TP 3-Simulation de lois

Durée : 3-4h

1 Simulations de lois avec les commandes `rand` et `grand`

1.1 Simulation avec la commande `rand`

1.1.1 La commande `rand`

La commande `rand`

La commande `rand` permet de générer des nombres pseudo-aléatoires. Si m et n sont des entiers naturels non nuls

```
rand(m,n)
```

renvoie une matrice de taille $m \times n$ dont les coefficients sont des nombres choisis aléatoirement selon une loi $\mathscr{U}([0,1])$.
La commande

```
rand()
```

renvoie un nombre choisi aléatoirement selon une loi $\mathscr{U}([0,1])$.

1.1.2 Loi de Bernoulli

Soit $p \in]0,1[$.

- Rappeler la loi d'une variable aléatoire suivant une loi de Bernoulli de paramètre p.

  Voir cours

- Rappeler l'expérience de référence définissant une loi de Bernoulli de paramètre p.

  Voir cours

On considère la fonction suivante :

```
function X=bernoulli(p)
    r = rand()
    if r<p then
        X=1
    else
        X=0
    end
endfunction
```

- Copier ce programme dans le fichier `.sci` et tester la fonction avec plusieurs valeurs de $p \in]0,1[$. Quelles sont les valeurs renvoyées ? Avec quelle probabilité ? En déduire que la fonction `bernoulli` simule une variable aléatoire suivant une loi que l'on précisera.

  `bernoulli(p)` prend les valeurs 0 et 1 avec probabilité $1-p$ et p respectivement. Cette fonction simule une variable aléatoire suivant une loi de Bernoulli de paramètre p.

- Écrire une fonction `echantillonBernoulli` qui prend en argument un entier `N`, un réel `p` dans $]0,1[$ et qui renvoie un vecteur ligne contenant le résultat de la simulation de `N` variables aléatoires indépendantes suivant la loi $\mathscr{B}(p)$.

```
function X=echantillonBernoulli(N,p)
    X = ones(1,N)
    for i = 1:N
        X(i) = bernoulli(p)
    end
endfunction
```

- Tester `echantillonBernoulli(10000,0.7)` puis avec la commande `tabul` déterminer le nombre d'apparition de chaque valeur. Est-ce cohérent ?

```
X=tabul(echantillonBernoulli(10000,0.7))
```

On obtient environ 7000 fois le nombre 1 et environ 3000 fois le nombre 0. Ceci cohérent car la proportion de 1 tend à s'approcher de p quand la taille de l'échantillon tend vers $+\infty$.

- Tracer le diagramme en bâtons du résultat de la simulation de 10000 variables aléatoires indépendantes suivant la loi $\mathscr{B}(0.7)$. (on utilisera la commande `bar` vue au TP2).

```
bar(X(:,1),X(:,2))
```

1.1.3 Loi binomiale

Soient $p \in]0,1[$ et $n \in \mathbb{N}^*$.

- Rappeler la loi d'une variable aléatoire suivant une loi binomiale de paramètres n et p.

Voir cours

- Rappeler l'expérience de référence définissant une loi binomiale de paramètres n et p.

Voir cours

- Écrire une fonction `binomiale` qui prend en paramètre un entier `n`, un réel `p` dans $]0,1[$ et qui renvoie le résultat d'une simulation d'une variable aléatoire suivant un loi $\mathscr{B}(n,p)$. On pourra utiliser la fonction `echantillonBernoulli` et la commande `sum`.

```
function X=binomiale(n,p)
    X=sum(echantillonBernoulli(n,p))
endfunction
```

1.2 Simulation avec la commande `grand`

On rappelle la syntaxe de la commande `grand` que nous avons déjà rencontrée.

La commande `grand`

La commande `grand` permet de générer des nombres pseudo-aléatoires selon une loi au choix de l'utilisateur. Si m et n sont des entiers naturels non nuls

```
grand(m,n,'loi',parametre1,parametre2)
```

génère une matrice de taille $m \times n$ dont les coefficients sont des nombres choisis aléatoirement selon une loi `loi` de paramètre `parametre1`, `parametre2`.

⚠ Le paramètre `loi` est une abréviation de 3 lettres propre à Scilab. On consultera l'aide pour déterminer l'abréviation correspondant à chacune des lois usuelles.

⚠ Les paramètres `parametre1` (resp. `parametre1`, `parametre2`) est (resp. sont) le(s) paramètre(s) de la loi `loi`. Pour plus de précision, on consultera l'aide.

