TP1 - Simulation de variables aléatoires

Mots clés :

- **Scilab**
- **Générateur aléatoire**
- `rand`
- **Loi exponentielle**
- **Loi discrète**
- **Loi normale**
- **Loi uniforme**
- **Méthode de Box-muller**
- **Méthode du rejet**

Élève : Boudjeltia Reda
M2 MIMSE Spé 3 AD
Date : 11 Octobre 2015

Table des matières

1 Test du générateur aléatoire

Pour faire nos essaies nous avons prit 50 classes. On s'aperçoit que la courbe converge vers 1 quand n devient très grand.

```
rand("seed",0)
test = rand

class = 50;

n = 1000;
xx = rand(n,1);
clf();
//red
histplot(class, xx, style=6);
legend(["1000"]);
```

Figure 1 – Code pour une simulation

Figure 2 – Échantillon de rand pour N allant de 10 à 1 000 000

2 Simulation d'une loi exponentielle

Soit U une variable aléatoire de loi uniforme.
On pose

$$Z = -\alpha.ln(U) \tag{1}$$

On a

$$\begin{aligned} \mathbb{P}(Z \leq t) &= \mathbb{P}(-\alpha \dot{\ln}(U) \leq t) && (2) \\ &= \mathbb{P}(U \leq e^{-\frac{t}{\alpha}}) && (3) \end{aligned}$$

Or

$$\begin{aligned} F_Z(t) &= \mathbb{P}(Z \leq t) && (4) \\ & && (5) \end{aligned}$$

Donc

$$p_Z(t) = \frac{\partial F_Z(t)}{\partial t} = -\frac{1}{\alpha}.e^{\frac{-t}{\alpha}} \tag{6}$$

On obtient donc la densité d'une variable aléatoire densité identique à celle de la loi exponentielle qui est de la forme

$$p_Z(t) = -\alpha.e^{-\lambda.t} \tag{7}$$

Par conséquent Z suit une loi exponentielle de paramètre $-\frac{1}{\alpha}$

Après avoir écrit la fonction `myExp` nous avons lancé cinq simulation avec des tailles d'échantillon N différent. Ainsi comme le montre les relevés ci-dessous nous avons une convergence assez rapide car à 10^2 la courbe se dessine selon la courbe de la fonction $f(x) : x \to -\frac{1}{\alpha}.e^{x/\alpha}$

```
deff("[p]=myexp(n,␣a)", "p=-1/a*log(rand(n,1))");
```

