Y_{\mathtt{n}})$ de Y.
  La variable `1 ./ sqrt(1-b)` contient une observation de ce `n`-échantillon.

Ainsi, la variable `c` contient une observation du `n`-échantillon $(D_1 \times Y_1, \dots, D_{\mathtt{n}} \times Y_{\mathtt{n}})$, c'est-à-dire une observation $(t_1, \dots, t_{\mathtt{n}})$ du `n`-échantillon $(T_1, \dots, T_{\mathtt{n}})$.

Finalement, la variable `c` contient donc l'observation d'un `n`-échantillon de la v.a.r. T.

Commentaire

L'énoncé original proposait la ligne 4 suivante :

```
4  c = a / sqrt(1-b)
```

Cette commande ne permettait pas d'aboutir au résultat voulu. En effet, la commande :

- `/` correspond à l'opérateur de division à droite. Autrement dit, la commande `A / B` renvoie la matrice `X` (si elle existe) telle que : `X` × `B` = `A`.
- `./` correspond à la division terme à terme. Autrement dit, la commande `A ./ B` renvoie la matrice composée des termes $\frac{a_{i,j}}{b_{i,j}}$. C'est bien ce qu'on voulait faire ici : diviser la 1ère coordonnée de la matrice `a` par la 1ère coordonnée de la matrice `sqrt(1-b)`, diviser la 2ème coordonnée de la matrice `a` par la 2ème coordonnée de la matrice `sqrt(1-b)` ....

- Enfin, la ligne 5 :

```
5  disp(sum(c)/n)
```

permet d'afficher la valeur $\frac{1}{\mathtt{n}}\sum_{i=1}^{\mathtt{n}} t_i$ qui correspond à une observation de la v.a.r. $\frac{1}{\mathtt{n}}\sum_{i=1}^{\mathtt{n}} T_i$, qui n'est autre que la moyenne empirique des variables aléatoires $T_1, \ldots, T_{\mathtt{n}}$).

Or, par loi faible des grands nombres (LfGN) :

$$\text{moyenne de l'observation } = \frac{1}{\mathtt{n}}\sum_{i=1}^{\mathtt{n}} t_i \simeq \mathbb{E}(T)$$

Ainsi, si `n` est assez grand, le programme renvoie une valeur approchée de $\mathbb{E}(T) = 0$.

Commentaire

- Le programme proposé par l'énoncé n'est ici rien d'autre qu'une illustration de l'idée naturelle pour obtenir une approximation de $\mathbb{E}(T)$:
  - × simuler un grand nombre de fois ($\mathtt{n} = 10000$ par exemple) la v.a.r. T.
    Formellement, on souhaite obtenir un $\mathtt{n}$-uplet $(t_1, \dots, t_{\mathtt{n}})$ qui correspond à l'observation d'un $\mathtt{n}$-échantillon $(T_1, \dots, T_{\mathtt{n}})$ de la v.a.r. T.
  - × réaliser la moyenne des résultats de cette observation.
- La réponse fournie à cette question passe à côté d'un détail : dans le programme d'ECE, l'énoncé de la LfGN comporte trois hypothèses. La suite de v.a.r. (T_n) doit être constituée de v.a.r. **indépendantes**, **de même espérance**, **de même variance**.
  Or, on pourrait démontrer que la v.a.r. n'admet pas de variance ! Il semble donc à première vue qu'on ne puisse pas appliquer la LfGN.
  Il existe en fait un énoncé de la LfGN (hors programme) se passant de l'hypothèse d'existence d'une variance. Ainsi, la réponse à cette question est toujours parfaitement correcte.
- Démontrons enfin que la v.a.r. T n'admet pas de variance. Pour cela, on raisonne par l'absurde. Supposons alors que la v.a.r. T admet une variance.
  - × Par formule de Koenig-Huygens : $\mathbb{V}(T) \;=\; \mathbb{E}(T^2) - \big(\mathbb{E}(T)\big)^2 \;=\; \mathbb{E}(T^2)$.
    En effet, d'après la question ***6.b)*** : $\mathbb{E}(T) = 0$.
  - × Or :
    $$\begin{aligned} \mathbb{E}\left(T^2\right) \;&=\; \mathbb{E}\left((DY)^2\right) \;=\; \mathbb{E}\left(D^2\, Y^2\right) \\ &=\; \mathbb{E}\left(D^2\right)\, \mathbb{E}\left(Y^2\right) \qquad \textit{(car les v.a.r. } D \textit{ et } Y \textit{ sont indépendantes)} \end{aligned}$$
  - × De plus, par théorème de transfert :
    $$\mathbb{E}(D^2) \;=\; (-1)^2 \times \mathbb{P}([D=-1]) + 1^2 \times \mathbb{P}([D=1]) \;=\; \frac{1}{2} + \frac{1}{2} \;=\; 1$$
    Ainsi : $\mathbb{V}(T) \;=\; \mathbb{E}\left(T^2\right) \;=\; \mathbb{E}\left(Y^2\right)$.
  - × Par ailleurs, la v.a.r. Y admet un moment d'ordre 2 si et seulement si l'intégrale $\displaystyle\int_{-\infty}^{+\infty} t^2\, f_Y(t)\, dt$ est absolument convergente, ce qui équivaut à démontrer la convergence pour ce calcul de moment du type $\displaystyle\int_{-\infty}^{+\infty} t^m f_Y(t)\, dt$.
    Comme la fonction f_Y est nulle en dehors de $[1, +\infty[$:
    $$\int_{-\infty}^{+\infty} t^2\, f_Y(t)\, dt \;=\; \int_{1}^{+\infty} t^2\, f_Y(t)\, dt$$
    Enfin, soit $t \in [1, +\infty[$: $t^2\, f_Y(t) \;=\; t^2\, \dfrac{2}{t^3} \;=\; \dfrac{2}{t}$.
    Or, l'intégrale $\displaystyle\int_1^{+\infty} \frac{1}{t}\, dt$ est une intégrale de Riemann, impropre en $+\infty$, d'exposant 1. Elle est donc divergente.
    On en déduit que la v.a.r. Y n'admet pas de moment d'ordre 2.
    On en déduit que la v.a.r. T n'admet pas de variance, ce qui est absurde.

□

I.2. Simulation d'une expérience aléatoire et des v.a.r. associées

EDHEC – 2018

- On dispose de trois pièces : une pièce numérotée 0, pour laquelle la probabilité d'obtenir Pile vaut $\frac{1}{2}$ et celle d'obtenir Face vaut également $\frac{1}{2}$, une pièce numérotée 1, donnant Face à coup sûr et une troisième pièce numérotée 2, donnant Pile à coup sûr.
On choisit l'une de ces pièces au hasard et on la lance indéfiniment.
Pour tout i de $\{0, 1, 2\}$, on note A_i l'événement : « on choisit la pièce numérotée i ».
Pour tout entier naturel k non nul, on note P_k l'événement : « on obtient Pile au lancer numéro k » et on pose $F_k = \overline{P_k}$.
On considère la variable aléatoire X, égale au rang d'apparition du premier Pile et la variable aléatoire Y, égale au rang d'apparition du premier Face. On convient de donner à X la valeur 0 si l'on n'obtient jamais Pile et de donner à Y la valeur 0 si l'on n'obtient jamais Face.

  On rappelle que, pour tout entier naturel m, l'instruction `grand(1, 1, 'uin', 0, m)` renvoie un entier aléatoire compris entre 0 et m (ceci de façon équiprobable).
  On décide de coder Pile par 1 et Face par 0.

  a) Compléter le script **Scilab** suivant pour qu'il permette le calcul et l'affichage de la valeur prise par la variable aléatoire X lors de l'expérience réalisée dans cet exercice.

```
1  piece = grand(1, 1, 'uin', ---, ---)
2  x = 1
3  if piece == 0 then
4      lancer == grand(1, 1, 'uin', ---, ---)
5      while lancer == 0
6          lancer = ---
7          x = ---
8      end
9  else
10     if piece == 1 then
11         x = ---
12     end
13 end
14 disp(x)
```

  Démonstration.

  - En ligne 1, la variable `piece` doit contenir un entier aléatoire compris entre 0 et 2 ce qui permet de coder le choix de la pièce qui sera utilisée pour les tirages.

```
1  piece = grand(1, 1, 'uin', 0, 2)
```

  - La structure conditionnelle qui suit (l'utilisation du `if`) permet de coder l'expérience pour chacune des pièces :
    - si la pièce 0 a été initialement choisie, on effectue un premier lancer qui doit donner Pile (représenté par 1) avec probabilité $\frac{1}{2}$ et Face (représenté par 0) avec probabilité $\frac{1}{2}$.

```
4      lancer == grand(1, 1, 'uin', 0, 1)
```

      On doit alors réitérer cet expérience tant que l'on n'obtient pas Pile, c'est à dire tant que l'on obtient Face, ce qui correspond à la condition :

```
5       while lancer == 0
```

À chaque tour de boucle (on y rentre tant qu'on n'obtient pas Pile), on effectue un nouveau lancer avec cette pièce :

```
6           lancer = grand(1, 1, 'uin', 0, 1)
```

On met alors à jour le compteur `x`, en l'incrémentant de 1 à chaque Face obtenu.

```
7           x = x + 1
```

Ce compteur a été initialisé à 1 de sorte qu'en sortie de boucle il contient ce nombre 1 incrémenté de 1 à chaque Face. Ainsi, `x` contient bien le rang d'apparition du premier Pile.

– si la pièce 1 a été initialement choisie, alors à chaque tirage on obtient Face. Dans ce cas, la v.a.r. X prend la valeur 0. On met à jour la variable `x` en conséquence.

```
10      if piece == 1 then
11          x = 0
12      end
```

– le dernier cas (choix de la pièce 2) n'apparaît pas explicitement dans cette structure conditionnelle (*cf* question suivante).

Commentaire

Afin de permettre une bonne compréhension des mécanismes en jeu, on a détaillé la réponse à cette question. Cependant, compléter correctement le programme **Scilab** démontre la bonne compréhension de la simulation demandée et permet certainement d'obtenir tous les points alloués à cette question.
On procédera de même dans les autres questions **Scilab**.

□

b) Justifier que le cas où l'on joue avec la pièce numérotée 2 ne soit pas pris en compte dans le script précédent.

Démonstration.
Comme signalé dans la question précédente, le choix de la pièce 2 n'apparaît pas explicitement dans la structure conditionnelle. Cette pièce renvoie Pile à chaque lancer. Ainsi, si cette pièce est choisie, X prend la valeur 1. La variable `x` qui est initialisée à 1 ne nécessite donc pas de mise à jour.

Le cas de la pièce numérotée 2 n'apparaît pas explicitement dans le script mais est bien géré par ce programme.

□

`ECRICOME – 2015`

- On effectue des tirages **sans remise** dans l'urne U_1, jusqu'à l'obtention de la boule noire.
  On note X la variable aléatoire qui prend pour valeur le nombre de tirages nécessaires pour l'obtention de la boule noire.
  On notera pour tout entier naturel i non nul :
  × N_i l'événement « on tire une boule noire lors du i-ième tirage ».
  × B_i l'événement « on tire une boule blanche lors du i-ième tirage ».

a) On simule 10000 fois cette expérience aléatoire.
Recopier et compléter le programme **Scilab** suivant pour qu'il affiche l'histogramme donnant la fréquence d'apparition du rang d'obtention de la boule noire :

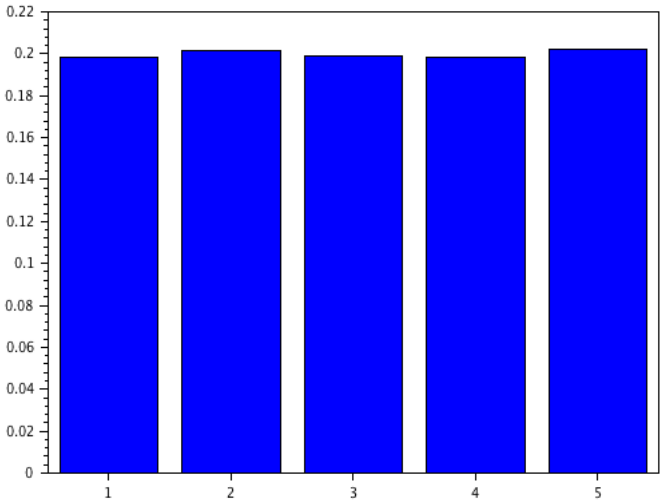
```
N = input('Donner un entier naturel non nul');
S = zeros(1,N);
for k = 1 : 10000
    i = 1;
    M = N;
    while ---
        i = i + 1;
        M = ---;
    end
    S(i) = S(i) + 1;
    end
disp(S / 10000)
bar(S / 10000)
```

Démonstration.

```
N = input('Donner un entier naturel non nul');
S = zeros(1,N);
for k = 1 : 10000
    i = 1;
    M = N;
    while grand(1, 1, 'uin', 1, M) <> M
        i = i + 1;
        M = M-1;
    end
    S(i) = S(i) + 1;
    end
disp(S / 10000)
bar(S / 10000)
```

□

b) On exécute le programme complété ci-dessus.
On entre 5 au clavier et on obtient l'histogramme suivant :



Quelle conjecture pouvez-vous émettre sur la loi de la variable aléatoire X ?

Démonstration.
On peut conjecturer que la v.a.r. X suit une loi uniforme $\mathcal{U}(\llbracket 1,5 \rrbracket)$. □

ECRICOME – 2016

- Dans tout l'exercice, X et Y sont deux variables aléatoires définies sur le même espace probabilisé et à valeurs dans $\mathbb{N}$. On dit que les deux variables X et Y sont **échangeables** si :

$$\forall (i,j) \in \mathbb{N}^2,\ \mathbb{P}([X=i] \cap [Y=j]) = \mathbb{P}([X=j] \cap [Y=i])$$

Résultats préliminaires

a) On suppose que X et Y sont deux variables indépendantes et de même loi.
Montrer que X et Y sont échangeables.

Démonstration.
Soit $(i,j) \in \mathbb{N}^2$.

$$\begin{aligned}
\mathbb{P}([X=i] \cap [Y=j]) &= \mathbb{P}([X=i]) \times \mathbb{P}([Y=j]) && \textit{(car } X \textit{ et } Y \textit{ sont indépendantes)} \\
&= \mathbb{P}([Y=i]) \times \mathbb{P}([Y=j]) && \textit{(car } X \textit{ et } Y \textit{ ont même loi)} \\
&= \mathbb{P}([Y=i]) \times \mathbb{P}([X=j]) && \textit{(car } X \textit{ et } Y \textit{ ont même loi)} \\
&= \mathbb{P}([X=j] \cap [Y=i]) && \textit{(car } X \textit{ et } Y \textit{ sont indépendantes)}
\end{aligned}$$

Ainsi, si X et Y sont indépendantes et de même loi, alors elles sont échangeables. □

b) On suppose que X et Y sont échangeables. Montrer, à l'aide de la formule des probabilités totales, que :

$$\forall i \in \mathbb{N},\ \mathbb{P}([X=i]) = \mathbb{P}([Y=i])$$

Démonstration.
Soit $i \in \mathbb{N}$.
La famille $([Y=j])_{j\in\mathbb{N}}$ est un système complet d'événements.
On en déduit, par application de la formule des probabilités totales :

$$\begin{aligned}
\mathbb{P}([X=i]) &= \sum_{j=0}^{+\infty} \mathbb{P}([X=i] \cap [Y=j]) \\
&= \sum_{j=0}^{+\infty} \mathbb{P}([X=j] \cap [Y=i]) && \textit{(car } X \textit{ et } Y \textit{ sont échangeables)} \\
&= \mathbb{P}([Y=i])
\end{aligned}$$

La dernière égalité est obtenue en appliquant la formule des probabilités totales avec le système complet d'événements $([X=j])_{j\in\mathbb{N}}$.