Exemple 1

- *`grand(m,n,"uin",n1,n2)` simule une variable aléatoire suivant la loi uniforme discrète sur* $[\![n1, n2]\!]$.
- *`grand(m,n,"bin",N,p)` simule une variable aléatoire suivant la binomiale de paramètres* (N, p).
- *`grand(m,n,"geom",p)` simule une variable aléatoire suivant la loi géométrique de paramètre* p.
- *`grand(m,n,"poi",lambda)` simule une variable aléatoire suivant la loi Poisson de paramètre* `lambda`.
- *`grand(m,n,"unf",a,b)` simule une variable aléatoire suivant la loi uniforme sur* $[a,b[$.
- *`grand(m,n,"exp",1/lambda)` simule une variable aléatoire suivant la exponentielle de paramètre* `lambda`.
- *`grand(m,n,"nor",m,sigma)` simule une variable aléatoire suivant la loi normale de paramètres* $(m, \texttt{sigma}^2)$.

2 Simulation de lois géométriques

2.1 Distribution théorique

Soient $p \in]0,1[$.

► Rappeler la loi d'une variable aléatoire suivant une loi géométrique de paramètre p.

Voir cours

Recopier le programme suivant :

```
function distTheo(p,N)
    x = 1:N
    y = (1-p)^(x-1)*p
    bar(x+0.5, y, width=0.4,'red')
endfunction
```

► Que contient la variable y ? Que représente-elle pour une variable $X \hookrightarrow \mathscr{G}(p)$?

La variable y est le vecteur ligne à N colonnes défini par

$$y = \left((p(1-p)^0, p(1-p)^1, \ldots, p(1-p)^{N-1}\right).$$

Autrement dit

$$y = (P(X=1), P(X=2), \ldots, P(X=N)).$$

2.2 Simulation avec `grand`

► En utilisant `grand`, écrire un programme qui demande à l'utilisateur d'entrer un entier naturel `N` et un réel `p` dans $]0,1[$ et qui renvoie un vecteur ligne contenant le résultat de la simulation de `N` variables aléatoires indépendantes suivant la loi $\mathscr{G}(p)$.

► A l'aide des commandes `tabul` et `bar`, compléter le programme pour qu'il affiche le diagramme en bâtons des effectifs.

► Modifier le programme pour qu'il affiche le diagramme en bâtons des fréquences.

```
p=input("Entrez la valeur du parametre p:")
N=input("Entrez la valeur de N:")
Obs = grand(1, N, "geom", p)
v=tabul(Obs)
bar(v(:,1), v(:,2)/N,width=0.2)
```

► Pour `p=0.2` et `N=10000`, comparer le diagramme en bâtons ainsi obtenu avec la distribution théorique `distTheo(0.2,20)`.

```
distTheo(0.2,20)
```

Voir figure 1.

2.3 Simulation à l'aide d'une loi de Bernoulli

► Rappeler l'expérience de référence définissant une loi géométrique de paramètre p.

Voir cours

On considère la fonction suivante :

```
function rang=geometrique(p)
    rang = 1
    aux=bernoulli(p)
    while aux==0
        rang=rang+1
        aux=bernoulli(p)
    end
endfunction
```