FIGURE 3 – Définition de notre fonction exponentielle

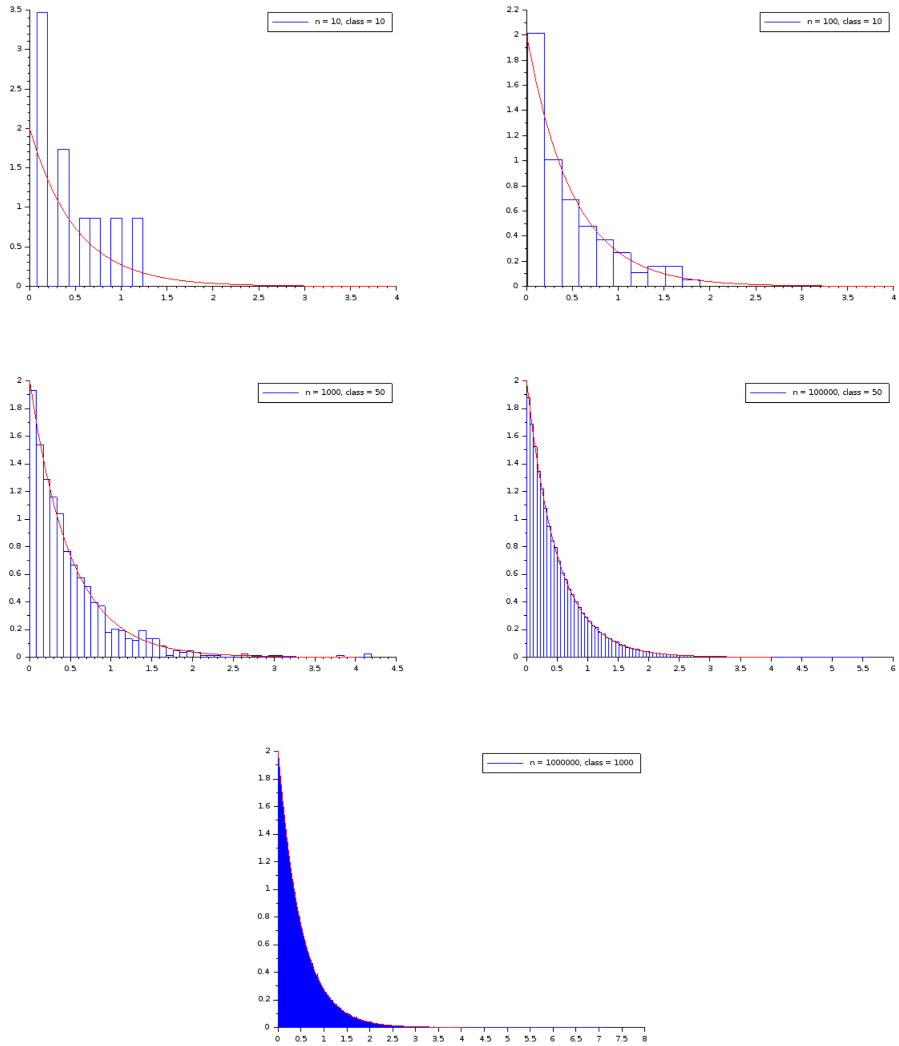

Figure 4 – Échantillon de myExp pour N allant de 10 à 1 000 000

3 Simulation de lois discrètes

```
function [y] = discrete(x,p)
    rand("uniform");
    d = length(p);
    pp = [0 p(1:(d-1))];
    cpp = cumsum(pp);
    cp = cumsum(p);
    U = rand(1,1);
    k = find((cpp <= U) & (U<cp));
    y = x(k);
endfunction

function [y] = simudsc(n, x, p)
    for j = 1:n
        y(j) = discrete(x,p);
    end
endfunction
```

Figure 5 – Fonction d'une loi discrète sur le vecteur x avec probabilités le vecteur p

Nous avons fait nos essaies avec x = [-3, -1.5, 0, 2, 2.5] et p = [0.2, 0.3, 0.1, 0.3, 0.1]