On en déduit : $\forall i \in \mathbb{N}$, $\mathbb{P}([X=i]) = \mathbb{P}([Y=i])$.

Commentaire

On vient de montrer l'implication suivante :

$$X \text{ et } Y \text{ sont échangeables } \Rightarrow \ X \text{ et } Y \text{ ont même loi}$$

□

Étude d'un exemple

Soient n, b et c trois entiers strictement positifs.
Une urne contient initialement n boules noires et b boules blanches. On effectue l'expérience suivante, en distinguant trois variantes.

- On pioche une boule dans l'urne.
  On définit X la variable aléatoire qui vaut 1 si cette boule est noire et 2 si elle est blanche.
- On replace la boule dans l'urne et :
  - × Variante 1 : on ajoute dans l'urne c boules de la même couleur que la boule qui vient d'être piochée.
  - × Variante 2 : on ajoute dans l'urne c boules de la couleur opposée à celle de la boule qui vient d'être piochée.
  - × Variante 3 : on n'ajoute pas de boule supplémentaire dans l'urne.
- On pioche à nouveau une boule dans l'urne.
  On définit Y la variable aléatoire qui vaut 1 si cette seconde boule piochée est noire et 2 si elle est blanche.

c) (i) Compléter la fonction **Scilab**suivante, qui simule le tirage d'une boule dans une urne contenant b boules blanches et n boules noires et qui retourne 1 si la boule tirée est noire, et 2 si la boule tirée est blanche.

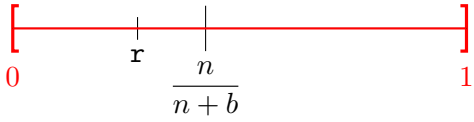
```
1  function res = tirage(b,n)
2      r = rand()
3      if ..... then
4          res = 2
5      else
6          res = 1
7      end
8  endfunction
```

Démonstration.

- D'après l'énoncé, la fonction `tirage` a pour but de simuler la v.a.r. X.
  Ainsi, le paramètre de sortie `res` de cette fonction doit :
  - × prendre la valeur 1 avec probabilité $\mathbb{P}([X=1]) = \dfrac{n}{n+b}$.
  - × prendre la valeur 2 avec probabilité $\mathbb{P}([X=2]) = \dfrac{b}{n+b}$.
- La fonction débute par la ligne 2 :

```
2      r = rand()
```

  L'instruction `rand()` renvoie un réel choisi aléatoirement dans $[0,1]$.
  Plus formellement, il s'agit de simuler une v.a.r. U telle que $U \hookrightarrow \mathcal{U}([0,1])$.
- Cette valeur `r` choisie aléatoirement dans $[0,1]$ permet d'obtenir la valeur `res`.



Deux cas se présentent.

– Si `r` $\leqslant \dfrac{n}{n+b}$: alors, on affecte à la variable `res` la valeur 1.
Ce cas se produit avec la probabilité attendue :

$$\mathbb{P}\left(\left[0 \leqslant U \leqslant \frac{n}{n+b}\right]\right) = \mathbb{P}\left(\left[U \leqslant \frac{n}{n+b}\right]\right) = \frac{n}{n+b} = \mathbb{P}([X=1])$$

– Si `r` $> \dfrac{n}{n+b}$: alors, on affecte à la variable `res` la valeur 2.
Ce cas se produit avec la probabilité attendue :

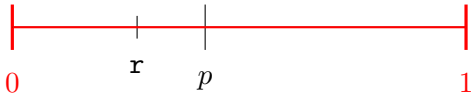
$$\mathbb{P}\left(\left[\frac{n}{n+b} < U \leqslant 1\right]\right) = \mathbb{P}\left(\left[\frac{n}{n+b} < U\right]\right) = 1 - \mathbb{P}\left(\left[U \leqslant \frac{n}{n+b}\right]\right) = \frac{b}{n+b} = \mathbb{P}([X=2])$$

- On en déduit la ligne 3 à compléter :

```
3     if r > n / (n + b) then
```

Commentaire

- L'idée développée ici est utilisée lorsque l'on souhaite simuler une v.a.r. X telle que $X \hookrightarrow \mathcal{B}(p)$ avec $p \in]0,1[$ à l'aide de la fonction `rand`.
  On choisit aléatoirement un réel `r` dans $[0,1]$:



  On obtient une valeur plus petite que p avec probabilité : $\mathbb{P}([U \leqslant p]) = p$.
  On obtient une valeur strictement plus grande que p avec probabilité :

$$q = \mathbb{P}([U > p]) = 1 - \mathbb{P}([U \leqslant p]) = 1 - p$$

  D'où le programme suivant :

```
1  function res = bernoulli(p)
2    r = rand()
3    if r < p then
4      res = 1
5    else
6      res = 0
7    end
8  endfunction
```

- Plus généralement, cette méthode permet d'obtenir une simulation de n'importe quelle v.a.r. X finie. Détaillons ce résultat. Soit X une v.a.r. telle que :
  - $X(\Omega) = \{x_1, \ldots, x_n\}$,
  - $\forall i \in [\![1,n]\!]$, $\mathbb{P}([X = x_i]) = p_i$.

  L'idée est alors de découper le segment $[0,1]$ en n intervalles $I_1, \ldots, I_n$.
  La taille du premier intervalle est p_1, celle du deuxième est p_2 et ainsi de suite.
  De sorte que, pour tout $i \in [\![1,n]\!]$:

$$\mathbb{P}([U \in I_i]) = p_i = \mathbb{P}([X = x_i])$$

  Il n'y a plus qu'à écrire le programme correspondant.
- Afin de permettre une bonne compréhension des mécanismes en jeu, on a détaillé la réponse à cette question. Cependant, compléter correctement le programme **Scilab** démontre la bonne compréhension de la simulation demandée et permet certainement d'obtenir tous les points alloués à cette question.
  On procédera de même dans les autres questions **Scilab**.

□

(ii) Compléter la fonction suivante, qui effectue l'expérience étudiée avec une urne contenant initialement b boules blanches, n boules noires et qui ajoute éventuellement c boules après le premier tirage, selon le choix de la variante dont le numéro est `variante`.
Les paramètres de sortie sont :

- `x` : une simulation de la variable aléatoire X
- `y` : une simulation de la variable aléatoire Y

```
1  function [x,y] = experience(b,n,c,variante)
2      x = tirage(b,n)
3      if variante == 1 then
4          if x == 1 then
5              ..........
6          else
7              ..........
8          end
9      elseif variante == 2 then
10         ..........
11         ..........
12         ..........
13         ..........
14         ..........
15     end
16     y = tirage(b,n)
17 endfunction
```

Démonstration.

- Comme on l'a vu dans la question précédente, l'instruction `tirage(b, n)` permet de simuler la v.a.r. X. C'est ce que réalise l'instruction en ligne 2 du programme :

```
2      x = tirage (b, n)
```

- Il reste alors à simuler la v.a.r. Y.
  Le paramètre de sortie `y` doit :
  - × prendre la valeur 1 avec probabilité $\mathbb{P}([Y=1]) = \dfrac{m}{m+d}$.
  - × prendre la valeur 2 avec probabilité $\mathbb{P}([Y=2]) = \dfrac{d}{m+d}$.

  où m (resp. d) représente le nombre de boules noires (resp. blanches) de l'urne après l'ajout éventuel issu du premier tirage.
- Plus précisément m et d sont définies en fonction du résultat du premier tirage et des variantes.
  Variante 1 : deux cas se présentent.
  - × Si `x` vaut 1 : c'est qu'on a tiré une boule noire lors du premier tirage.
    On remet, en plus de cette boule, c boules noires dans l'urne. Ainsi :
    $$m = n + c \quad \text{et} \quad d = b \quad \text{(pas de modification)}$$
  - × Sinon (`x` vaut 2) : c'est qu'on a tiré une boule blanche lors du premier tirage.
    On remet, en plus de cette boule, c boules noires dans l'urne. Ainsi :
    $$m = n \quad \text{(pas de modification)} \quad \text{et} \quad d = b + c$$

  On en déduit les lignes 3 à 8 du programme dans lesquelles on met à jour les variables `n` et `b` en fonction de leur nouvelle valeur.

```
3      if variante == 1 then
4        if x == 1 then
5          n = n + c
6        else
7          b = b + c
8        end
```

Variante 2 : l'étude est similaire mais, comme on remet des boules de couleur opposée à la première boule tirée, les rôles de `n` et `b` sont échangés par rapport à la première variante. On en déduit les lignes 9 à 14 du programme dans lesquelles on met à jour les variables `n` et `b` en fonction de leur nouvelle valeur.

```
9     else if variante == 2 then
10      if x == 1 then
11        b = b + c
12      else
13        n = n + c
14      end
```

Variante 3 : il n'y a pas de modifications des boules blanches ou noires. Il n'y a donc pas lieu de mettre à jour les variables `n` et `b`.

- On obtient alors la valeur `y` en simulant le tirage dans l'urne modifiée (c'est à dire avec les valeurs de `n` et `b` mise à jour). C'est l'objectif de la ligne 16.

```
16    y = tirage (b, n)
```

Commentaire

- L'énoncé comporte une petite coquille. En effet, en ligne 9, on devait lire :

```
9     elseif variante == 2 then
```

  en lieu et place de :

```
9     else if variante == 2 then
```

- La différence est subtile.
  - Dans le programme corrigé, la structure conditionnelle englobante contient 2 branchements dont le $2^{\text{ème}}$ (en ligne 9) est soumis à la condition **variante** == 2.
  - Dans le programme d'origine, la structure conditionnelle englobante contient 2 branchements dont le $2^{\text{ème}}$ (en ligne 9) débouche sur une nouvelle structure conditionnelle if **variante** == 2. Il faut alors fermer cette $2^{\text{ème}}$ structure conditionnelle à l'aide d'un end. Le nombre de lignes allouées n'est donc plus suffisant.
- Notons que le respect du nombre de lignes alloué n'est pas déterminant. C'est plutôt une indication que donne le concepteur sur le nombre de lignes que le programme doit normalement prendre. Mais on peut raisonnablement penser que tout programme juste (même s'il ne respecte pas le nombre de lignes restant) sera accepté.

□

(iii) Compléter la fonction suivante, qui simule l'expérience N fois (avec $N \in \mathbb{N}^*$), et qui estime la loi de X, la loi de Y et la loi du couple (X, Y).
Les paramètres de sortie sont :

- `loiX` : un tableau unidimensionnel à deux éléments qui estime $[\mathbb{P}([X=1]), \mathbb{P}([X=2])]$
- `loiY` : un tableau unidimensionnel à deux éléments qui estime $[\mathbb{P}([Y=1]), \mathbb{P}([Y=2])]$
- `loiXY` : un tableau bidimensionnel à deux lignes et deux colonnes qui estime :

$$\begin{bmatrix} \mathbb{P}([X=1]\cap[Y=1]) & \mathbb{P}([X=1]\cap[Y=2]) \\ \mathbb{P}([X=2]\cap[Y=2]) & \mathbb{P}([X=1]\cap[Y=2]) \end{bmatrix}$$

```
function [loiX,loiY,loiXY] = estimation(b,n,c,variante,N)
    loiX = [0,0]
    loiY = [0,0]
    loiXY = [0,0;0,0]
    for k = 1 : N
        [x,y] = experience(b,n,c,variante)
        loiX(x) = loiX(x) + 1
        ..........
        ..........
    end
    loiX = loiX / N
    loiY = loiY / N
    loiXY = loiXY / N
endfunction
```

Démonstration.

Commentaire

L'énoncé de cette question comporte une erreur.
Bien évidemment, la loi du couple (X,Y) est caractérisé par la matrice :

$$\begin{bmatrix} \mathbb{P}([X=1]\cap[Y=1]) & \mathbb{P}([X=1]\cap[Y=2]) \\ \mathbb{P}([X=2]\cap[Y=1]) & \mathbb{P}([X=2]\cap[Y=2]) \end{bmatrix}$$

- Dans la question précédente, on a simulé les v.a.r. X et Y.
  Il s'agit maintenant d'obtenir une approximation des lois de ces v.a.r.
- Rappelons tout d'abord que X ne prend que deux valeurs : $X(\Omega)=\{1,2\}$.
  Ainsi, la loi de X est entièrement déterminée par les valeurs :

$$\mathbb{P}([X=1]) \quad \text{et} \quad \mathbb{P}([X=2])$$

  L'idée naturelle pour obtenir un approximation de ces valeurs est :
  - × de simuler un grand nombre de fois (N est ce grand nombre) la v.a.r. X.
    Formellement, on souhaite obtenir un N-uplet $(x_1,\dots,x_N)$ qui correspond à l'observation d'un N-échantillon $(X_1,\dots,X_N)$ de la v.a.r. X.
  - × de compter le nombre de 1 (resp. de 2) de cette observation.

  Cette idée est justifiée par la loi faible des grands nombres (LfGN) qui affirme :

$$\frac{\text{nombre de 1 de l'observation}}{\text{taille de l'observation}} \simeq \mathbb{P}([X=1])$$

- Dans le programme, les valeurs $(x_1,\dots,x_N)$ sont obtenues par des appels successifs (à l'aide d'une boucle `for`) à la fonction `experience` et stockées les unes après les autres dans la variable `x`.

```
for k = 1 : N
    [x , y] = experience(b, n, c, variante)
```

  (seules les valeurs de `x` *nous intéressent dans un premier temps)*
  Le tableau `loiX` est alors mis à jour à chaque tour de boucle :

```
        loiX(x) = loiX(x) + 1
```

Détaillons cette mise à jour :

× si `x` vaut 1 alors l'instruction suivante est effectuée :

```
loiX(1) = loiX(1) + 1
```

× si `x` vaut 2 alors l'instruction suivante est effectuée :

```
loiX(2) = loiX(2) + 1
```

Cela signifie que le 1er élément du tableau compte le nombre de 1 de l'observation et que le 2ème compte le nombre de 2.
Une fois cette boucle effectuée, l'approximation formulée par la LfGN est obtenue en divisant ces nombres par la taille de l'observation :

```
11    loiX = loiX / N
```

- On agit de même pour obtenir l'approximation de la loi de Y.
On génère des observations $(y_1, \dots, y_N)$ d'un N-échantillon $(Y_1, \dots, Y_N)$ de la v.a.r. Y.

```
5     for k = 1 : N
6       [x , y] = experience(b, n, c, variante)
```

(ce sont maintenant les valeurs de `y` *qui nous intéressent)*
Puis on met à jour le tableau `loiY` de sorte à compter le nombre de 1 et de 2 que contient cette observation.

```
8     loiY(y) = loiY(y) + 1
```

L'approximation formulée par la LfGN est obtenue en divisant ces nombres par la taille de l'observation :

```
12    loiY = loiY / N
```

- On agit encore de même pour obtenir l'approximation de la loi de couple. On génère des couples d'observations $((x_1, y_1), \dots, (x_N, y_N))$ d'un N-échantillon $((X_1, Y_1), \dots, (X_N, Y_N))$ du couple (X, Y). C'est toujours les lignes 5 et 6 :

```
5     for k = 1 : N
6       [x , y] = experience(b, n, c, variante)
```

(ce sont à présent les couples de valeurs qui nous intéressent)
Puis on met à jour le tableau `loiXY` de sorte à compter le nombre de $(1,1)$, de $(1,2)$, de $(2,1)$, de $(2,2)$:

```
9     loiXY(x, y) = loiXY(x, y) + 1
```

L'approximation formulée par la LfGN est obtenue en divisant ces nombres par la taille de l'observation :

```
13    loiXY = loiXY / N
```

□

(iv) On exécute notre fonction précédente avec $b = 1$, $n = 2$, $c = 1$, $N = 10000$ et dans chacune des variantes. On obtient :

```
-->[loiX,loiY,loiXY] = estimation(1,2,1,1,10000)
    LoiXY =
        0.49837     0.16785
        0.16697     0.16681

    LoiY =
        0.66534     0.33466

    LoiX =
        0.66622     0.33378

-->[loiX,loiY,loiXY] = estimation(1,2,1,2,10000)
    LoiXY =
        0.33258     0.33286
        0.25031     0.08425

    LoiY =
        0.58289     0.41711

    LoiX =
        0.66544     0.33456

-->[loiX,loiY,loiXY] = estimation(1,2,1,3,10000)
    LoiXY =
        0.44466     0.22098
        0.22312     0.11124

    LoiY =
        0.66778     0.33222

    LoiX =
        0.66564     0.33436
```

En étudiant ces résultats, émettre des conjectures quant à l'indépendance et l'échangeabilité de X et Y dans chacune des variantes.
On donne les valeurs numériques approchées suivantes :

$$\begin{aligned}
0.33 \times 0.33 &\simeq 0.11 \\
0.33 \times 0.41 &\simeq 0.14 \\
0.33 \times 0.58 &\simeq 0.19 \\
0.33 \times 0.66 &\simeq 0.22 \\
0.41 \times 0.66 &\simeq 0.27 \\
0.58 \times 0.66 &\simeq 0.38 \\
0.66 \times 0.66 &\simeq 0.44
\end{aligned}$$

Démonstration.