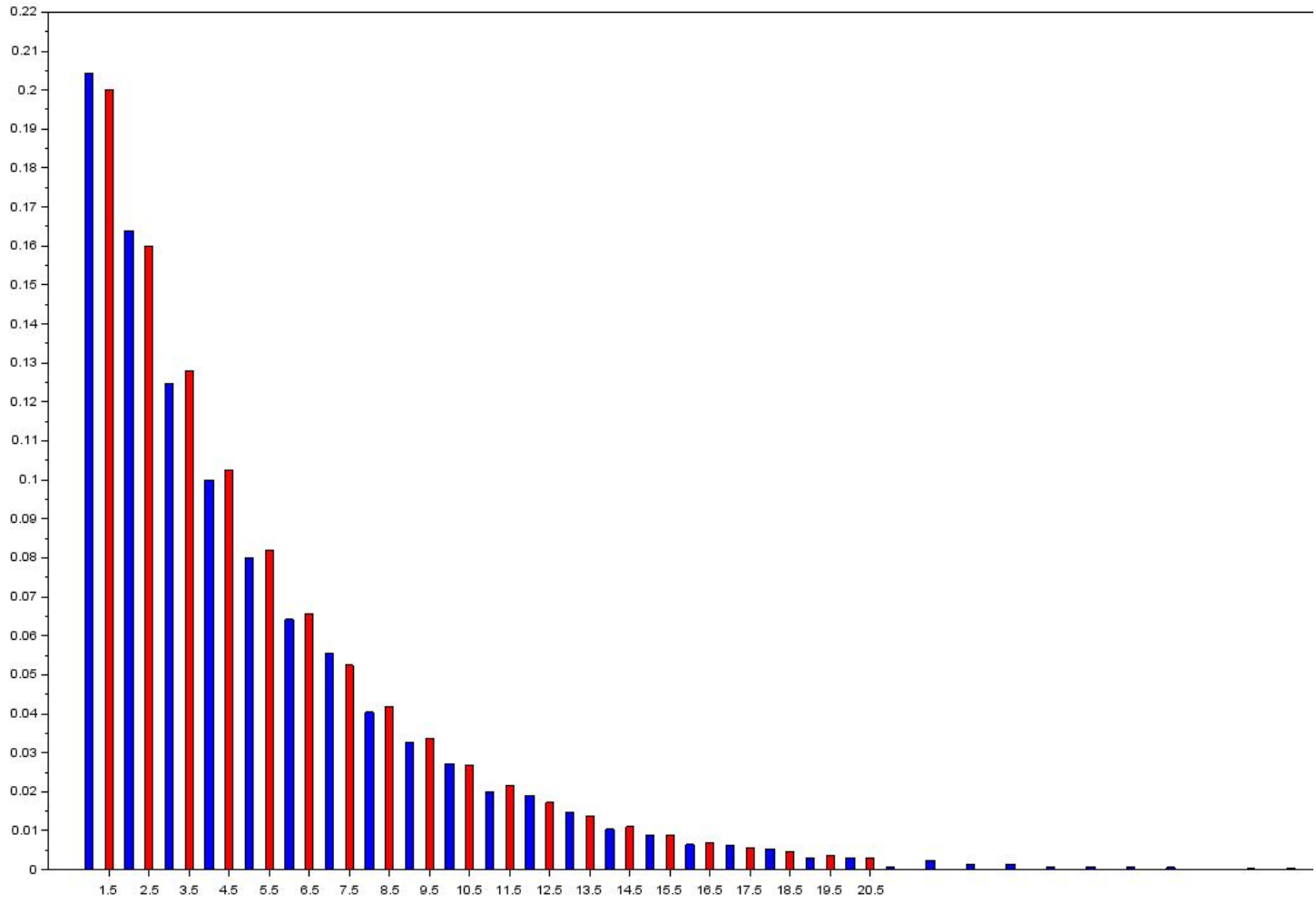

FIGURE 1 – En rouge, la distribution théorique, en bleu la distribution expérimentale obtenue avec `grand` pour une loi géométrique de paramètre 0.2

- Copier ce programme dans le fichier `.sci` et tester la fonction avec plusieurs valeurs de $p \in]0,1[$. Justifier que la fonction `geometrique` simule une variable aléatoire suivant une loi que l'on précisera.

> La variable `rang` est initialisée à 1 et la variable `aux` est le résultat d'une épreuve de Bernoulli de paramètre `p`. Si cette épreuve est un échec, c'est-à-dire si `aux` vaut 0 alors on incrémente `rang` de 1 et on réalise une nouvelle épreuve de Bernoulli de paramètre `p`. Le programme termine dès qu'on a réalisé une épreuve pour laquelle `aux` vaut 1 et renvoie alors la valeur de `rang`. Ainsi, cette fonction renvoie le rang du premier succès lorsqu'on réalise une succession d'épreuves de Bernoulli indépendantes de paramètre `p`. La fonction simule donc une variable aléatoire suivant une loi géométrique de paramètre `p`.

- Écrire une fonction `echantillonGeometrique` qui prend en argument un entier `N`, un réel `p` dans $]0,1[$ et qui renvoie un vecteur ligne contenant le résultat de la simulation de `N` variables aléatoires indépendantes suivant la loi $\mathscr{G}(p)$.

```
function X=echantillonGeometrique(N,p)
    X = ones(1,N)
    for i = 1:N
        X(i) = geometrique(p)
    end
endfunction
```

- Tracer le diagramme en bâtons des fréquences du résultat de la simulation de 10000 variables aléatoires indépendantes suivant la loi $\mathscr{G}(0.2)$. (on utilisera la commande `bar`).
- Comparer ce diagramme avec le diagramme du paragraphe précédent et avec la distribution théorique.

```
Obs2 =tabul(echantillonGeometrique(10000,0.2))
scf(1);bar(Obs2(:,1), Obs2(:,2)/N,width=0.2);
distTheo(p,20)
```

Voir figure 2.

