Simulation en biologie
Compte-rendu

Etienne JEAN

October 8, 2018

1 Exercice 2

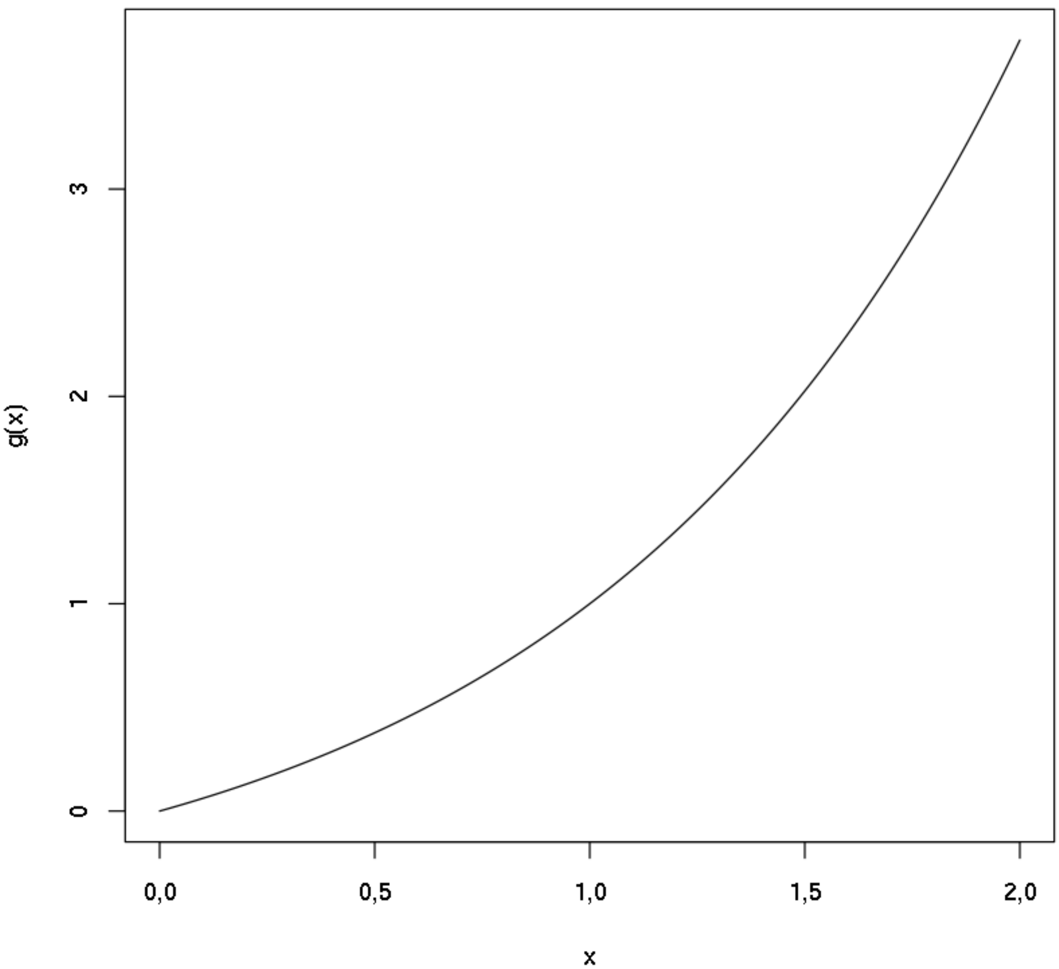
Le but de cet exercice est de trouver numériquement la valeur de l'intégrale de la fonction g. La fonction g est définie par :

```
In [1]: # g function
        g <- function(x){
            return( (exp(x)-1) / (exp(1)-1) )
        }
```

Et voici une représentation de g sur l'intervalle [0; 2]

```
In [2]: # Values to evaluate g
        valX <- seq(from=0, to=2, len=1000)
        valY = g(valX)
```

```
In [3]: # plot of g
        plot(valX, valY, xlab='x', ylab='g(x)', type='l')
```



Analytiquement, on peut trouver la valeur exacte de l'integrale de g sur [0; 2]. La fonction `integrate` de R donne également une très bonne approximation.