- **Variante 1** :
  - × Indépendance. On lit $\mathbb{P}([X = 1] \cap [Y = 1])$ sur la coordonnée $(1, 1)$ de `loiXY`.

    Donc : $\mathbb{P}([X = 1] \cap [Y = 1]) \simeq 0,49$

    On lit $\mathbb{P}([X = 1])$ sur la $1^{\text{ère}}$ coordonnée de `loiX`. Donc $\mathbb{P}([X = 1]) \simeq 0,66$.
    On lit $\mathbb{P}([Y = 1])$ sur la $1^{\text{ère}}$ coordonnée de `loiX`. Donc $\mathbb{P}([Y = 1]) \simeq 0,66$.

D'après les données de l'énoncé, on obtient donc :

$$\mathbb{P}([X=1])\mathbb{P}([Y=1]) \simeq 0,44 \neq \mathbb{P}([X=1]\cap[Y=1])$$

On conjecture alors que X et Y ne sont pas indépendantes.

× Échangeabilité. On lit $\mathbb{P}([X=1]\cap[Y=2])$ sur la coordonnée $(1,2)$ de `loiXY`, et $\mathbb{P}([X=2]\cap[Y=1])$ sur la coordonnée $(2,1)$ de `loiXY`. Donc :

$$\mathbb{P}([X=1]\cap[Y=2]) \simeq 0,16 \simeq \mathbb{P}([X=1]\cap[Y=2])$$

On conjecture alors que X et Y sont échangeables.

Dans le cas de la variante 1, on conjecture que les v.a.r. X et Y sont échangeables et non indépendantes.

- **Variante 2** :

× Indépendance. Par lecture : $\mathbb{P}([X=1]\cap[Y=1]) \simeq 0,33$

De plus : $\mathbb{P}([X=1]) \simeq 0,66$ et $\mathbb{P}([Y=1]) \simeq 0,58$.

D'après les données de l'énoncé, on obtient donc :

$$\mathbb{P}([X=1])\mathbb{P}([Y=1]) \simeq 0,38 \neq \mathbb{P}([X=1]\cap[Y=1])$$

On conjecture alors que X et Y ne sont pas indépendantes.

× Échangeabilité. Par lecture : $\mathbb{P}([X=1]\cap[Y=2]) \simeq 0,33$ et $\mathbb{P}([X=2]\cap[Y=1]) \simeq 0,25$.

Donc : $\mathbb{P}([X=1]\cap[Y=2]) \neq \mathbb{P}([X=2]\cap[Y=1])$.

On conjecture alors que X et Y ne sont pas échangeables.

Dans le cas de la variante 2, on conjecture que les v.a.r. X et Y sont non échangeables et non indépendantes.

- **Variante 3** :

× Indépendance. Par lecture :

$$\mathbb{P}([X=1])\mathbb{P}([Y=1]) \simeq 0,66\times 0,66 \simeq 0,44 \simeq \mathbb{P}([X=1]\cap[Y=1])$$

$$\mathbb{P}([X=1])\mathbb{P}([Y=2]) \simeq 0,66\times 0,33 \simeq 0,22 \simeq \mathbb{P}([X=1]\cap[Y=2])$$

$$\mathbb{P}([X=2])\mathbb{P}([Y=1]) \simeq 0,33\times 0,66 \simeq 0,22 \simeq \mathbb{P}([X=2]\cap[Y=1])$$

$$\mathbb{P}([X=2])\mathbb{P}([Y=2]) \simeq 0,33\times 0,33 \simeq 0,11 \simeq \mathbb{P}([X=2]\cap[Y=2])$$

On conjecture alors que X et Y sont indépendantes.

× Échangeabilité. Par lecture :

$$\mathbb{P}([X=1]\cap[Y=2]) \simeq 0,22 \simeq \mathbb{P}([X=1]\cap[Y=2])$$

On conjecture alors que X et Y sont échangeables.

Dans le cas de la variante 3, on conjecture que les v.a.r. X et Y sont échangeables et indépendantes.

□

EML – 2017

- On considère une urne contenant initialement une boule bleue et deux boules rouges.
  On effectue, dans cette urne, des tirages successifs de la façon suivante : on pioche une boule au hasard et on note sa couleur, puis on la replace dans l'urne en ajoutant une boule de la même couleur que celle qui vient d'être obtenue.

  Pour tout k de $\mathbb{N}^*$, on note B_k l'événement : « on obtient une boule bleue au $k^{\text{ème}}$ tirage »

  R_k l'événement : « on obtient une boule rouge au $k^{\text{ème}}$ tirage ».

a) Recopier et compléter la fonction suivante afin qu'elle simule l'expérience étudiée et renvoie le nombre de boules rouges obtenues lors des n premiers tirages, l'entier n étant entré en argument.

```
function s = EML(n)
    b = 1 // b désigne le nombre de boules bleues présentes dans l'urne
    r = 2 // r désigne le nombre de boules rouges présentes dans l'urne
    s = 0 // s désigne le nombre de boules rouges obtenues lors des n tirages
    for k = 1:n
        s = rand()
        if ... then
            ...
        else
            ...
        end
    end
endfunction
```

Démonstration.

- Le programme consiste, au fur et à mesure des tirages :
  - × à mettre à jour les variables `b` et `r`, désignant respectivement le nombre de boules bleues et le nombre de boules rouges présentes dans l'urne.
  - × à comptabiliser le nombre `s` de boules rouges tirées.
- L'instruction `rand()` permet de simuler une v.a.r. de loi uniforme sur $]0,1[$.
  Le résultat obtenu (stocké dans la variable `x`) va permettre de simuler le tirage :
  - × si `x` $\in]0, \frac{\texttt{b}}{\texttt{b+r}}[$, on considère qu'on a tiré une boule bleue dans l'urne. Dans ce cas :
    on ajoute une boule bleue dans l'urne : `b = b+1`.
    on ne modifie pas le nombre de boules rouges de l'urne.
    on ne modifie pas le nombre de boules rouges tirées.
    ($\frac{\texttt{b}}{\texttt{b+r}}$ est la proportion de boules bleues dans l'urne)
  - × si `x` $\in [\frac{\texttt{b}}{\texttt{b+r}}, 1[$, on considère qu'on a tiré une boule rouge dans l'urne. Dans ce cas :
    on ajoute une boule rouge dans l'urne : `r = r+1`.
    on ne modifie pas le nombre de boules bleues de l'urne.
    on met à jour le nombre de boules rouges tirées : `s = s+1`.
    ($1 - \frac{\texttt{b}}{\texttt{b+r}} = \frac{\texttt{r}}{\texttt{b+r}}$ est la proportion de boules rouges dans l'urne)

En résumé, on obtient :

```
7   if x < b / (b+r) then
8       b = b + 1
9   else
10      r = r + 1
11      s = s + 1
12  end
```

Commentaire

- Pour être en accord avec le nombre de lignes utilisées dans le programme on pouvait remplacer les lignes 10 et 11 par :

```
10      r = r + 1; s = s + 1
```

- Les variables `r` et `s` évoluent de la même façon : elles sont incrémentées d'une unité à chaque tirage d'une boule rouge. Ainsi, la relation : $\texttt{s} = \texttt{r} - 2$ est vérifiée à chaque tour de boucle (on parle d'invariant de boucle) et donc aussi à la fin du programme.

□

b) On exécute le programme suivant :

```
1  n = 10
2  m = 0
3  for i = 1:1000
4      m = m + EML(n)
5  end
6  disp(m/1000)
```

On obtient 6.657. Comment interpréter ce résultat ?

Démonstration.

- La fonction `EML` permet de simuler la v.a.r. S_n (introduite plus tard dans l'énoncé) égale au nombre de boules rouges obtenues au cours des n premiers tirages.
- Le programme consiste à simuler (à l'aide de l'appel `EML(10)`) de simuler un grand nombre de fois ($N = 1000$ est ce grand nombre) la v.a.r. S_{10}.
  Formellement, on souhaite obtenir un N-uplet $(v_1, \dots, v_N)$ qui correspond à l'observation d'un N-échantillon $(V_1, \dots, V_N)$ de la v.a.r. S_{10}.
  (cela signifie que les v.a.r. V_1, …, V_N sont indépendantes et sont de même loi que S_{10})
- La variable `m`, initialisée à 0, est mise à jour à chaque tour de boucle i par l'ajout de la dernière valeur v_i créée. De sorte que, à l'issue de la boucle, la variable `m` contient : $\sum_{i=1}^{N} v_i$.
- Enfin, en ligne 6, l'instruction `disp(m/1000)` permet de réaliser l'affichage de la division par 1000 de la valeur contenue dans `m`. C'est donc la valeur : $\frac{1}{N}\sum_{i=1}^{N} v_i$ qui est affichée.
  Il s'agit de la moyenne empirique des N simulations de la v.a.r. S_{10}.
- Or, en vertu de la loi faible des grands nombres (LfGN) :

$$\text{moyenne de l'observation} = \frac{1}{N}\sum_{i=1}^{N} v_i \simeq \mathbb{E}(S_{10})$$

Le programme fournit une approximation de $\mathbb{E}(S_{10})$. Le résultat obtenu : $\mathbb{E}(S_{10}) \simeq 6.657$ signifie qu'on obtient en moyenne un peu moins de 7 boules rouges lorsque l'on procède à 10 tirages successifs (en respectant le protocole de l'énoncé) dans l'urne.

Commentaire

Comme $6.657 \simeq \dfrac{2}{3} \times 10$, on peut faire l'hypothèse que : $\mathbb{E}(S_n) = \dfrac{2}{3} \times n$.
Ceci semble être le cas puisque la question ***9.*** demande justement de démontrer cette égalité. □

EDHEC – 2019

- Soit n un entier naturel supérieur ou égal à 3.
  Une urne contient une boule noire non numérotée et $n-1$ boules blanches dont $n-2$ portent le numéro 0 et une porte le numéro 1. On extrait ces boules au hasard, une à une, sans remise, jusqu'à l'apparition de la boule noire.
  Pour chaque i de $[\![1, n-1]\!]$, on note B_i l'événement : « le $i^{\text{ème}}$ tirage donne une boule blanche », on pose $\overline{B_i} = N_i$, et on note X la variable aléatoire égale au rang d'apparition de la boule noire.
- On note Y la variable aléatoire qui vaut 1 si la boule numérotée 1 a été piochée lors de l'expérience précédente et qui vaut 0 sinon.

On rappelle qu'en **Scilab**, la commande `grand(1, 1, 'uin', a, b)` simule une variable aléatoire suivant la loi uniforme $[\![a, b]\!]$.

a) Compléter le script **Scilab** suivant afin qu'il simule l'expérience aléatoire décrite dans cet exercice et affiche la valeur prise par la variable aléatoire X.
On admettra que la boule noire est codée tout au long de ce script par le nombre `nB + 1`, où `nB` désigne le nombre de boules blanches.

```
1  n = input('Entrez une valeur pour n : ')
2  nB = n-1
3  X = 1
4  u = grand(1, 1, 'uin', 1, nB+1)
5  while u < nB + 1
6      nB = ----
7      u = grand(1, 1, 'uin', 1, ----)
8      X = ----
9  end
10 disp(X, 'La boule noire est apparue au tirage numéro')
```

Démonstration.
Détaillons les différents éléments de ce programme.

- En ligne 1, on récupère la valeur de `n`, variable qui contiendra le nombre de boules, à l'aide d'une dialogue avec l'utilisateur.

```
1  n = input('Entrez une valeur pour n : ')
```

- On initialise alors la variable `X` qui contiendra la valeur prise par X pour la succession de tirages simulés et la variable `nB` qui contient le nombre de boules blanches.

```
2  nB = n-1
3  X = 1
```

- On simule ensuite un premier tirage dans l'urne. Pour ce faire, on considère initialement que les boules sont numérotées de 1 à n. Les boules blanches sont numérotées 1 à $n-1$ sont les boules blanches et la boule noire est numérotée n.

```
4  u = grand(1, 1, 'uin', 1, nB+1)
```

Plus précisément, on stocke dans la variable `u` le résultat de la simulation d'une v.a.r. U qui suit la loi $\mathcal{U}(\llbracket 1, \texttt{nB + 1} \rrbracket)$. On obtient ainsi un entier choisi aléatoirement entre 1 et `nB+1`, variable contenant le nombre de boules en tout dans l'urne.

- S'ensuit une structure itérative permettant de simuler la succession de tirages tant que la boule noire n'a pas été tirée.
  - × La ligne 4 permet de réaliser l'itération tant que la valeur de `u` est différente de celle de `nB+1` (on peut le faire grâce à l'inégalité de la ligne 5 car `u` prend sa valeur dans $\llbracket 1, \texttt{nB+1} \rrbracket$). Il faut donc comprendre que la boule noire est numérotée `nB+1` tout au long du programme.

```
5  while u < nB + 1
```

  - × En ligne 5, on met à jour le nombre de boules dans l'urne après avoir procédé au tirage.

```
6      nB = nB - 1
```

  - × Puis on procède à un nouveau tirage.

```
7      u = grand(1, 1, 'uin', 1, nB+1)
```

    Il faut comprendre que l'on renumérote à chaque tirage les boules dans l'urne. Si initialement l'urne contient 5 boules numérotées de 1 à 5, on considère que les boules dont le numéro est dans $\llbracket 1, 4 \rrbracket$ sont blanches et que la boule 5 est noire. Si on tire la boule 2, alors il ne reste plus que 4 boules dans l'urne dont 3 blanches qu'on peut alors renuméroter 1, 2 et 3. La boule noire est alors numérotée 4.
  - × Enfin, on met à jour la variable `X` qui compte le nombre de tirages faits jusqu'à présent.

```
8      X = X + 1
9  end
```

Finalement, on obtient le programme complet suivant.

```
1  n = input('Entrez une valeur pour n : ')
2  nB = n-1
3  X = 1
4  u = grand(1, 1, 'uin', 1, nB+1)
5  while u < nB + 1
6      nB = nB - 1
7      u = grand(1, 1, 'uin', 1, nB+1)
8      X = X + 1
9  end
10 disp(X, 'La boule noire est apparue au tirage numéro')
```

Commentaire

Afin de permettre une bonne compréhension des mécanismes en jeu, on a détaillé la réponse à cette question. Cependant, compléter correctement le programme **Scilab** démontre la bonne compréhension de la simulation demandée et permet certainement d'obtenir tous les points alloués à cette question.
On procédera de même dans les autres questions **Scilab**.