2.4 Simulation à l'aide d'une loi exponentielle

Proposition 1

Soient $\theta > 0$ et $X \hookrightarrow \mathscr{E}(1)$. Alors $\lceil \theta X \rceil$ suit une loi géométrique de paramètre $1 - e^{-\frac{1}{\theta}}$.

Ici $\lceil x \rceil$ désigne la partie entière supérieure de x c'est-à-dire

$$\lceil x \rceil = n \Longleftrightarrow n - 1 < x \leqslant n.$$

Soit $\lambda > 0$.

- Rappeler la densité d'une variable aléatoire suivant une loi exponentielle de paramètre λ.

> Voir cours

Recopiez le programme suivant à la suite des vos précédents programmes (dans le fichier .sce)

```
theta=1/log(1.25)

Obs3=ceil(theta*grand(1,N, "exp",1))
v3=tabul(Obs3)
bar(v3(:,1), v3(:,2)/N,width=0.2,'green')
```

- Quel est le rôle de la commande `ceil`? On pourra consulter l'aide ou tester `ceil(x)` pour différent nombres réels `x`.

> La commande `ceil` donne la partie entière supérieure d'un nombre.

- Expliquer pourquoi `Obs3` est la simulation de `N` variables aléatoires indépendantes suivant une loi géométrique dont on précisera le paramètre.

> La commande `grand(1,N, "exp",1)` simule `N` variables aléatoires indépendantes suivant une loi exponentielle de paramètre 1, donc d'après la proposition 1, `Obs3` simule `N` variables aléatoires indépendantes suivant une loi géométrique de paramètre $1 - e^{-\frac{1}{\texttt{theta}}} = 0.2$.