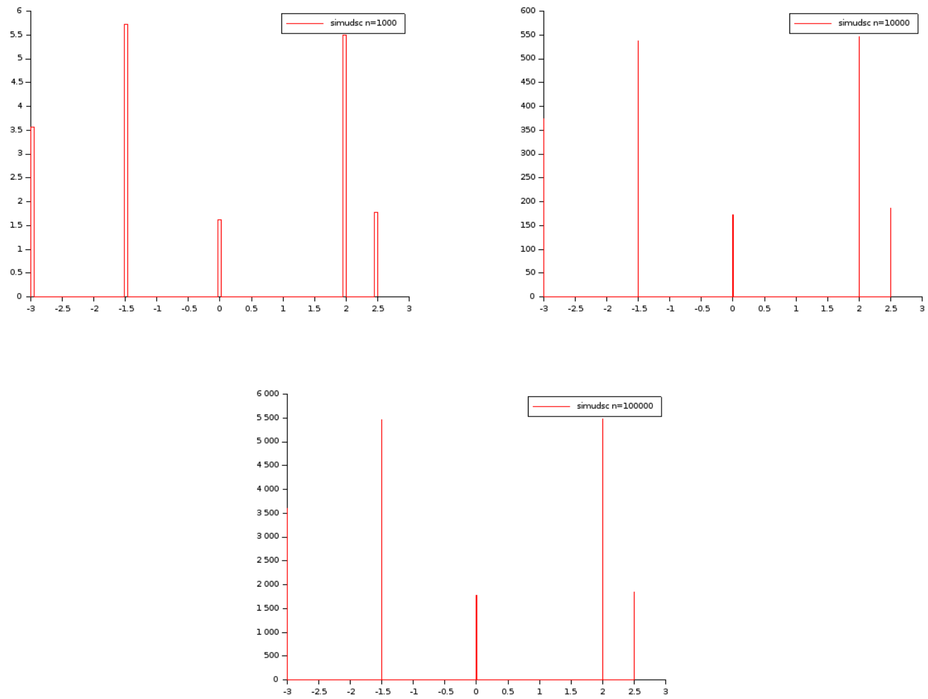

Figure 6 – Échantillon de X pour N allant de 1000 à 100 000

A partir des figures ci dessus on peut constater que les valeurs sont bien -3, -1.5, 0, 2, et 2.5.

Ainsi, si on considère que 5500 représente 30% de notre population. Par produits en croix, nous devons retrouver 20% et 10% pour les autres valeurs.

```
-->1.5*0.3/5.5
 ans  =

    0.0818182

-->3.5*0.3/5.5
 ans  =

    0.1909091
```

Figure 7 – Vérification de nos simulations

Au final, nous retrouvons bien la répartitions voulu.

4 Simulation d'une loi normale

4.1 Méthode de Box-Muller

Afin d'obtenir un échantillon d'une loi uniforme sur $[a, b]$ à partir d'une loi uniforme sur $[0, 1]$ (fonction `rand`). On pose $Z = \texttt{rand}(b - a) + a$

Ensuite, on a $X = \sqrt{R}.cos(\Theta)$ avec R qui suit une loi uniforme sur $[0, 2.\pi]$ et R une loi exponentielle de paramètre $\frac{1}{2}$.

```
function [X] = BM(n)
    theta = rand(n,1)*2*%pi+0;
    R = myexp(n,1/2);

    for i = 1:n
        X(i) = cos(theta(i))*sqrt(R(i));
    end

endfunction
```