□

b) Compléter les lignes 4 et 9 ajoutées au script précédent afin que le script qui suit renvoie et affiche, en plus de celle prise par X, la valeur prise par Y.

```
1  n = input('Entrez une valeur pour n : ')
2  nB = n-1
3  X = 1
4  Y = ----
5  u = grand(1, 1, 'uin', 1, nB+1)
6  while u < nB + 1
7      nB = ----
8      if u == 1 then
9          Y = ----
10     end
11     u = grand(1, 1, 'uin', 1, ----)
12     X = ----
13 end
14 disp(X, 'La boule noire est apparue au tirage numéro ')
15 disp(Y, 'La valeur de Y est ')
```

Démonstration.

- Il s'agit de simuler la v.a.r. Y. On crée la variable `Y`, destinée à contenir la valeur prise par Y lors de la simulation. On initialise `Y` à 0 puisqu'avant les tirages, la boule 1 n'a pas encore été piochée.

```
4  Y = 0
```

- Il faut alors tester, pour chaque tirage simulé (à chaque tour de boucle) si l'on a obtenu la boule numérotée 1. La structure conditionnelle du programme permet justement de tester si la variable `u`, qui contient le numéro de la boule tirée, vaut 1. Si c'est le cas, c'est qu'on a tiré la boule 1. Il faut alors mettre à jour la variable `Y` en conséquence.

```
8      if u == 1 then
9          Y = 1
10     end
```

Commentaire

- Il faut bien comprendre qu'à chaque simulation de tirage on procède à une renumérotation des boules. Reprenons l'exemple détaillé précédemment. Initialement l'urne contient 5 boules numérotées de 1 à 5. La boule 5 est noire et les autres blanches. La boule 1 porte l'inscription 1 et les boules 2, 3 et 4 portent l'inscription 0. Imaginons les tirages suivants :
  - × si on tire la boule 2, alors les 4 boules restantes dans l'urne sont renumérotées de 1 à 4. La 4 est la noire, les boules 1, 2, 3 sont blanches et c'est la 1 qui portent l'inscription 1.
  - × si on tire enstuie la boule 1, alors on renumérote les boules : la 3 est la noire, les boules 1 et 2 sont blanches portent toutes deux l'inscription 0.
  - × si on tire ensuite la boule 1, alors on renumérote les boules : la 2 est la noire, la 1 est blanche et porte l'inscription 0.

  Cette série de tirages, permet de comprendre que la renumérotation peut provoquer plusieurs tirages successifs de boules boules 1. Mais celles-ci ne sont pas considérées comme portant l'inscription 1 tout au long du programme.
- Il est à noter que seule la première mise à jour de `Y` a un effet : si la variable `Y` contient 1, les mises à jour suivantes écrasent cette valeur pour la remplacer par 1. □

II. Illustration de la LfGN / méthode de Monte-Carlo

II.1. Valeur approchée d'une probabilité

ESSEC – I – 2015

- On considère une v.a.r. D (de densité f nulle sur $]-\infty, 0[$) et on note R la fonction définie sur $[0, +\infty[$ par $R(x) = \mathbb{P}([D > x])$.
- On suppose que l'on a défini une fonction d'entête `function r = R(x)` qui renvoie la valeur de `R` au point `x`. On considère X une v.a.r. qui suit la loi exponentielle de paramètre 1.
- On cherche alors à calculer une valeur approchée du réel S défini par :

$$\mathbb{P}\left(\left[R(X) \leqslant \frac{k+c}{v}\right]\right) = \mathrm{e}^{-S}$$

où k (coût de stockage), c (coût d'achat), et v (prix de vente) sont des constantes.

a) Compléter le script **Scilab** qui suit, puis expliquer pourquoi il affiche une valeur approchée de S.

```
1  k = input('k = '); c = input('c = '); v = input('v = ');
2  compt = 0;
3  for i = 1:1000 do
4      X = grand(1, 1, "exp", 1)
5      if ---
6          compt = compt + 1;
7      end
8  end
9  disp('S = '); disp(-log(compt/1000));
```

Démonstration.

```
5      if R(X) <= (k+c)/v
```

□

EDHEC – 2015

- Trois personnes, notées A, B et C entrent simultanément dans une agence bancaire disposant de deux guichets. Les clients A et B occupent simultanément à l'instant 0 les deux guichets tandis que C attend que l'un des deux guichets se libère pour se faire servir.
  On suppose que :
  - × les durées de passage au guichet des trois personnes A, B et C sont mesurées en heures et on suppose que ce sont des variables aléatoires indépendantes, notées respectivement X, Y et Z, et suivant toutes la loi uniforme sur $[0, 1[$.
  - × la durée du changement de personne à un guichet est négligeable.
- On pose $U = \min(X, Y)$ et $V = \max(X, Y)$ et on admet que U et V sont des v.a.r. .
  On note T le temps total passé par C dans l'agence bancaire.
- On rappelle que, si `a` et `b` sont deux vecteurs lignes de taille n, les commandes `m = min(a,b)` et `M = max(a,b)` renvoient les vecteurs `m` et `M`, de même taille que `a` et `b`, et tels que, pour tout i de $[\![1, n]\!]$, on ait : `m(i) = min(a(i),b(i))` et `M(i) = max(a(i),b(i))`.
- On rappelle également que `grand(1,n,'unf',0,1)` simule n variables aléatoires indépendantes suivant la loi uniforme sur $[0, 1[$.

a) Compléter les commandes **Scilab** suivantes pour qu'elles permettent de simuler n fois les variables aléatoires U, V et T, pour n entré par l'utilisateur :

```
1  n = input('entrez la valeur de n :')
2  x = grand(1,n,'unf',0,1)
3  y = grand(1,n,'unf',0,1)
4  z = grand(1,n,'unf',0,1)
5  u = --- ; disp(u, 'u = ')
6  v = --- ; disp(v, 'v = ')
7  t = --- ; disp(t, 't = ')
```

Démonstration.

```
5  u = min(x,y) ; disp(u, 'u = ')
6  v = max(x,y) ; disp(v, 'v = ')
7  t = u + z ; disp(t, 't = ')
```

□

b) Que représente l'événement $[T \geqslant V]$?

Démonstration.
L'événement $[T \geqslant V]$ est réalisé si et seulement si la personne C sort de l'agence après que les personnes A et B soient sorties. □

c) On souhaite déterminer une valeur approchée de la probabilité $\mathbb{P}([T \geqslant V])$, notée p, en simulant un grand nombre de fois le passage des clients A, B et C aux guichets.
Compléter les commandes `p = ----- ; disp(p, 'p = ')` pour que, placées sous les commandes écrites dans le programme précédent, elles permettent d'obtenir une valeur approchée de p.

Démonstration.

```
8  p = length(find(t >= v)) / n
9  disp(p, 'p = ')
```

□

d) Lors de plusieurs essais des commandes ci-dessus, avec $n = 10000$, la réponse donnée par `Scilab` est comprise entre 0.66 et 0.67.
Que peut-on conjecturer quant à la valeur exacte de p ?

Démonstration.
On peut conjecturer : $p = \dfrac{2}{3}$. □

`ECRICOME – 2017`

- Soit n un entier naturel non nul.
On effectue une série illimitée de tirages d'une boule avec remise dans une urne contenant n boules numérotées de 1 à n. Pour tout entier naturel k non nul, on note X_k la variable aléatoire égale au numéro de la boule obtenue au k-ième tirage.
Pour tout entier naturel k non nul, on note S_k la somme des numéros des boules obtenues lors des k premiers tirages :

$$S_k = \sum_{i=1}^{k} X_i$$

On considère enfin la v.a.r. T_n égale au nombre de tirages nécessaires pour que, pour la première fois, la somme des numéros des boules obtenues soit supérieure ou égale à n.
Exemple : avec $n = 10$, si les numéros obtenus aux cinq premiers tirages sont dans cet ordre 2, 4, 1, 5 et 9, alors on obtient : $S_1 = 2$, $S_2 = 6$, $S_3 = 7$, $S_4 = 12$, $S_5 = 21$ et $T_{10} = 4$.

- On montre que la suite de v.a.r. $(T_n)_{n\geqslant 1}$ converge en loi vers la v.a.r. Y à valeurs dans $\mathbb{N}^*$ définie par :

$$\forall k \in \mathbb{N}^*,\ \mathbb{P}([Y=k]) = \frac{k-1}{k!}$$

a) On rappelle qu'en langage **Scilab**, l'instruction `grand(1,1,'uin',1,n)` renvoie un entier aléatoire de $[\![1,n]\!]$. Compléter la fonction ci-dessous, qui prend en argument le nombre n de boules contenues dans l'urne, afin qu'elle simule la variable aléatoire T_n :

```
1  function y = T(n)
2      S = ................
3      y = ................
4      while ..............
5          tirage = grand(1,1,'uin',1,n)
6          S = S + tirage
7          y = ................
8      end
9  endfunction
```

Démonstration.

- On rappelle que la v.a.r. T_n est égale au nombre de tirages nécessaires pour que, pour la première fois, la somme des numéros des boules obtenue soit supérieure ou égale à n.
- C'est le paramètre de sortie `y` de la fonction `T` qui doit contenir le nombre de tirages obtenus par simulation. On procède comme suit :
  - × la variable `y` est initialisée à 0 (au début de l'expérience aucun tirage n'a été réalisé).
    On crée par ailleurs la variable `S` qui va permettre de sommer successivement les résultats obtenus par simulation d'un tirage d'une boule dans l'urne.
    La variable `S` est, elle aussi, initialisée à 0.

```
2      S = 0
3      y = 0
```

  - × on simule alors des tirages successifs dans l'urne. La simulation d'un tel tirage est fournie par l'instruction donnée dans l'énoncé :

```
5          tirage = grand(1, 1, 'uin', 1, n)
```

    À chaque nouveau tirage, on doit mettre à jour en conséquence les variables `y` et `S` : on ajoute 1 à `y` pour signaler qu'un nouveau tirage a eu lieu et on ajoute la valeur de ce tirage à `S`.

```
6          S = S + tirage
7          y = y + 1
```

    On s'intéresse ici au nombre de tirages nécessaires pour que la somme des résultats des tirages simulés dépasse n. On doit effectuer cette succession de tirages tant que cette somme n'a pas dépassé n.

```
4      while S < n
```

- En résumé, on obtient le programme suivant :

```
function y = T(n)
    S = 0
    y = 0
    while S < n
        tirage = grand(1, 1, 'uin', 1, n)
        S = S + tirage
        y = y + 1
    end
endfunction
```

□

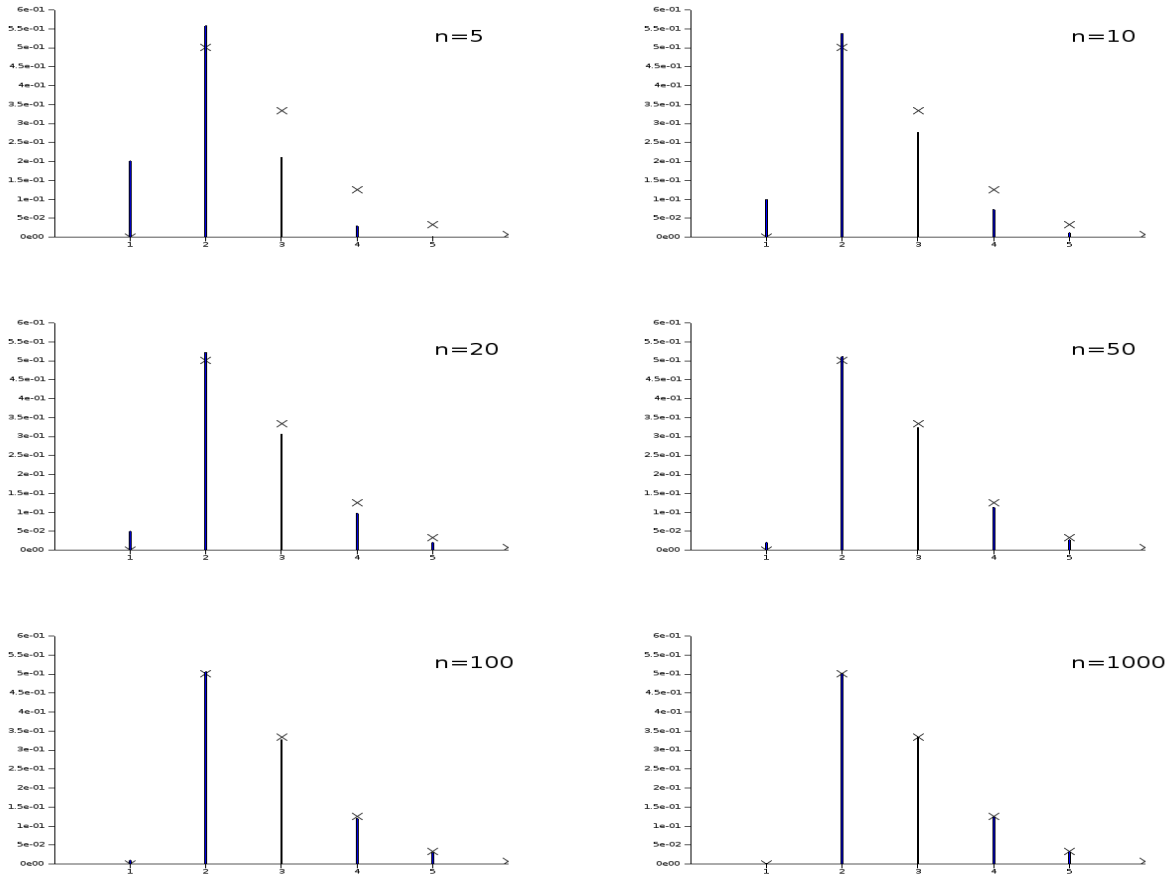
b) On suppose déclarée la fonction précédente et on écrit le script ci-dessous :

```
function y = freqT(n)
    y = zeros(1, n)
    for i = 1 : 100000
        k = T(n)
        y(k) = y(k) + 1
    end
    y = y / 100000
endfunction
```

```
function y = loitheoY(n)
    y = zeros(1, n)
    for k = 1 : n
        y(k) = (k-1) / prod(1:k)
    end
endfunction

clf
n = input('n = ?')
plot2d(loitheoY(6), style=-2)
x = freqT(n)
bar(x(1:5))
```

L'exécution de ce script pour les valeurs de n indiquées a permis d'obtenir les graphes ci-dessous :



Expliquer ce que représente les vecteurs renvoyés par les fonctions `freqT` et `loitheoY`. Comment ces vecteurs sont-ils représentés graphiquement dans chaque graphique obtenu ?