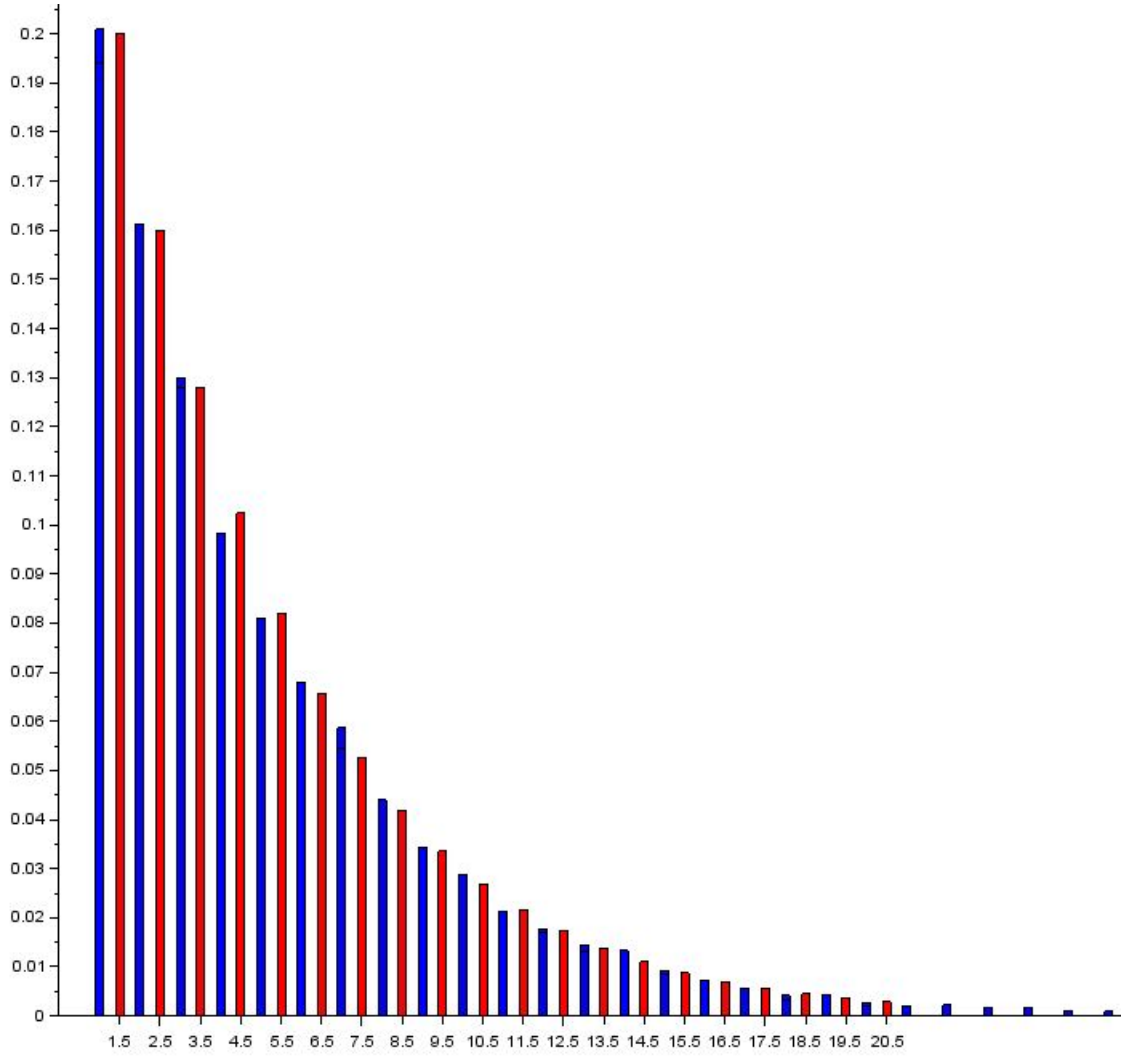

FIGURE 2 – En rouge, la distribution théorique, en bleu la distribution expérimentale obtenue avec `echantillonGeometrique` pour une loi géométrique de paramètre 0.2