Figure 8 – code de la fonction BM

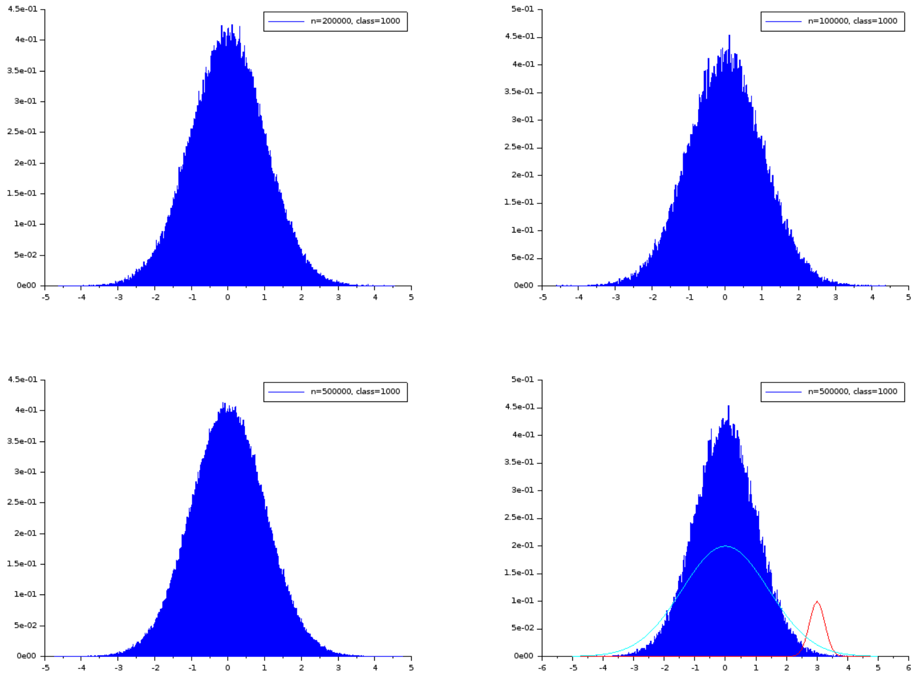

Figure 9 – Échantillon de X pour N allant de 10 à 1 000 000

4.2 Méthode du rejet

```
function [vect]=rejet1()

        u = rand(1,1)*2-1;
        v = rand(1,1)*2-1;
        r = u**2 + v**2;
        if r>1 then
            vect = rejet1();
        end

        if r>1 then
            rejet1();
        else
            z = sqrt((-2*log(r))/r);
            x = u*z;
            y = v*z;

            X = x;
            Y = y;
            //print(%io(2),X,Y,r);

            vect(1) = X
            vect(2) = Y
            vect(3) = r
        end
endfunction
```

Figure 10 – Une itération du rejet

```
function [p]=rejet(n)
    for i=1:n
        V = rejet1();
        p(1)(i) = V(1);
        p(2)(i) = V(2);
    end
endfunction
```

Figure 11 – Fonction de rejet final

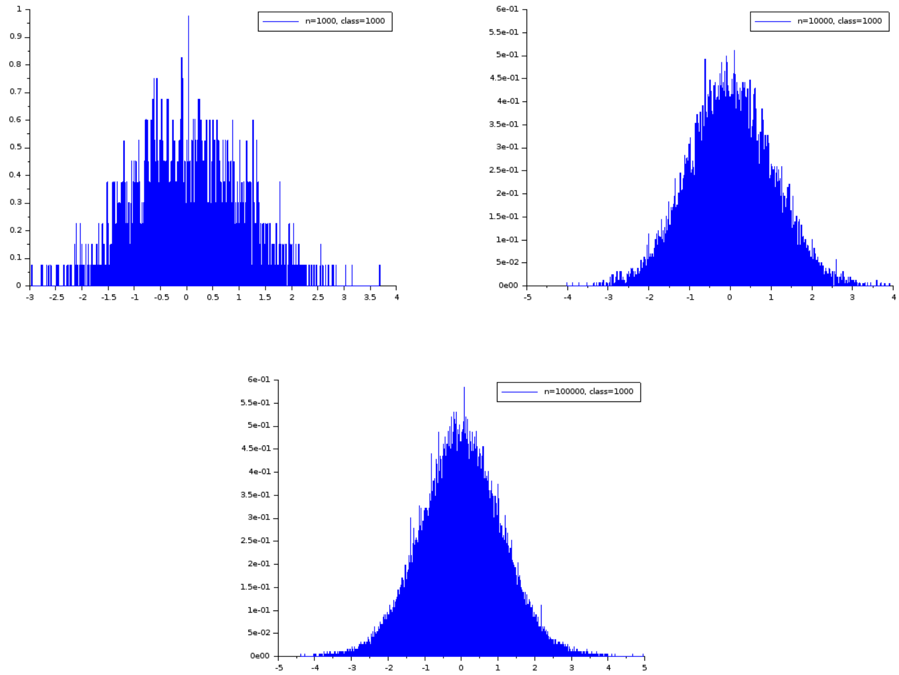

Figure 12 – Méthode du rejet pour n=1000 à 1000000

4.3 Comparaison des méthodes

Tout d'abord, comme on peu le voir sur les figures de chaque méthodes. Box-Muller donne des courbe plus rapidement des courbes plus lisse sans bruit, donc avec plus d'exactitude. Ensuite, nous avons comparé les méthodes sur les temps de calcul. On peut noté que sur un PC puissant on a attendu plus de 3 minutes pour avoir la courbes du rejet avec n = 100 000 sur une machine équipé d'un processeur `Intel Xeon` à 4 cœurs tournant à 3.4 Ghz.
Sur cette même machine nous avons testé les méthodes grâce à la fonction `timer`. A noter que `timer` renvoie le temps CPU consommé par `Scilab` depuis le dernier appel avec une précision de 100 nanosecondes.

n	Rejet	Box-Muller
100	0.004	<0.001
1 000	0.052	0.002
10 000	1.069	0.072
100 000	90.221	7.63
500 000	>200	196.025

Figure 13 – Tableau comparatif méthode Box-Muller / méthode du rejet

Au final, il nettement plus intéressant d'utiliser la méthode de Box-Muller pour sa de convergence rapide mais aussi pour sa complexité moyenne.