Démonstration.
Commençons par la description de la fonction `loitheoY`.

- La fonction `loitheoY` renvoie un paramètre nommé `y` qui est initialement affecté à un vecteur ligne de taille n rempli de 0 :

```
2          y = zeros(1,n)
```

- Le $k^{\text{ème}}$ coefficient de ce vecteur est ensuite affecté à la valeur $\dfrac{k-1}{k!}$.

```
4          y(k) = (k-1)/prod(1:k)
```

- On construit ainsi le vecteur : $\big(\mathbb{P}([Y=1]), \mathbb{P}([Y=2]), \dots, \mathbb{P}([Y=\texttt{n}])\big)$.

L'instruction `loitheoY(n)` renvoie un vecteur contenant la loi théorique de la v.a.r. Y.

Traitons maintenant le cas de la fonction `freqT`.

- Cette fonction renvoie un paramètre nommé `y` qui est initialement affecté à un vecteur ligne de taille n remplie de 0 :

```
2          y = zeros(1,n)
```

- La fonction a pour but de produire une approximation de la loi théorique de la v.a.r. T_n.
  On rappelle que $T_n(\Omega) = [\![1, n]\!]$. La loi de T_n est donc entièrement déterminée par les valeurs :
  $$\big(\mathbb{P}([T_n = 1]), \mathbb{P}([T_n = 2]), \dots, \mathbb{P}([T_n = n])\big)$$
- Pour ce faire, l'idée est :
  - × de simuler un grand nombre de fois ($N = 100000$ est ce grand nombre) la v.a.r. T_n.
    Formellement, on souhaite obtenir un N-uplet $(z_1, \dots, z_N)$ qui correspond à l'observation d'un N-échantillon $(Z_1, \dots, Z_N)$ de la v.a.r. T_n.
    (cela signifie que les v.a.r. Z_1, …, Z_N sont indépendantes et sont de même loi que T_n)
  - × de compter le nombre de 1, de 2, …, de n contenus dans cette observation.

  Cette idée est justifiée par la loi faible des grands nombres (LfGN) qui affirme :
  $$\forall k \in [\![1, n]\!], \quad \frac{\text{nombre de } k \text{ de l'observation}}{\text{taille } (N) \text{ de l'observation}} \simeq \mathbb{P}([T_n = k])$$
- Dans le programme, les valeurs $(z_1, \dots, z_N)$ sont obtenues par des appels successifs (à l'aide d'une structure itérative, ici une boucle `for`) à la fonction `T` et stockées les unes après les autres dans la variable `k`.

```
3   for i = 1 : 100000
4     k = T(n)
```

  Le tableau `y` est alors mis à jour à chaque tour de boucle :

```
5     y(k) = y(k) + 1
```

  Détaillons cette mise à jour :
  - × si `k` vaut 1 alors l'instruction suivante est effectuée :
    ```
    y(1) = y(1) + 1
    ```
  - × …
  - × si `k` vaut n alors l'instruction suivante est effectuée :
    ```
    y(n) = y(n) + 1
    ```

  Cela signifie que le $k^{\text{ème}}$ coefficient du tableau compte le nombre de k de l'observation.
  Une fois cette boucle effectuée, l'approximation formulée par la LfGN est obtenue en divisant ces nombres par la taille de l'observation :

```
7   y = y / 100000
```

- En résumé, l'instruction `freqT(n)` renvoie un vecteur qui donne la fréquence d'apparition de chaque entier $k \in [\![1, \mathtt{n}]\!]$ dans une observation de taille 100000 de la v.a.r. T_n.

> D'après la LfGN, `freqT(n)` renvoie un vecteur qui est une approximation du vecteur $\big(\mathbb{P}([T_n = 1]), \mathbb{P}([T_n = 2]), \dots, \mathbb{P}([T_n = n])\big)$.

Commentons enfin la représentation graphique.

- La ligne 10 du script permet de générer la représentation graphique du vecteur issu de la fonction `loitheoY` :

```
10  plot2d(loitheoY(6), style=-2)
```

Rappelons que `style=-2` est l'argument de la commande `plot2d` permettant d'afficher les points dans le plan avec des croix.

Le vecteur `loitheoY(6)` est représenté graphiquement par des croix.

- La ligne 10 du script permet de générer la représentation graphique du vecteur issu de la fonction `freqT` :

```
11 x = freqT(n)
12 bar(x(1:5))
```

Rappelons que `bar` est une commande générant un diagramme en bâtons.

Le vecteur `x` issu de l'instruction `x = freqT(n)` est représenté par un diagramme en bâtons.

Commentaire

- Il faut noter qu'on ne représente pas la loi de Y en intégralité puisque l'appel `loitheoY(6)` produit le vecteur :

$$\big(\mathbb{P}([Y=1]),\mathbb{P}([Y=2]),\dots,\mathbb{P}([Y=6])\big)$$

De même, l'instruction `x(1:5)` sélectionne les 5 premiers coefficients du vecteur généré par `x = freqT(n)`. On obtient ainsi une approximation du vecteur :

$$\big(\mathbb{P}([T_n=1]),\mathbb{P}([Y=2]),\dots,\mathbb{P}([T_n=5])\big)$$

- Les croix sont placées exactement au même endroit sur les 6 graphiques. C'est logique puisque le vecteur `loitheoY(6)` est indépendant de la valeur prise par n.

□

c) Expliquer en quoi cette succession de graphiques permet d'illustrer la convergence en loi de (T_n) vers Y.

Démonstration.

- D'après la question précédente :
  - × le vecteur `freqT(n)` est une approximation de la loi de $T_{\mathtt{n}}$.
  - × le vecteur `loitheoY(n)` représente la loi de Y.
- La succession de graphiques démontre que, lorsque la valeur de `n` grandit, les coefficients de `freqT(n)` se rapprochent de ceux de `loitheoY(n)`. Dès la valeur `n` $= 50$, les deux graphiques semblent très proches. Pour `n` $= 1000$ les deux graphiques coïncident même à l'œil nu.
- Cette succession de graphiques illustre donc la propriété :

$$\forall k\in\mathbb{N}^*,\quad \mathbb{P}([T_n=k])\underset{n\to+\infty}{\longrightarrow}\mathbb{P}([Y=k])$$

(le $k^{\text{ème}}$ coefficient de `freqT(n)` se rapproche du $k^{\text{ème}}$ coefficient de `loitheoY(n)` lorsque l'entier `n` grandit)

Cette succession de graphiques permet d'illustrer la convergence en loi de (T_n) vers Y.

Commentaire

- Cette succession de graphiques permet aussi de parler de vitesse de convergence. On s'aperçoit que l'approximation de `loitheoY(n)` par `freqT(n)` est déjà correcte pour `n` = 10 ; satisfaisante pour `n` = 20 ; bonne pour `n` = 50 et encore meilleure pour les valeurs suivantes. Ceci suggère que la v.a.r. T_n converge en loi vers la v.a.r. Y de manière très rapide.
- On a évoqué en question ***15.*** la loi faible des grands nombres (LfGN). L'idée est de simuler un grand nombre de fois (N fois) la v.a.r. T_n (à n fixé) afin d'obtenir une répartition des valeurs possibles de T_n et ainsi une approximation de la loi de T_n. Dans l'énoncé, on fait le choix $N = 100000$. Cela suggère que le résultat de convergence fournit par la LfGN a une vitesse de convergence faible : N doit être un nombre réellement **grand** pour que l'approximation obtenue soit bonne.

□

`HEC – 2019`

- On note S une variable aléatoire à valeurs dans $\{-1, 1\}$ dont la loi est donnée par :

$$\mathbb{P}([S = -1]) \;=\; \mathbb{P}([S = +1]) \;=\; \frac{1}{2}$$

- On note X_2 une variable aléatoire qui suit la loi binomiale $\mathcal{B}\left(2, \frac{1}{2}\right)$.
  On suppose que les variables aléatoires X_2 et S sont indépendantes et on pose $Y_2 = S\,X_2$.
- On devait démontrer que la v.a.r. $X_2 - (S + 1)$ suit la même loi que Y_2.

Le script **Scilab** suivant permet d'effectuer des simulations de la variable aléatoire Y_2 définie dans la question précédente.

```
1  n = 10
2  X = grand(n,2,'bin',2,0.5)
3  B = grand(n,2,'bin',1,0.5)
4  S = 2 * B - ones(n,2)
5  Z1 = [S(1:n,1) .* X(1:n,1) , X(1:n,1) - S(1:n,1) - ones(n,1)]
6  Z2 = [S(1:n,1) .* X(1:n,1) , X(1:n,2) - S(1:n,2) - ones(n,1)]
```

a) Que contiennent les variables `X` et `S` après l'exécution des quatre premières instructions ?

Démonstration.

- L'instruction `X = grand(n,2,'bin',2,0.5)` permet de stocker dans la variable `X` une matrice à `n` lignes et 2 colonnes où chaque colonne contient une observation d'un `n`-échantillon de loi $\mathcal{B}\left(2, \frac{1}{2}\right)$, c'est-à-dire un `n`-échantillon de la v.a.r. X. Plus précisément, la première colonne de `X` contient une observation $(c_1, \dots, c_n)$ d'un `n`-échantillon $(C_1, \dots, C_n)$ de X, et la deuxième colonne de `X` contient une observation $(c'_1, \dots, c'_n)$ d'un `n`-échantillon $(c'_1, \dots, C'_n)$ de `X`.
  (les v.a.r. C_i et C'_i sont indépendantes et ont même loi que la v.a.r. X)
- L'instruction `B = grand(n,2,'bin',1,0.5)` permet de stocker dans la variable `B` une matrice à `n` lignes et 2 colonnes où la première colonne contient une observation $(b_1, \dots, b_n)$ d'un `n`-échantillon $(B_1, \dots, B_n)$ de loi $\mathcal{B}\left(\frac{1}{2}\right)$, et la deuxième colonne contient une observation $(b'_1, \dots, b'_n)$ d'un `n`-échantillon $(B'_1, \dots, B'_n)$ de loi $\mathcal{B}\left(\frac{1}{2}\right)$.
  (les v.a.r. B_i et B'_i sont indépendantes et ont même loi $\mathcal{B}\left(\frac{1}{2}\right)$)

- On commence par rappeler que l'instruction `ones(n,2)` permet d'obtenir une matrice à `n` lignes et 2 colonnes dont tous les coefficients sont égaux à 1.
  On en déduit que l'instruction `S = 2 * B - ones(n,2)` permet de stocker dans la variable `S` une matrice à `n` lignes et 2 colonnes.
  La première colonne contient l'observation $(s_1, \dots, s_n) = (2\,b_1 - 1, \dots, 2\,b_n - 1)$ du `n`-échantillon $(2\,B_1 - 1, \dots, 2\,B_n - 1)$.
  La deuxième colonne contient l'observation $(s'_1, \dots, s'_n) = (2\,b'_1 - 1, \dots, 2\,b'_n - 1)$ du `n`-échantillon $(2\,B'_1 - 1, \dots, 2\,B'_n - 1)$.
- Les v.a.r. $2\,B_i - 1$ et $2\,B'_i - 1$ sont indépendantes et de même loi. Déterminons cette loi : on note B une v.a.r. de loi $\mathcal{B}\left(\frac{1}{2}\right)$ et on cherche la loi de $V = 2\,B - 1$.
  - × Tout d'abord, comme $B \hookrightarrow \mathcal{B}\left(\frac{1}{2}\right)$, on a : $B(\Omega) = \{0, 1\}$.
    On en déduit : $V(\Omega) = \{2 \times 0 - 1, 2 \times 1 - 1\} = \{-1, 1\}$.
  - × De plus :
    $$[V = 1] \;=\; [2\,B - 1 = 1] \;=\; [2\,B = 2] \;=\; [B = 1]$$
    D'où : $\mathbb{P}([V = 1]) = \mathbb{P}([B = 1]) = \dfrac{1}{2}$
  - × Enfin, comme la famille $([V = -1], [V = 1])$ est un système complet d'événements :
    $$\mathbb{P}([V = -1]) \;=\; 1 - \mathbb{P}([V = 1]) \;=\; 1 - \frac{1}{2} \;=\; \frac{1}{2}$$
    On en déduit que la v.a.r. $V = 2\,B - 1$ suit la même loi que la v.a.r. S.
- La variable `S` contient donc une matrice à `n` lignes et 2 colonnes, où la première colonne contient une observation $(s_1, \dots, s_n)$ d'un `n`-échantillon $(S_1, \dots, S_n)$ de la v.a.r. S, et la deuxième colonne contient une observation $(s'_1, \dots, s'_n)$ du `n`-échantillon $(S'_1, \dots, S'_n)$ de la v.a.r. S.
  (les v.a.r. S_i et S'_i sont indépendantes et ont même loi que la v.a.r. S)

Commentaire

- Comme souvent dans les sujets HEC, l'une des difficultés provient de la prise d'initiatives nécessaires pour répondre à une question.
- Dans cette question par exemple, déterminer la loi de la v.a.r. $V = 2\,B - 1$ n'est pas difficile. C'est prendre l'initiative de déterminer cette loi qui constitue la difficulté.

□

b) Expliquer pourquoi, après l'exécution des six instructions, chacun des coefficients des matrices `Z1` et `Z2` contient une simulation de la variable aléatoire Y_2.

Démonstration.

- L'instruction `Z1 = [S(1:n,1) .* X(1:n,1) , X(1:n,1) - S(1:n,1) - ones(n,1)]` permet de stocker dans la variable `Z1` une matrice à `n` lignes et 2 colonnes.
  Le contenu de la première colonne est donné par la commande `S(1:n,1) .* X(1:n,1)` et celui de la deuxième colonne par `X(1:n,1) - S(1:n,1) - ones(n,1)`.
- L'instruction `S(1:n,1) .* X(1:n,1)` permet d'obtenir une matrice colonne à `n` lignes contenant l'observation $(d_1, \dots, d_n) = (s_1 \times c_1, \dots, s_n \times c_n)$ du `n`-échantillon $(S_1\,C_1, \dots, S_n\,C_n)$.
  Or, les S_i suivent la même loi que S et les C_i suivent la même loi que X_2. Ainsi, les $S_i\,C_i$ suivent la même loi que Y_2.
  La première colonne de `Z1` contient donc une observation $(d_1, \dots, d_n)$ d'un `n`-échantillon $(D_1, \dots, D_n)$ de Y_2.
  (les v.a.r. D_i sont indépendantes et ont même loi que la v.a.r. Y_2)

- L'instruction `X(1:n,1) - S(1:n,1) - ones(n,1)` permet d'obtenir une matrice colonne à `n` lignes contenant l'observation $(d'_1, \dots, d'_n) = (c_1 - s_1 - 1, \dots, c_n - s_n - 1)$ du `n`-échantillon $(C_1 - S_1 - 1, \dots, C_n - S_n - 1)$.
  Or, les S_i suivent la même loi que S et les C_i suivent la même loi que X_2. Ainsi, les $C_i - S_i - 1$ suivent la même loi que $X_2 - S - 1$. De plus, d'après la question ***2.b)***, la v.a.r. $X_2 - S - 1$ suit la même loi que Y_2.
  La deuxième colonne de `Z1` contient donc une observation $(d'_1, \dots, d'_n)$ d'un `n`-échantillon $(D'_1, \dots, D'_n)$ de Y_2.
  (les v.a.r. D'_i sont indépendantes et ont même loi que la v.a.r. Y_2)
- De même, l'instruction `Z2 = [S(1:n,1) .* X(1:n,1) , X(1:n,2) - S(1:n,2) - ones(n,1)]` permet de stocker dans la variable `Z2` une matrice à `n` lignes et 2 colonnes.
  Le contenu de la première colonne est donné par la commande `S(1:n,1) .* X(1:n,1)` et celui de la deuxième colonne par `X(1:n,2) - S(1:n,2) - ones(n,1)`.
- La première colonne de `Z2` est identique à celle de `Z1`, donc la première colonne de `Z2` contient l'observation $(d_1, \dots, d_n)$ du `n`-échantillon $(D_1, \dots, D_n)$ de Y_2.
- L'instruction `X(1:n,2) - S(1:n,2) - ones(n,1)` permet d'obtenir une matrice colonne à `n` lignes contenant l'observation $(d''_1, \dots, d''_n) = (c'_1 - s'_1 - 1, \dots, c'_n - s'_n - 1)$ du `n`-échantillon $(C'_1 - S'_1 - 1, \dots, C'_n - S'_n - 1)$.
  Or, les S'_i suivent la même loi que S et les C'_i suivent la même loi que X_2. Ainsi, les $C'_i - S'_i - 1$ suivent la même loi que $X_2 - S - 1$, donc que Y_2.
  La deuxième colonne de `Z2` contient donc une observation $(d''_1, \dots, d''_n)$ d'un `n`-échantillon $(D''_1, \dots, D''_n)$ de Y_2.
  (les v.a.r. D''_i sont indépendantes et ont même loi que la v.a.r. Y_2)

Finalement, chacun des coefficients des matrices `Z1` et `Z2` contient une simulation de la v.a.r. Y_2.