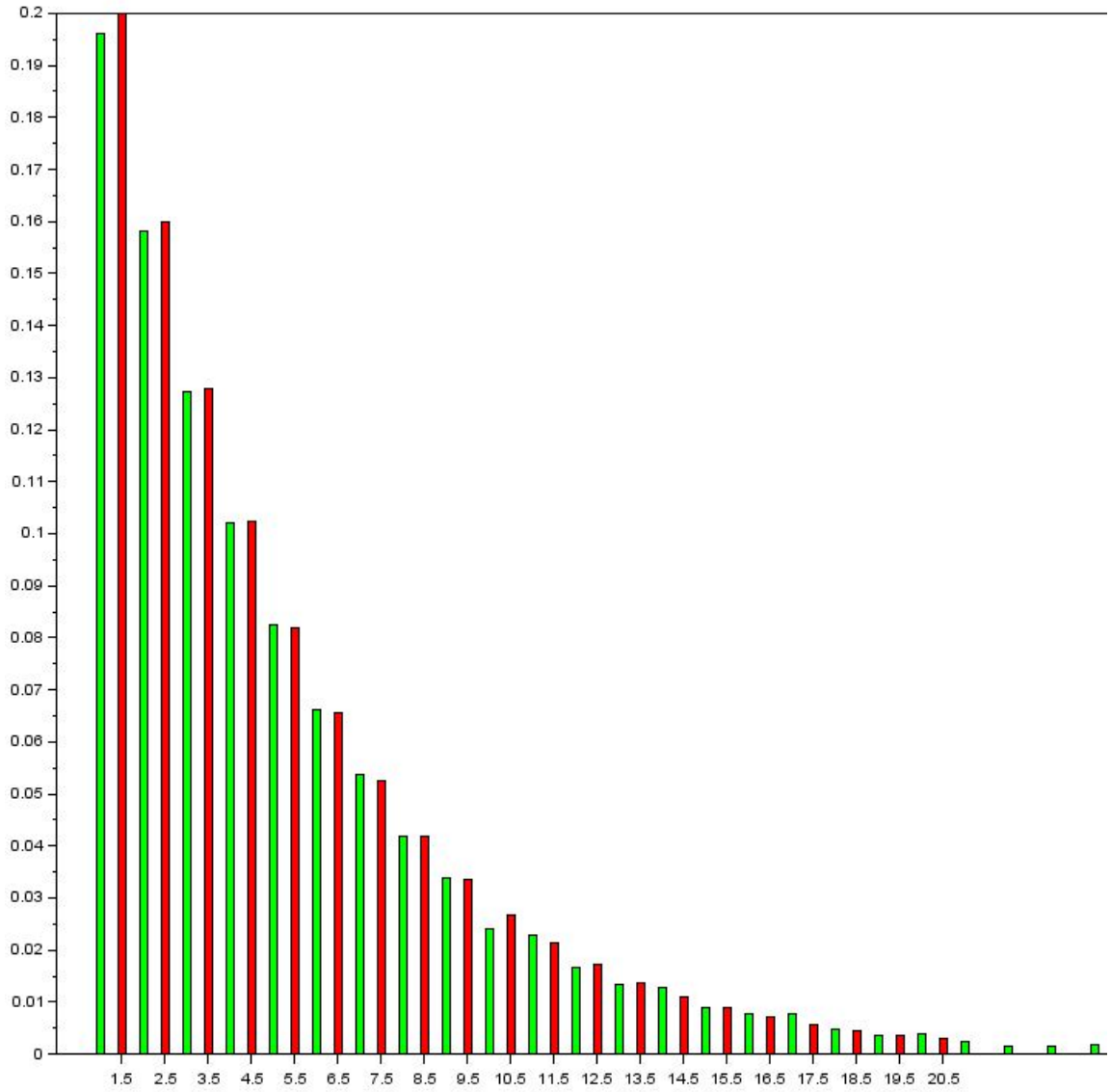

FIGURE 3 – En rouge, la distribution théorique, en vert la distribution expérimentale obtenue avec une loi exponentielle pour une loi géométrique de paramètre 0.2

► Comparer le diagramme en bâtons obtenu avec les diagrammes précédents.

```
scf(1);bar(v3(:,1), v3(:,2)/N,width=0.2,'green')
distTheo(0.2,20)
```

Voir figure 3.

3 Lois uniformes

3.1 Méthode d'inversion

Théorème (théorème d'inversion)

Soit X une variable aléatoire réelle de fonction de répartition $F : \mathbb{R} \longrightarrow [0,1]$. On définit une fonction G sur $]0,1[$ par

$$\forall y \in]0,1[,\ G(y) = \inf\{x \in \mathbb{R} \mid F(x) > y\}.$$

Soit U une variable aléatoire réelle suivant une loi $\mathscr{U}(]0,1[)$.
Alors, G(U) est une variable aléatoire dont la fonction de répartition est F.

Grâce à ce résultat, dès lors que l'on sait simuler une loi uniforme, on va pouvoir simuler n'importe quelle loi dont on connaît la fonction de répartition.

3.2 Simulation d'une loi exponentielle

Soit $\lambda > 0$.

- Soit X une variable aléatoire suivant une loi exponentielle de paramètre λ. Calculer sa fonction de répartition F

En notant f la densité de X.

$$F(x) = P([X \leqslant x]) = \int_{-\infty}^{x} f(t)dt = \begin{cases} \int_{-\infty}^{x} 0\,dt & \text{si } x < 0, \\ \int_{0}^{x} \lambda e^{-\lambda t}\,dt & \text{si } x \geqslant 0. \end{cases} \quad \text{car } \forall t < 0,\ f(t) = 0$$

$$= \begin{cases} 0 & \text{si } x < 0, \\ 1 - e^{-\lambda x} & \text{si } x \geqslant 0. \end{cases}$$

- Calculer la fonction G associée à F et définie dans l'énoncé du théorème.

Soit $y \in]0,1[$. Si $x < 0$ alors $F(x) \leqslant y$. Soit $x \geqslant 0$, alors

$$\begin{aligned} F(x) > y &\iff 1 - e^{-\lambda x} > y \iff 1 - y > e^{-\lambda x} \\ &\iff \ln(1-y) > -\lambda x \\ &\iff -\frac{\ln(1-y)}{\lambda} < x. \end{aligned}$$

Ainsi, comme $-\frac{\ln(1-y)}{\lambda} \geqslant 0$ on a

$$G(y) = \inf\left\{x \geqslant 0 \mid -\frac{\ln(1-y)}{\lambda} < x\right\} = -\frac{\ln(1-y)}{\lambda}.$$

- En déduire une simulation Scilab d'une variable aléatoire X suivant une loi exponentielle. Plus précisément, écrire une fonction `expo` qui prend en paramètre un réel `lambda` (strictement positif) et qui renvoie le résultat de la simulation de X en utilisant la commande `rand`.

```
function X=expo(lambda)
    X=-log(1-rand())/lambda
endfunction
```