5 Annexe

5.1 Code complet

```
//Exercice 1
rand("seed",0)
test = rand

class = 50;

n = 1000;
xx = rand(n,1);
clf();
//red
histplot(class, xx, style=6);
legend(["1000"])

n = 10000;
yy = rand(n,1);
//green
clf();
histplot(class, yy, style=5);
legend(["10000"])

n = 100000;
zz = rand(n,1);
//grey
clf();
histplot(class, zz, style=3);
legend(["100000"])

n = 1000000;
aa = rand(n,1);
//blue
clf();
histplot(class, aa, style=2);
legend(["1000000"])


//Exercice 2

deff("[p]=myexp(n,␣a)", "p=-1/a*log(rand(n,1))");
n=1000000;
x=[0:0.1:4];
clf();
histplot(1000, myexp(n,2), style=2);
plot2d(x,2*exp(-2*x), style=5);
legend(["n␣=␣1000000,␣class␣=␣1000"])
```

Figure 14 – Code complet avec les tests

```
//Exercice 3

x1=[0,1,2,3,4,5,6,7,8];
p1=[1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9]

//
// simule une variable aléatoire discrète
// ensemble de valeurs possibles x_1,....,x_d
// avec probabilités respectives p_1,...,p_d
// x=( x_1,....,x_d), p=(p_1,...,p_d)
//
function [y] = discrete(x,p)
    rand("uniform");
    d = length(p);
    pp = [0 p(1:(d-1))];
    cpp = cumsum(pp);
    cp = cumsum(p);
    U = rand(1,1);
    k = find((cpp <= U) & (U<cp));
    y = x(k);
endfunction



x = [0,1,2,3,4];
p = [1/5, 1/5, 1/5, 1/5, 1/5];
a = discrete(x,p);


function [y] = simudsc(n, x, p)
    for j = 1:n
        y(j) = discrete(x,p);
    end
endfunction

v1 = [-3,-1.5,0,2,2.5];
p1 = [0.2,0.3,0.1,0.3,0.1];

n = 1000;
yy = simudsc(n,v1,p1);
clf();
histplot(100, yy, style=5);
legend(["simudsc␣n=1000"])
```

Figure 15 – Code complet avec les tests

```
//Exercice 4a

a = 9;
b = 10;
r = rand(1,1)*(b-a)+a;

function [X] = BM(n)
    theta = rand(n,1)*2*%pi+0;
    R = myexp(n,1/2);
    for i = 1:n
        X(i) = cos(theta(i))*sqrt(R(i));
    end
endfunction

clf();
n=100000;
x=[-5:0.1:5];
histplot(1000, BM(n), style=2);
legend(["n=500000,␣class=1000"]);
plot2d(x,1/(4*sqrt(2*%pi))*exp(-(x-3)**2/2*4**2), style=1);
plot2d(x,1/(2*sqrt(2*%pi))*exp(-x**2/4), style=1);

X = grand(n,1,"nor",3,4);
histplot(1000, X, style=4);

//Exercice 4b
function [vect]=rejet1()

        u = rand(1,1)*2-1;
        v = rand(1,1)*2-1;
        r = u**2 + v**2;
        if r>1 then
            vect = rejet1();
        end

        if r>1 then
            rejet1();
        else
            z = sqrt((-2*log(r))/r);
            x = u*z;
            y = v*z;

            X = x;
            Y = y;
            //print(%io(2),X,Y,r);

            vect(1) = X
            vect(2) = Y
            vect(3) = r
        end
endfunction
```

FIGURE 16 – Code complet avec les tests

```
function [p]=rejet(n)
    for i=1:n
        V = rejet1();
        p(1)(i) = V(1);
        p(2)(i) = V(2);
    end
endfunction

n = 100000;
histplot(1000, rejet(n), style=2);
legend(["n=100000,␣class=1000"])
```

Figure 17 – Code complet avec les tests