Commentaire

On aurait pu utiliser davantage les commandes Scilab. En effet, l'appel classique permettant d'extraire la première colonne de la matrice `S` est plutôt `S(:,1)` (que `S(1:n,1)`). □

c) On modifie la première ligne du script précédent en affectant à `n` une valeur beaucoup plus grande que 10 (par exemple, 100000) et en lui adjoignant les deux instructions 7 et 8 suivantes :

```
7  p1 = length(find(Z1(1:n,1) == Z1(1:n,2))) / n
8  p2 = length(find(Z2(1:n,1) == Z2(1:n,2))) / n
```

Quelles valeurs numériques approchées la loi faible des grands nombres permet-elle de fournir pour `p1` et `p2` après l'exécution des huit lignes du nouveau script ?

Dans le langage **Scilab**, la fonction `length` fournit la « longueur » d'un vecteur ou d'une matrice et la fonction `find` calcule les positions des coefficients d'une matrice pour lesquels une propriété est vraie, comme l'illustre le script suivant :

```
--> A = [1 ; 2 ; 0 ; 4]
--> B = [2 ; 2 ; 4 ; 3]
--> length(A)
    ans = 4.
--> length([A , B])
    ans = 8.
--> find(A < B)
    ans = 1. 3. // car 1 < 2 et 0 < 4, alors que 2 ⩾ 2 et 4 ⩾ 3
```

Démonstration.

- Commençons par commenter l'instruction :

```
7  p1 = length(find(Z1(1:n,1) == Z1(1:n,2))) / n
```

  × L'instruction `find(Z1(1:n,1) == Z1(1:n,2))` permet d'obtenir une matrice ligne contenant les positions des coefficients des matrices `Z1(1:n,1)` et `Z1(1:n,2)` égaux. Autrement dit, on obtient une matrice ligne contenant les indices i tels que $d_i = d'_i$.
  (on rappelle que $(d_1, \dots, d_n)$ est une observation d'un `n`*-échantillon $(D_1, \dots, D_n)$ de $S\,X_2$ et $(d'_1, \dots, d'_n)$ est une observation d'un* `n`*-échantillon $(D'_1, \dots, D'_n)$ de $X_2 - S - 1$)*

  × L'instruction `length(find(Z1(1:n,1) == Z1(1:n,2)))` permet d'obtenir la longueur de la matrice précédente. Ainsi, on obtient le nombre de fois où $d_i = d'_i$, pour $i \in [\![1, \mathtt{n}]\!]$.

  × Enfin, on divise ce nombre par la taille `n` de l'observation.
  Or, par loi faible des grands nombres (LfGN) :

$$\frac{\text{nombre de fois où } d_i = d'_i}{\text{taille de l'observation}} \;\simeq\; \mathbb{P}([S\,X_2 = X_2 - S - 1])$$

La variable `p1` contient une valeur approchée de $\mathbb{P}([S\,X_2 = X_2 - S - 1])$.

- Commentons ensuite l'instruction :

```
8  p2 = length(find(Z2(1:n,1) == Z2(1:n,2))) / n
```

  × L'instruction `find(Z2(1:n,1) == Z2(1:n,2))` permet d'obtenir une matrice ligne contenant les positions des coefficients des matrices `Z1(2:n,1)` et `Z1(2:n,2)` égaux. Autrement dit, on obtient une matrice ligne contenant les indices i tels que $d_i = d''_i$.
  (on rappelle que $(d_1, \dots, d_n)$ est une observation d'un `n`*-échantillon $(D_1, \dots, D_n)$ de $S\,X_2$ et $(d''_1, \dots, d''_n)$ est une observation d'un* `n`*-échantillon $(D''_1, \dots, D''_n)$ de $X'_2 - S - 1$, où la v.a.r. X'_2 suit la même loi que X_2)*

  × L'instruction `length(find(Z2(1:n,1) == Z2(1:n,2)))` permet d'obtenir la longueur de la matrice précédente. Ainsi, on obtient le nombre de fois où $d_i = d''_i$, pour $i \in [\![1, \mathtt{n}]\!]$.

  × Enfin, on divise ce nombre par la taille `n` de l'observation.
  Or, par loi faible des grands nombres (LfGN) :

$$\frac{\text{nombre de fois où } d_i = d''_i}{\text{taille de l'observation}} \;\simeq\; \mathbb{P}(\big[S\,X_2 = X'_2 - S - 1\big])$$

La variable `p2` contient une valeur approchée de $\mathbb{P}([S\,X_2 = X'_2 - S - 1])$ où la v.a.r. X'_2 suit la même loi que X_2.

Commentaire

- Le programme proposé par l'énoncé n'est ici rien d'autre qu'une illustration de l'idée naturelle pour obtenir une approximation de $\mathbb{P}([S\,X_2 = X_2 - S - 1])$:
  - × simuler un grand nombre de fois ($\texttt{n} = 100000$) les v.a.r. $S\,X_2$ et $X_2 - S - 1$.
    Formellement, on souhaite obtenir une observation $(d_1, \ldots, d_n)$ d'un $\texttt{n}$-échantillon $(D_1, \ldots, D_n)$ de la v.a.r. $S\,X_2$, et une observation $(d'_1, \ldots, d'_n)$ d'un $\texttt{n}$-échantillon $(D'_1, \ldots, D'_n)$ de la v.a.r. $X_2 - S - 1$.
  - × de compter le nombre de fois où $d_i = d'_i$, pour $i \in [\![1, \texttt{n}]\!]$.
- L'objectif de cette question **Scilab** est de revenir sur un point important en probabilités :

$$X \text{ et } Y \text{ ont même loi} \quad \not\Rightarrow \quad X = Y$$

  (mais bien sûr, si $X = Y$, alors X et Y ont même loi)
- On peut avoir la confirmation du point précédent en exécutant le programme **Scilab**.
  On obtient :

```
p1 =
    0.1261
p2 =
    0.2179
```

  On constate qu'on a $\texttt{p1} \neq 1$ et $\texttt{p2} \neq 1$. Ainsi, ici :
  - × les v.a.r. $S\,X_2$ et $X_2 - S - 1$ ont même loi,
  - × mais $\mathbb{P}([S\,X_2 = X_2 - S - 1]) \neq 1$. Cela démontre en particulier : $S\,X_2 \neq X_2 - S - 1$.

□

II.2. Valeur approchée d'une espérance

HEC – 2015

- On considère une v.a.r. X telle que $X \hookrightarrow \mathcal{E}(\lambda)$.
- On considère la fonction F définie sur $\mathbb{R}$ à valeurs réelles telle que : $F(x) = \exp(-\mathrm{e}^{-\lambda x})$.
  F est la fonction de répartition d'une v.a.r. T. *(T suit la loi de Gumbel de paramètre λ)*
- On peut démontrer que $Z = -\dfrac{1}{\lambda}\,\ln(\lambda X)$ a même loi que T.
- On suppose alors que $\lambda = 1$.
  On démontre que F réalise une bijection de $\mathbb{R}$ dans $]0,1[$ et que sa bijection réciproque G est :

$$\begin{array}{rccl} G & : & \mathbb{R} & \to &]0,1[\\ & & x & \mapsto & -\ln(-\ln(x)) \end{array}$$

  On démontre alors que $G(U)$ a même loi que T (méthode d'inversion).

a) Par une méthode de votre choix, écrire en **Scilab** les commandes qui permettent de simuler la loi de T.

Démonstration.

- Méthode 1 : utilisation de la question ***10.b)***
  On propose la fonction **Scilab** suivante :

```
1  function t = SimuT()
2      x = grand(1, 1, 'exp', 1)
3      t = -log(x)
4  endfunction
```

Détaillons les éléments de ce script.

× **Début de la fonction**
On commence par préciser la structure de la fonction :

- cette fonction se nomme `SimuT`,
- elle ne prend pas de paramètre en entrée,
- elle admet pour variable de sortie la variable `t`

```
1  function t = SimuT()
```

× **Contenu de la fonction**
En ligne 2, on stocke dans la variable `x` une simulation d'une v.a.r. de loi $\mathcal{E}(1)$.

```
2      x = grand(1, 1, 'exp', 1)
```

D'après la question ***10.b)***, la v.a.r. $Z = -\ln(X)$ suit la même loi que T, dès lors que $X \hookrightarrow \mathcal{E}(1)$.
(on rappelle encore une fois que, dans cette question : $\lambda = 1$)
Ainsi, en ligne 3, on stocke dans la variable `t` une simulation de la v.a.r. T.

```
3      t = -log(x)
```

- Méthode 2 : utilisation de la question ***11.c)***
On propose la fonction **Scilab** suivante :

```
1  function t = SimuT()
2      u = rand()
3      t = -log( -log(x) )
4  endfunction
```

Détaillons les éléments de ce script.

× **Début de la fonction**
On commence par préciser la structure de la fonction :

- cette fonction se nomme `SimuT`,
- elle ne prend pas de paramètre en entrée,
- elle admet pour variable de sortie la variable `t`

```
1  function t = SimuT()
```

× **Contenu de la fonction**
En ligne 2, on stocke dans la variable `u` une simulation d'une v.a.r. de loi $\mathcal{U}(]0,1[)$.

```
2      u = rand()
```

D'après la question ***11.c)***, la v.a.r. $W = -\ln\big(-\ln(U)\big)$ suit la même loi que T, dès lors que $U \hookrightarrow \mathcal{U}(]0,1[)$.
Ainsi, en ligne 3, on stocke dans la variable `t` une simulation de la v.a.r. T.

```
3      t = -log( -log(x) )
```

□

b) Écrire en **Scilab** les commandes qui permettent de renvoyer une valeur numérique approchée de $\mathbb{E}(T)$ en utilisant la méthode de Monte-Carlo.

Démonstration.

- L'idée naturelle, qu'est la méthode de Monte-Carlo, pour obtenir une approximation de l'espérance $\mathbb{E}(T)$ est :
  - × de simuler un grand nombre de fois ($N = 10000$ sera ici ce grand nombre) la v.a.r. T.
    Formellement, on souhaite obtenir un N-uplet $(t_1, \dots, t_N)$ qui correspond à l'observation d'un N-échantillon $(T_1, \dots, T_N)$ de la v.a.r. T.
    (les v.a.r. T_i sont indépendantes et de même loi que T)
  - × de réaliser la moyenne des résultats de cette observation.

  Cette idée est justifiée par la loi faible des grands nombres (LfGN) qui affirme :

$$\text{moyenne de l'observation} \;=\; \frac{1}{N}\sum_{i=1}^{N} t_i \;\simeq\; \mathbb{E}(T)$$

- On propose alors le programme **Scilab** suivant :

```
1  N = 10000
2  T = zeros(1,N)
3  for k = 1:N
4      T(k) = SimuT()
5  end
6  E = mean(T)
7  disp(E)
```

Détaillons les éléments de ce script.

× **Début du programme**
La ligne 1 permet de stocker dans la variable `N` le nombre de réalisations souhaitées, ici 10000.

```
1  N = 10000
```

La ligne 2 permet de stocker dans la variable `T` un vecteur ligne à N colonnes constitué uniquement de 0. On souhaite remplir les coordonnées de ce vecteur avec N simulations de la v.a.r. T. C'est donc ce vecteur `T` qui contiendra le N-uplet d'observations $(t_1, \dots, t_N)$.

```
2  T = zeros(1,N)
```

× **Structure itérative**
Les lignes 3 à 5 permettent de stocker des les N coordonnées de `T` des simulations de la v.a.r. T, en exploitant la fonction `SimuT` définie dans la question précédente. Pour cela, on met en place une structure itérative (boucle for) :

```
3  for k = 1:N
4      T(k) = SimuT()
5  end
```

× **Fin du programme**
À l'issue de cette boucle, la variable `T` contient une observation du N-échantillon $(T_1, \dots, T_N)$ de la v.a.r. T.
On calcule alors la moyenne des observations avec la commande `mean` et on stocke ce résultat dans une variable `E`.

```
6  E = mean(T)
```

On finit en renvoyant cette moyenne.

```
7  disp(E)
```

Commentaire

- Un tel niveau d'explication n'est pas attendu aux concours : l'écriture du programme démontre la compréhension de toutes les commandes en question. On décrit ici de manière précise les instructions afin d'aider le lecteur un peu moins habile en **Scilab**.
- On a utilisé en ligne 6 la fonction `mean` prédéfinie en **Scilab**. D'autres solutions sont possibles :
  - × On peut aussi utiliser la fonction `sum` :

```
6  E = sum(T) / N
```

  - × On peut aussi effectuer la somme à l'aide d'une boucle :

```
1  N = 10000
2  T = 0
3  for k = 1:N
4      T = T + SimuT()
5  end
6  E = T / N
```

Toutes ces solutions rapportent sans aucun doute la totalité des points. □

ESSEC − I − 2016

- On considère X une v.a.r. à densité. On suppose qu'il existe un unique intervalle I_X que lequel F_X (fonction de répartition de X) réalise une bijection de classe C^1 strictement croissante de I_X sur $]0,1[$. On note alors G_X la bijection réciproque de F_X, définie de $]0,1[$ sur I_X.
- On note $\beta \in]0,1[$ un niveau de confiance.
- Enfin, on définit $r_\beta(X)$ appelé « Value at Risk » au niveau de confiance β de X, par :

$$r_\beta(X) = G_X(\beta)$$

C'est une grandeur qui permet d'évaluer le risque pris par l'acteur qui détient l'actif dont les pertes sont modélisées par X.