- Écrire un programme qui demande à l'utilisateur la valeur d'un réel `lambda`, qui calcule le résultat de la simulation de 10000 variables aléatoires indépendantes suivant la loi exponentielle de paramètre `lambda` (sans l'afficher!) et qui affiche l'histogramme des valeurs obtenues avec 40 classes.
- Sur le même graphique, tracer la densité de la variable X.

```
lambda=input("Entrez la valeur du parametre lambda:")
N=10000
E=ones(1,N);
for k=1:N
    E(k)=expo(lambda);
end
x = linspace(0,max(E),100);

scf(3);plot2d(x,lambda*exp(-lambda*x),strf="000",style=5)
scf(3);histplot(40,E)
```

Voir figure 4.

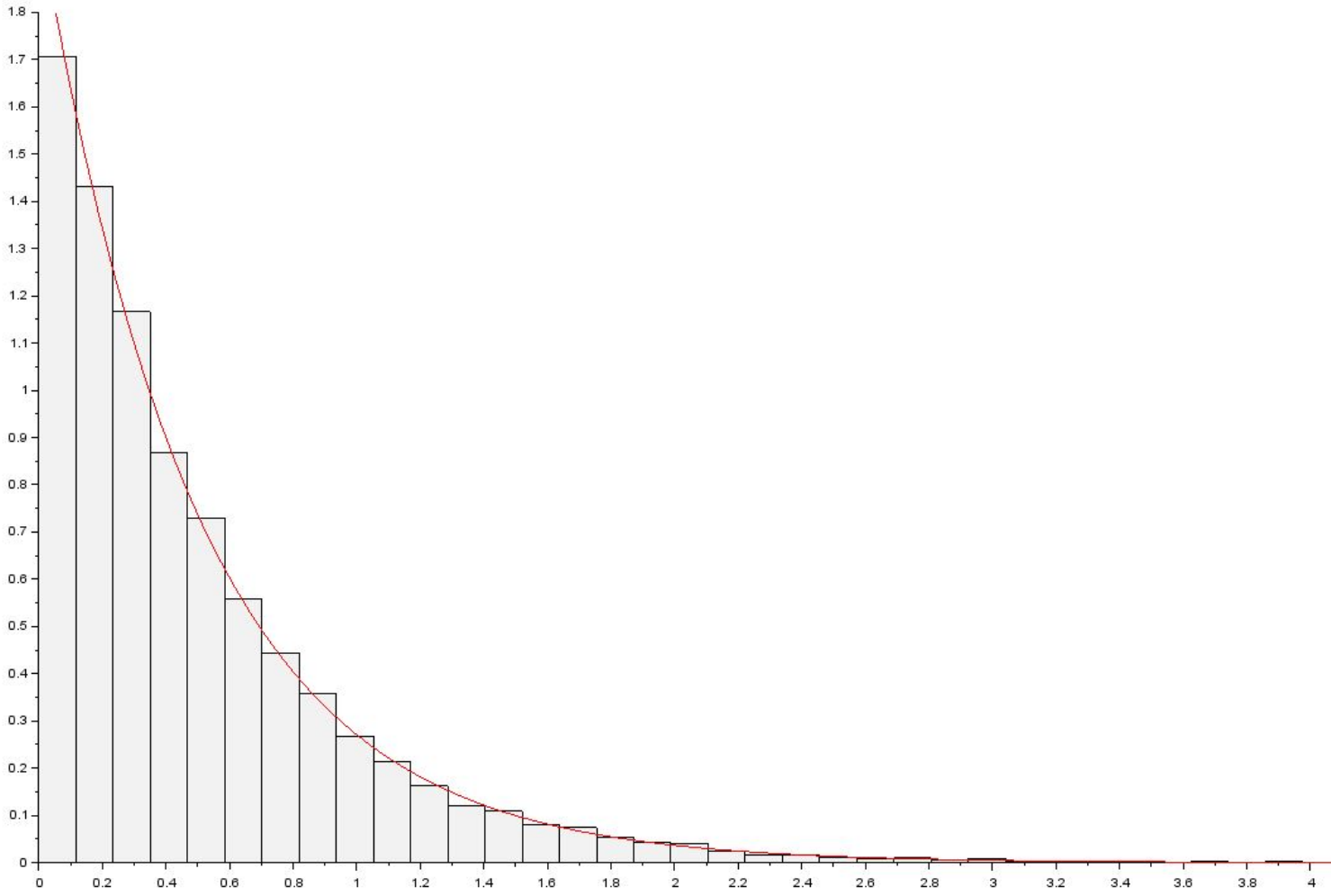

FIGURE 4 – Histogramme obtenu pour `lambda`= 2 ; en rouge la densité d'une loi exponentielle de paramètre 2.