- On considère une suite de v.a.r. $(X_k)_{k\geqslant 1}$ mutuellement indépendantes et de même loi que X.
Pour tout $\omega \in \Omega$, et $n \in \mathbb{N}^*$, on ordonne $X_1(\omega), \ldots, X_n(\omega)$ dans l'ordre croissant.
On note alors $X_{1,n}(\omega), \ldots, X_{n,n}(\omega)$ les valeurs obtenues.
En particulier, $X_{1,n}(\omega)$ est la plus petite de ses valeurs et $X_{n,n}(\omega)$ la plus grande.
- Pour tout $n \in \mathbb{N}^*$ tel que $n\beta \geqslant 1$, on s'intéresse dans l'épreuve à $Y_n = X_{\lfloor n\beta \rfloor, n}$ qui est un estimateur de $r_\beta(X)$.
- On définit « l'Expected Shortfall » de X de niveau de confiance β par :

$$ES_\beta(X) = \frac{1}{1-\beta}\int_{r_\beta(X)}^{+\infty} x f_X(x)\, dx$$

On démontre que :

$$ES_\beta(X) = r_\beta(X) + \frac{1}{1-\beta}\,\mathbb{E}(\max(X - r_\beta(X), 0))$$

a) On suppose que l'on a défini une fonction d'entête `function R = triCroissant(T)` qui renvoie le tableau des valeurs se trouvant dans T rangées dans l'ordre croissant.
Par exemple :

```
--> triCroissant([0, -1, 0, 2, 4, 2, 3])
   ans  =
       [-1, 0, 0, 2, 2, 3, 4]
```

Écrire une fonction **Scilab** d'en-tête `function r = VaR(X, beta)` qui renvoie la valeur de l'estimation obtenue avec l'estimateur Y_n pour $r_\beta(X)$ si le tableau `X` contient la réalisation de l'échantillon $(X_1, \dots, X_n)$ et `beta` la valeur de β.

Démonstration.

```
1  function r = VaR(X,beta)
2    n = length(X)
3    Z = triCroissant(X)
4    r = Z(floor(n * beta))
5  endfunction
```

Détaillons l'obtention de cette fonction.
D'après la question précédente, on sait que $Y_n = X_{\lfloor n\beta \rfloor, n}$ est un estimateur convergent de $r_\beta(X)$.
Pour déterminer Y_n à partir de X_1, …, X_n, on souhaite donc :

1) définir n. Il s'agit de la taille de l'échantillon X_1, …, X_n.
Donc on stocke cette valeur dans la variable `n` avec la commande suivante :

```
2    n = length(X)
```

2) obtenir une réalisation de $X_{1,n}$, …, $X_{n,n}$, c'est-à-dire ordonner une réalisation de X_1, …, X_n dans l'ordre croissant.
On stocke ce nouveau vecteur dans la variable `Z` avec la commande suivante :

```
3    Z = triCroissant(x)
```

3) sélectionner dans ce vecteur la réalisation de $X_{\lfloor n\beta \rfloor, n}$, ce qui correspond à la $\lfloor n\beta \rfloor^{\text{ème}}$ coordonnée du vecteur `Z`.
On stocke ce résultat dans la variable de sortie `r`, ce qui donne :

```
4    r = Z(floor(n * beta))
```

où la commande `floor` correspond à la fonction partie entière. □

b) En utilisant la méthode de Monte-Carlo, dont on supposera la validité, et la fonction `VaR` précédente, écrire une fonction **Scilab** qui calcule une valeur approchée de $ES_\beta(X)$ à partir de la réalisation d'un échantillon de taille n de la loi de X dont les valeurs se trouvent dans le tableau `X` et de la valeur de β se trouvant dans la variable `beta`.

Démonstration.
On souhaite obtenir une valeur approchée de $ES_\beta(X)$. Pour ce faire, on doit tout d'abord obtenir une valeur approchée de l'espérance de la v.a.r. $V = h(X - r_\beta(X))$.

- L'idée naturelle pour obtenir une approximation de cette espérance est :
  - × de simuler un grand nombre de fois ($N = 10000$ par exemple) la v.a.r. V.
    Formellement, on souhaite obtenir un N-uplet $(v_1, \dots, v_N)$ qui correspond à l'observation d'un N-échantillon $(V_1, \dots, V_N)$ de la v.a.r. V.

× de réaliser la moyenne des résultats de cette observation.

Cette idée est justifiée par la loi faible des grands nombres (LfGN) qui affirme :

$$\text{moyenne de l'observation} = \frac{1}{N}\sum_{i=1}^{N} v_i \;\simeq\; \mathbb{E}(V)$$

- Il reste à obtenir le N-uplet $(v_1, \ldots, v_N)$. Pour ce faire, on dispose d'un N-uplet $(x_1, \ldots, x_N)$ qui correspond à l'observation d'un N-échantillon $(X_1, \ldots, X_N)$ de la v.a.r. X.
  Cette observation permet d'obtenir $(v_1, \ldots, v_N)$ en écrivant :

$$v_i \;=\; h\big(x_i - r_\beta(X)\big)$$

  Il est à noter que, comme le suggère l'énoncé, la valeur de $r_\beta(X)$ sera obtenue par l'appel à la fonction `VaR`.
- La valeur approchée de $ES_\beta(X)$ est alors obtenue par le calcul : $r_\beta(X) + \dfrac{1}{1-\beta}\,\dfrac{1}{N}\displaystyle\sum_{i=1}^{N} v_i$.
- Ce qui aboutit à la fonction suivante :

```
1  function ES = ExpectedShortfall(X, beta)
2      N = length(X)
3      V = zeros(1, N)
4      r = VaR(X, beta)
5      for i=1:N
6          V(i) = max(X(i)-r, 0)
7      end
8      ES = r + 1/(1-beta) * 1/N * sum(V)
9  endfunction
```

Détaillons les différents éléments de ce code :

× le paramètre **X** correspond à l'observation $(x_1, \ldots, x_N)$.

× en ligne 2, on récupère la taille de l'échantillon fourni en paramètre.

× en ligne 3, on crée une matrice ligne de taille N permettant à terme de stocker l'observation $(v_1, \ldots, v_N)$.

× en lignes 5 à 7, on effectue une boucle permettant d'obtenir successivement chaque v_i à l'aide de la valeur x_i fournit en paramètre.

× en ligne 8, on obtient la valeur approchée de $ES_\beta(X)$ en effectuant le calcul annoncé.

Commentaire

- Afin de permettre une bonne compréhension des mécanismes en jeu, on a détaillé la réponse à cette question. Cependant, écrire correctement la fonction **Scilab** démontre la bonne compréhension et permet certainement d'obtenir tous les points alloués.
- On s'est servi en ligne 6 de la fonction `max` prédéfinie en **Scilab**.
  Il était aussi possible d'utiliser une structure conditionnelle :

```
6          if X(i)-r > 0 then
7              V(i) = X(i)-r
8          else
9              V(i) = 0
10         end
```

- Enfin, il est possible de calculer la somme $\sum_{i=1}^{N} v_i$ sans créer une matrice de taille N. Pour ce faire, on crée une variable `S` qu'on initialise à 0 et qu'on met à jour dans la boucle. On obtient le programme suivant :

```
1  function ES = ExpectedShortfall(X, beta)
2      N = length(X)
3      S = 0
4      r = VaR(X, beta)
5      for i=1:N
6          S = S + max(X(i)-r, 0)
7      end
8      ES = r + 1/(1-beta) * 1/N * S
9  endfunction
```

□

ESSEC – I – 2018

Soit $x \in]0, \alpha[$.

On considère la fonction φ_x définie sur $\mathbb{R}_+$ par : $\varphi_x(t) = \begin{cases} t & \text{si } t \leqslant x \\ 0 & \text{sinon} \end{cases}$

On démontrait alors :

$$\sigma(x) = \frac{\mathbb{E}(\varphi_x(Y_{n-1}))}{\mathbb{P}([Y_{n-1} \leqslant x])}$$

où $X_1, \ldots, X_n$ est une famille de v.a.r. mutuellement indépendantes et de même loi que X (à valeurs dans $]0, \alpha[$) ; pour tout $\omega \in \Omega$, $Y_n(\omega) = \max(X_1(\omega), \ldots, X_n(\omega))$ est le plus grand des réels $X_1(\omega), \ldots, X_n(\omega)$.
Enfin, on suppose que l'on a défini une fonction **Scilab** d'entête `function x = simulX(n)` qui retourne une simulation d'un échantillon de taille n de la loi de X sous la forme d'un vecteur de longueur n.

1. En déduire une fonction **Scilab** `function s = sigma(x,n)` qui retourne une valeur approchée de $\sigma(x)$ obtenue comme quotient d'une estimation de $\mathbb{E}(\varphi_x(Y_{n-1}))$ et de $\mathbb{P}([Y_{n-1} \leqslant x])$.
On utilisera la fonction `simulX` pour simuler des échantillons de la loi de X, et on rappelle que si `v` est un vecteur, `max(v)` est égal au plus grand élément de `v`.

Démonstration.

- On commence par écrire une fonction qui retourne une réalisation de la v.a.r. Y_n.

```
function y = simulY(n)
  X = simulX(n)
  y = max(X)
endfunction
```

La première commande `X = simulX(n)` permet de créer un vecteur `X` contenant la réalisation d'un n-échantillon de même loi que X.
On sait de plus : $Y_n = \max(X_1, \dots, X_n)$. D'où la commande : `y = max(X)`.

- On code ensuite la fonction φ_x.

```
function v = phi(t,x)
  if t <= x then
    v = t
  else
    v = 0
  end
endfunction
```

- On finit par la fonction permettant d'obtenir une valeur approchée de $\sigma(x)$.

```
function s = sigma(x,n)
  N = 10000
  V = zeros(1,N)
  W = zeros(1,N)
  for k = 1:N
    z = simulY(n)
    V(k) = phi(z,x)
    if z <= x then
      W(k) = 1
    else
      W(k) = 0
    end
  end
  esp = mean(V)
  prob = mean(W)
  s = esp/prob
endfunction
```

Détaillons ce programme.

- La fonction `sigma` a pour but de produire une approximation :
  - × d'une part de $\mathbb{E}(\varphi_x(Y_{n-1}))$,
  - × d'autre part de $\mathbb{P}([Y_{n-1} \leqslant x])$.

  pour en déduire une valeur approchée de $\sigma(x) = \dfrac{\mathbb{E}(\varphi_x(Y_{n-1}))}{\mathbb{P}([Y_{n-1} \leqslant x])}$.
- Pour obtenir une valeur approchée de $\mathbb{E}(\varphi_x(Y_{n-1}))$, l'idée est :
  - × de simuler un grand nombre de fois ($N = 10000$ est ce grand nombre) la v.a.r. $\varphi_x(Y_{n-1})$.
    Formellement, on souhaite obtenir un N-uplet $(v_1, \dots, v_N)$ qui correspond à l'observation d'un N-échantillon $(V_1, \dots, V_N)$ de la v.a.r. $V = \varphi_x(Y_{n-1})$.
    (cela signifie que les v.a.r. V_1, …, V_N sont indépendantes et sont de même loi que la v.a.r. $V = \varphi_x(Y_{n-1})$)

× d'effectuer la moyenne de ces N observations.

Cette idée est justifiée par la loi faible des grands nombres (LfGN) qui affirme :

$$\frac{1}{N}\sum_{i=1}^{N} v_i \ \simeq\ \mathbb{E}(V) = \mathbb{E}(\varphi_x(Y_{n-1}))$$

- Dans le programme, les valeurs $(v_1, \dots, v_N)$ sont obtenues par des appels successifs (à l'aide d'une structure itérative, ici une boucle `for`) aux fonctions `simulY` et `phi`, et stockées les unes après les autres dans le vecteur `V`.

```
5    for k = 1:N
6      z = simulY(n)
7      V(k) = phi(z,x)
```

Une fois cette boucle terminée, l'approximation formulée par la LfGN est obtenue en effectuant la moyenne de ces observations :

```
14    esp = mean(V)
```

- Pour obtenir une valeur approchée de $\mathbb{P}([Y_{n-1} \leqslant x])$, l'idée est :
  × de simuler un grand nombre de fois (toujours N fois) la v.a.r. Y_{n-1}.
  Formellement, on souhaite obtenir un N-uplet $(z_1, \dots, z_N)$ qui correspond à l'observation d'un N-échantillon $(Z_1, \dots, Z_N)$ de la v.a.r. Y_{n-1}.
  × de compter le nombre de réalisations inférieures ou égales à x dans cette observation.

Cette idée est toujours guidée par la LfGN qui affirme :

$$\frac{\text{nombre de } z_i \text{ inférieurs à } x}{\text{taille } (N) \text{ de l'observation}} \ \simeq\ \mathbb{P}([Y_{n-1} \leqslant x])$$

- Dans le programme, on stocke, à l'aide d'une structure itérative (la même boucle `for` que précédemment), dans chaque coordonnée d'un vecteur `W` :
  × la valeur 1 si z_i est inférieur à x,
  × la valeur 0 si z_i est strictement supérieur à x.

```
8      if z <= x then
9        W(k) = 1
10     else
11       W(k) = 0
12     end
```

Une fois cette boucle terminée, l'approximation formulée par la LfGN est obtenue en effectuant la moyenne des coordonnées de `W` :

```
15    prob = mean(W)
```

- À ce stade, le programme fournit :
  × une valeur approchée de $\mathbb{E}(\varphi_x(Y_{n-1})$ stockée dans `esp`,
  × une valeur approchée de $\mathbb{P}([Y_{n-1} \leqslant x])$ stockée dans `prob`.

Enfin, pour approcher le réel $\sigma(x)$, on effectue le quotient de ces deux approximations :

```
16    s = esp/prob
```

Commentaire

- Un tel niveau d'explication n'est pas attendu aux concours : l'écriture du programme démontre la compréhension de toutes les commandes en question.
  On décrit ici de manière précise les instructions afin d'aider le lecteur un peu moins habile en **Scilab**.
- L'énoncé suggérait d'écrire une seule fonction, ce qui est tout à fait faisable en définissant `simulY` et `phi` à l'intérieur de la fonction `sigma`.
  Cependant, l'utilisation de sous-fonctions favorise la clarté d'un programme en le structurant. On privilégiera donc, lorsque c'est possible, l'écriture d'un programme à l'aide de sous-fonctions.
- On pouvait également utiliser la commande `find` pour obtenir une valeur approchée de $\mathbb{P}([Y_{n-1} \leqslant x])$:

```
1  function s = sigma(x,n)
2    N = 10000
3    Z = zeros(1,N)
4    V = zeros(1,N)
5    for k = 1:N
6      Z(k) = simulY(n)
7      V(k) = phi(Z(k),x)
8    end
9    esp = mean(V)
10   indices = find(Z <= x)
11   prob = length(indices)/N
12   s = esp/prob
13 endfunction
```

Dans le programme, le vecteur `Z` contient N réalisations de la v.a.r. Y_{n-1}.
La commande `(Z <= x)` fournit un vecteur de taille N de $i^{\text{ème}}$ coordonnée :

× le booléen « vrai » si la $i^{\text{ème}}$ coordonnée de `Z` est inférieure à x,

× le booléen « faux » sinon.

La commande `find(Z <= x)` permet alors d'obtenir le vecteur des indices pour lesquels le booléen est « vrai ».
Ainsi, `length(indices)` renvoie la longueur du vecteur `indices`, c'est-à-dire le nombre de booléens « vrai » dans le vecteur `(Z <= x)`. Autrement dit, `length(indices)` renvoie le nombre de réalisations de Y_{n-1} inférieures à x parmi les N observations.
Par loi faible des grands nombres : $\dfrac{\text{nombre de } z_i \text{ inférieurs à } x}{\text{taille } (N) \text{ de l'observation}} \simeq \mathbb{P}([Y_{n-1} \leqslant x])$.