3.3 Simulation de la loi uniforme sur $[a,b]$

Soient $a<b$.

- Soit X une variable aléatoire suivant une loi uniforme sur $[a,b]$. Calculer sa fonction de répartition F

$$F(x)=\begin{cases}0 & \text{si } x<a\\ \frac{x-a}{b-a} & \text{si } a\leqslant x<b\\ 1 & \text{si } x>b.\end{cases}$$

- Calculer la fonction G associée à F et définie dans l'énoncé du théorème.

Soit $y\in]0,1[$. Si $x<a$ alors $F(x)\leqslant y$. Si $x\in[a,b]$, alors

$$F(x)>y \iff \frac{x-a}{b-a}>y \iff x>y(b-a)+a.$$

Si $x>b$ alors $F(x)>y$. Ainsi,

$$G(y)=\min[b, y(b-a)+a)=y(b-a)+a.$$

- En déduire une simulation Scilab d'une variable aléatoire X suivant une loi uniforme sur $[a,b]$. Plus précisément, écrire une fonction `unif` qui prend en paramètre des réels `a` et `b` (avec $a<b$) et qui renvoie le résultat de la simulation de X en utilisant la commande `rand`.

```
function X=unif(a,b)
   X=(b-a)*rand()+a
endfunction
```

4 Simulation de variables aléatoires aux concours

Exercice 1

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, 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-ième tirage"*
- R_k *l'événement : "on obtient une boule rouge au k-ième tirage"*

1. *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
    r=2
    s=0
    for k=1:n
        x=rand()
        if ... then
            ...
        else
            ...
        end
    end
endfunction
```

où `b` désigne le nombre de boules bleues présentes dans l'urne, `r` le nombre de boules rouges présentes dans l'urne et `s` désigne le nombre de boules rouges obtenues lors des n tirages.

2. *On exécute le programme suivant :*

```
n=10
m=0
for i=1:1000
    m=m+EML(n)
end
disp(m/1000)
```

On obtient `6.657`. Comment interpréter ce résultat ?

Exercice 2

Soit p un réel de $]0;1[$.
Deux individus A *et* B *s'affrontent dans un jeu de Pile ou Face dont les règles sont les suivantes :*

- *le joueur* A *dispose d'une pièce amenant Pile avec la probabilité* $\frac{2}{3}$ *et lance cette pièce jusqu'à l'obtention du deuxième Pile ; on note* X *la variable aléatoire prenant la valeur du nombre de Face alors obtenus ;*
- *le joueur* B *dispose d'une autre pièce amenant Pile avec la probabilité p et lance cette pièce jusqu'à l'obtention d'un Pile ; on note* Y *la variable aléatoire prenant la valeur du nombre de Face alors obtenus ;*
- *Le joueur* A *gagne si son nombre de Face obtenus est inférieur ou égal à celui de* B *; sinon c'est le joueur* B *qui gagne.*

On dit que le jeu est équilibré lorsque les joueurs A *et* B *ont la même probabilité de gagner.*

1. Simulation informatique

(a) Écrire une fonction Scilab d'en-tête `function x = simule_X()` *qui simule la variable aléatoire* X.

(b) On suppose que l'on dispose d'une fonction `simule_Y` *qui, prenant en argument un réel p de* $]0;1[$, *simule la variable aléatoire* Y. *Expliquer ce que renvoie la fonction suivante :*

```
1.  function r = mystere(p)
2.        r = 0
3.        N = 10^4
4.        for k = 1:N
5.                  x = simule_X()
6.                  y = simule_Y(p)
7.                  if x <= y then
8.                            r = r + 1/N
9.                  end
10.       end
11.  endfunction
```

5 Éléments du programme officiel

1. Compétences mises en jeu :
   - C1 : produit et interpréter des résumées numériques et graphiques d'une série statistiques ou d'une loi.
   - C2 : Modéliser et simuler des phénomènes (aléatoires et déterministes) et les traduire en langage mathématique.
   - C6 : Porter un regard critique sur les méthodes d'estimation et de simulation.
2. Exigibles de première année :
   - `bar`, `histplot`, `grand`, `rand`
   - utilisation des boucles `while` et `for`