Donc la variable `prob` est bien une valeur approchée de $\mathbb{P}([Y_{n-1} \leqslant x])$. □

II.3. Valeur approchée d'une intégrale

EDHEC – 2018

- On considère la fonction f qui à tout réel x associe : $f(x) = \displaystyle\int_0^x \ln(1+t^2)\,dt$.

1. On rappelle qu'en **Scilab**, la commande `grand(1, 1, 'unf', a, b)` simule une variable aléatoire suivant la loi uniforme sur $[a, b]$. Compléter le script **Scilab** suivant pour qu'il calcule et affiche, à l'aide de la méthode de Monte-Carlo, une valeur approchée de $f(1)$:

```
1  U = grand(1, 100 000, 'unf', 0, 1)
2  V = log(1 + U .^2)
3  f = -----
4  disp(f)
```

Démonstration.

- L'idée de la méthode de Monte-Carlo est de faire apparaître $f(1) = \displaystyle\int_0^1 \ln(1+t^2)\ dt$ sous forme d'une espérance qu'on pourra alors approcher à l'aide d'une simulation informatique.
- On considère U une v.a.r. telle que $U \hookrightarrow \mathcal{U}([0,1])$ de densité : $f_U : t \mapsto \begin{cases} 1 & \text{si } t \in [0,1] \\ 0 & \text{sinon} \end{cases}$.

  Notons alors V la v.a.r. définie par $V = g(U) = \ln(1+U^2)$.

  D'après le théorème de transfert, la v.a.r. V admet une espérance si et seulement si l'intégrale $\displaystyle\int_0^1 g(t)\ f_U(t)\ dt$ est absolument convergente.
  Les fonctions g et f_U étant à valeurs positives, cela revient à démontrer qu'elle est convergente.
  La fonction $t \mapsto g(t)\ f_U(t)$ étant de classe $\mathcal{C}^0_m$ sur $[0,1]$, l'intégrale $\displaystyle\int_0^1 g(t)\ f_U(t)\ dt$ est bien définie.

  La v.a.r. V admet une espérance.
- Enfin, par définition de f_U on obtient l'espérance de V sous la forme :

$$\mathbb{E}(V) \ = \ \int_0^1 g(t)\ f_U(t)\ dt \ = \ \int_0^1 \ln(1+t^2)\ dt \ = \ f(1)$$

- L'énoncé demande donc de déterminer une valeur approchée de $\mathbb{E}(V)$. L'idée naturelle pour obtenir une approximation de cette espérance est :
  - × de simuler un grand nombre de fois ($N = 100000$ par exemple) la v.a.r. V.
    Formellement, on souhaite obtenir un N-uplet $(v_1, \dots, v_N)$ qui correspond à l'observation d'un N-échantillon $(V_1, \dots, V_N)$ de la v.a.r. V.
    (les v.a.r. V_i sont indépendantes et ont même loi que V)
  - × de réaliser la moyenne des résultats de cette observation.

  Cette idée est justifiée par la loi faible des grands nombres (LfGN) qui affirme :

$$\text{moyenne de l'observation} = \frac{1}{N}\sum_{i=1}^{N} v_i \ \simeq \ \mathbb{E}(V)$$

- Cela se traduit de la manière suivante en **Scilab** :
  - × la ligne 1 permet d'obtenir des valeurs $(u_1, \dots, u_{100000})$ qui correspondent à l'observation d'un 100000-échantillon $(U_1, \dots, U_{100000})$ de la v.a.r. U.
  - × en ligne 2, on applique la fonction g à tous les éléments du 100000-uplet précédent, ce qui permet d'obtenir des valeurs $(v_1, \dots, v_{100000})$ qui correspondent à l'observation d'un 100000-échantillon $(V_1, \dots, V_{100000})$ de la v.a.r. V.
  - × en ligne 3, il faut calculer la moyenne de ces observations.
    On complète donc cette ligne comme suit.

```
3  f = mean(V)
```

Commentaire

- Un tel niveau d'explication n'est pas attendu aux concours : l'écriture de la ligne manquante démontre la compréhension de toutes les commandes en question.
  On décrit ici de manière précise les instructions afin d'aider le lecteur un peu moins habile en **Scilab**.
- On a utilisé en ligne 3 une fonction prédéfinie en **Scilab**. D'autres solutions sont possibles. Tout d'abord, on peut utiliser la fonction `sum` :

```
3  f = sum(V) / 100000
```

On peut aussi effectuer la somme à l'aide d'une boucle :

```
3  S = 0
4  for i = 1:100000
5      S = S + V(i)
6  end
7  f = S / 100000
```

Étant donné l'espace alloué par le programme (une ligne), le concepteur avait certainement en tête la première ou la deuxième solution. Cependant, il est raisonnable de penser que toute réponse juste sera comptée comme telle. Ainsi, la dernière solution rapporte certainement la totalité des points.

□

- On démontrait alors :
$$\sum_{k=0}^{+\infty} u_k = \int_0^1 \frac{1}{1-\ln(1+t^2)}\,dt$$

2. Modifier le script précédent pour donner un valeur approchée de $\sum_{k=0}^{+\infty} u_k$.

Démonstration.

- Comme en question ***6***, il s'agit d'utiliser la méthode de Monte-Carlo. L'idée est de faire apparaître $\int_0^1 \frac{1}{1-\ln(1+t^2)}\,dt$ sous forme d'une espérance qu'on pourra alors approcher à l'aide d'une simulation informatique.
- On considère U une v.a.r. telle que $U \hookrightarrow \mathcal{U}([0,1])$ de densité :
$$f_U : t \mapsto \begin{cases} 1 & \text{si } t \in [0,1] \\ 0 & \text{sinon} \end{cases}$$
Notons alors W la v.a.r. définie par $W = h(U) = \frac{1}{1-\ln(1+U^2)}$ où la fonction h est définie par :
$$h : t \mapsto \begin{cases} 0 & \text{si } t < 0 \\ \dfrac{1}{1-\ln(1+t^2)} & \text{si } t \in [0,1] \\ 0 & \text{si } t > 1 \end{cases}$$
- D'après le théorème de transfert, la v.a.r. W admet une espérance si et seulement si l'intégrale $\int_0^1 h(t)\ f_U(t)\ dt$ est absolument convergente.

Les fonctions g et f_U étant à valeurs positives, cela revient à démontrer qu'elle est convergente.
La fonction $t \mapsto h(t)\ f_U(t)$ étant de classe $\mathcal{C}^0_m$ sur $[0,1]$, l'intégrale $\displaystyle\int_0^1 h(t)\ f_U(t)\ dt$ est bien définie.

La v.a.r. W admet une espérance.

- Enfin, par définition de f_U on obtient l'espérance de W sous la forme :

$$\mathbb{E}(W) \ = \ \int_0^1 h(t)\ f_U(t)\ dt$$

- On peut donc appliquer la méthode de Monte-Carlo. Il s'agit, comme on l'a vu en question ***6*** d'approcher la valeur de $\mathbb{E}(W)$ par la quantité :

$$\frac{1}{N}\sum_{k=1}^{N} w_i$$

où $(w_1, \ldots, w_N)$ est un N-uplet d'observation du N-échantillon $(W_1, \ldots, W_N)$ de la v.a.r. W.

- En procédant comme en question ***6***, on obtient :

```
1 U = grand(1, 100 000, 'unf', 0, 1)
2 W = 1 / (1 - log(1 + U .^2) )
3 res = mean(W)
4 disp(res)
```

□

III. Chaînes de Markov

EDHEC – 2017

- Les sommets d'un carré sont numérotés 1, 2, 3, et 4 de telle façon que les côtés du carré relient le sommet 1 au sommet 2, le sommet 2 au sommet 3, le sommet 3 au sommet 4 et le sommet 4 au sommet 1.
  Un mobile se déplace aléatoirement sur les sommets de ce carré selon le protocole suivant :
  - × Au départ, c'est à dire à l'instant 0, le mobile est sur le sommet 1.
  - × Lorsque le mobile est à un instant donné sur un sommet, il se déplace à l'instant suivant sur l'un quelconque des trois autres sommets, et ceci de façon équiprobable.

  Pour tout $n \in \mathbb{N}$, on note X_n la variable aléatoire égale au numéro du sommet sur lequel se situe le mobile à l'instant n. D'après le premier des deux points précédents, on a donc $X_0 = 1$.
- Pour tout n de $\mathbb{N}$, on considère la matrice-ligne de $\mathcal{M}_{1,4}(\mathbb{R})$:

$$U_n = \begin{pmatrix} \mathbb{P}([X_n = 1]) & \mathbb{P}([X_n = 2]) & \mathbb{P}([X_n = 3]) & \mathbb{P}([X_n = 4]) \end{pmatrix}$$

- On montre que, si l'on pose $A = \dfrac{1}{3}\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$, on a :

$$\forall n \in \mathbb{N},\ U_{n+1} = U_n\ A$$

a) Compléter le script **Scilab** suivant pour qu'il affiche les 100 premières positions autres que celle d'origine, du mobile dont le voyage est étudié dans ce problème, ainsi que le nombre n de fois où il est revenu sur le sommet numéroté 1 au cours de ses 100 premiers déplacements (on pourra utiliser la commande `sum`).

```
1  A = [---] / 3
2  x = grand(100,'markov',A,1)
3  n = ---
4  disp(x)
5  disp(n)
```

Démonstration.

- On commence par écrire la matrice A.

```
1  A = [0, 1, 1, 1; 1, 0, 1, 1; 1, 1, 0, 1; 1, 1, 1, 0] / 3
```

- La commande `grand(100,'markov',A,1)` renvoie une trajectoire possible du mobile de taille 100. C'est une matrice ligne $[x_1, \dots, x_{100}]$ où les éléments $x_1 \in X_1(\Omega)$, $\dots$, $x_{100} \in X_{100}(\Omega)$ sont des sommets visités par le mobile dans un parcours de taille 100.
  Cette matrice est stockée dans la variable `x`.
- On complète le deuxième trou du programme par l'instruction :

```
3  n = sum(x == 1)
```

  Détaillons brièvement :
  - l'instruction `x == 1` est un test d'égalité effectué terme à terme.
    Plus précisément, cette commande permet le calcul de la matrice ligne de taille 100 correspondant à :
    $$[x_1 == 1, \dots, x_{100} == 1]$$
    (on teste si chaque élément de la matrice `x` *vaut* 1*)*
    Le résultat obtenu est une matrice ligne de booléens.
  - la commande `sum` permet de sommer tous les éléments de la matrice ligne de booléens précédents. Lors de cette somme, les booléens `True` sont convertis en l'entier 1 et les booléens `False` sont convertis en l'entier 0. Ainsi, `sum(x == 1)` permet de compter le nombre de coefficients qui sont égaux à 1 de la matrice `x`.

Commentaire

- Un tel niveau d'explication n'est pas attendu aux concours : l'écriture des lignes manquantes démontre la compréhension de toutes les commandes en question.
  On décrit ici de manière précise les instructions afin d'aider le lecteur un peu moins habile en **Scilab**.
- La ligne 3 est une manière de procéder spécifique à **Scilab**.
  On tire notamment profit de tous les outils prédéfinis du langage permettant la manipulation simple et efficace (en temps de calcul) de matrices, qui, rappelons-le, est l'objet de base de ce langage.
- Cependant, on pouvait opter pour une version plus générale, pouvant s'adapter plus facilement aux autres langages. Pour ce faire, on remplace la ligne 3 par les instructions suivantes :

```
3  n = 0
4  for i = 1:100
5      if x(i) == 1 then
6          n = n+1
7      end
8  end
```

Ces instructions permettent, à l'aide d'une structure itérative (boucle `for`) de tester un à un les éléments de `x`. Si le coefficient testé vaut 1, on incrémente un compteur `n` initialisé à 0. Ainsi, on compte le nombre d'éléments de `x` qui sont égaux à 1.
Cette version, bien que moins idiomatique, est évidemment acceptée aux concours.

□

b) Après avoir exécuté cinq fois ce script, les réponses concernant le nombre de fois où le mobile est revenu sur le sommet 1 sont : $n = 23$, $n = 28$, $n = 23$, $n = 25$, $n = 26$.
En quoi est-ce normal ?

Démonstration.

- Pour tout $n \in \mathbb{N}$, notons Y_n la v.a.r. définie par :

$$Y_n = \begin{cases} 1 & \text{si } X_n = 1 \\ 0 & \text{si } X_n \neq 1 \end{cases}$$

  La v.a.r. Y_n suit une loi de Bernoulli de paramètre $p_n = \mathbb{P}([X_n = 1]) = \dfrac{1}{4} + \dfrac{3}{4}\left(-\dfrac{1}{3}\right)^n$.

- On remarque alors :
  - pour $n = 1$, $p_0 = 0$,
  - pour $n \geqslant 2$, $p_n \simeq \dfrac{1}{4}$. En effet :

$$\frac{3}{4}\left(-\frac{1}{3}\right)^2 = \frac{3}{4}\,\frac{1}{3^2} = \frac{1}{12} \simeq \frac{1}{10} = 0,1$$

$$\frac{3}{4}\left(-\frac{1}{3}\right)^3 = \frac{3}{4}\,\frac{-1}{3^3} = -\frac{1}{36} \simeq -\frac{3}{100} = -0,03$$

$$\frac{3}{4}\left(-\frac{1}{3}\right)^4 = \frac{3}{4}\,\frac{1}{3^4} = -\frac{1}{108} \simeq \frac{1}{100} = 0,01$$

et pour $n \geqslant 5$, $\left|\dfrac{3}{4}\ \left(-\dfrac{1}{3}\right)^n\right| = \dfrac{1}{4\ 3^{n-1}} < 0,01$.

Ainsi, pour $n \geqslant 2$, on peut considérer que $Y_n \hookrightarrow \mathcal{B}\left(\frac{1}{4}\right)$ et que $(Y_2, \ldots, Y_{100})$ est un 99-échantillon d'une v.a.r. Y qui suit la loi de Bernoulli de paramètre $\frac{1}{4}$.

- La matrice obtenue par l'instruction `x == 1` est une simulation informatique de cet échantillon. Autrement dit, cette matrice s'écrit $[y_2, \ldots, y_{100}]$ où, pour tout $i \in [\![2, 100]\!]$, $y_i \in Y_i(\Omega)$.

  En vertu de la loi (faible) des grands nombres :

$$\frac{1}{99}\sum_{k=2}^{100} y_k \ \simeq\ \mathbb{E}(Y) = \mathbb{P}([Y=1]) = \frac{1}{4} \quad \text{et ainsi} \quad \sum_{k=2}^{100} y_k \ \simeq\ \frac{99}{4} \simeq 25$$

- On conclut en remarquant que $\sum_{k=1}^{100} y_k = \sum_{k=2}^{100} y_k$ car $y_1 = 0$ (Y_1 est constante égale à 0).

Il est donc normal que le mobile revienne environ 25 fois en position 1.

Commentaire

- Nous avons essayé ici de formaliser la réponse à cette question. Cependant, la formulation de la question (« En quoi est-ce normal ? ») laisse penser qu'une démonstration moins formelle serait acceptée.
- L'idée est alors de dire qu'à chaque étape (hormis la 1ère), le mobile a environ 1 chance sur 4 de se retrouver en position 1. Donc en 99 étapes, il est passé par la position 1 environ $99 \times \dfrac{1}{4} \simeq 25$ fois.

□