Preuve :

$$g(x) = \frac{exp(x) - 1}{exp(1) - 1}$$

On intègre g entre 0 et 2

$$I = \int_0^2 g(u)du$$

$$I = \int_0^2 \frac{exp(u) - 1}{exp(1) - 1}du$$

Par linéarité de l'intégrale :

$$I = \frac{1}{exp(1) - 1} \int_0^2 exp(u) - 1du$$

$$I = \frac{1}{exp(1)-1}[exp(u)-u]_0^2$$

$$I = \frac{1}{exp(1)-1}(exp(2)-2-exp(0)+0)$$

On obtient finalement :

$$I = \frac{exp(2)-3}{exp(1)-1}$$

```
In [4]: # Valeur à trouver de l'intégration
        # fonction integrate
        integrate(g, lower = 0, upper = 2)
        # valeur exacte
        Iexact = (exp(2)-3)/(exp(1)-1)
        Iexact
```

```
2,554328 with absolute error < 2,8e-14
```

2,55432841472039

```
In [5]: # 2 : MC par tirage noir ou blanc
        n = 100                                 # nombre de tirages
        nb_rep = 10000                          # nb de repetition de la methode

        list_Ibw = c()
        m = g(2)                                # majorant de g
        for (i in 1:nb_rep){
            tirageX = runif(n, min=0, max=2)    # x entre 0 et 2
            tirageY = runif(n, min=0, max=m)    # y entre 0 et m
            ns = sum(g(tirageX)-tirageY >= 0)   # nombre de tirages sous g
            Ibw = m*(2-0)*ns/n                  # valeur estimée de I
            list_Ibw = c(list_Ibw, Ibw)
        }
        Ibw = mean(list_Ibw)                    # valeur tirage noir et blanc
        Ibw
```

2,56004447545771

```
In [6]: # 3 : MC simple
        list_Isimple = c()
        for(i in 1:nb_rep){
            tirageX = runif(n, min=0, max=2)    # x entre 0 et 2
            Isimple = (2-0)/n*sum(g(tirageX))
            list_Isimple = c(list_Isimple, Isimple)
        }
        Isimple = mean(list_Isimple)
        Isimple
```

2,55262590132917

In [7]:
```
# 4 : MC suivant l'importance
alpha = 2
beta = 1
h <- function(x){
    return(g(x)/(dbeta(x/2, alpha, beta)/2))
}


list_Iimport = c()
for (i in 1:nb_rep){
    tirageX = rbeta(n, alpha, beta)*2
    Iimport = mean(h(tirageX))
    list_Iimport = c(list_Iimport, Iimport)
}

Iimport = mean(list_Iimport)
Iimport
```

2,55376079132008

In [8]:
```
# 5 : Amelioration BIS
h <- function(x, alpha, beta){
    return(g(x)/(dbeta(x/2, alpha, beta)/2))
}

fn <- function(par){
    alpha = par[1]
    beta = par[2]
    score = c()
    for(i in 1:100){
        tirageX = rbeta(n, alpha, beta)*2
        score = c(score, abs(Iexact - mean(h(tirageX, alpha, beta))))
    }
    return(mean(score))
}
```

In [9]:
```
optimal_par = suppressWarnings(optim(c(2, 1), fn))$par
```

In [10]:
```
optimal_par
```

1. 2,30815888213052 2. 0,888594799315007

In [11]:
```
alpha = optimal_par[1]
beta = optimal_par[2]
h <- function(x){
    return(g(x)/(dbeta(x/2, alpha, beta)/2))
}
```

```
list_Ioptim = c()
for (i in 1:nb_rep){
    tirageX = rbeta(n, alpha, beta)*2
    Ioptim = mean(h(tirageX))
    list_Ioptim = c(list_Ioptim, Ioptim)
}

Ioptim = mean(list_Ioptim)
Ioptim
```

2,55427783879106

```
In [12]: # 6 : Comparaison
         MSE_Iexact = function(I){
             return(mean((Iexact - I))^2)
         }
         # MC noir et blanc
         print(-log(MSE_Iexact(Ibw)))
         # MC simple
         print(-log(MSE_Iexact(Isimple)))
         # MC suivant l'importance
         print(-log(MSE_Iexact(Iimport)))
         # MC suivant l'importance optimisé
         print(-log(MSE_Iexact(Ioptim)))
```

```
[1] 10,32895
[1] 12,7513
[1] 14,9481
[1] 19,78407
```

Finalement, la précision des différentes méthodes varie beaucoup.

La méthode *MC simple* est de loin la moins précise.

De manière assez surprenante, la méthode *MC noir et blanc* est assez performante, et comparable à la méthode MC suivant l'importance, avec une loi beta de parametres alpha=2 et beta=1.

La méthode la plus précise est aussi la plus élaborée, à savoir la méthode *MC suivant l'importance*, pour laquelle les paramètres de la fonction beta ont été optimisés.

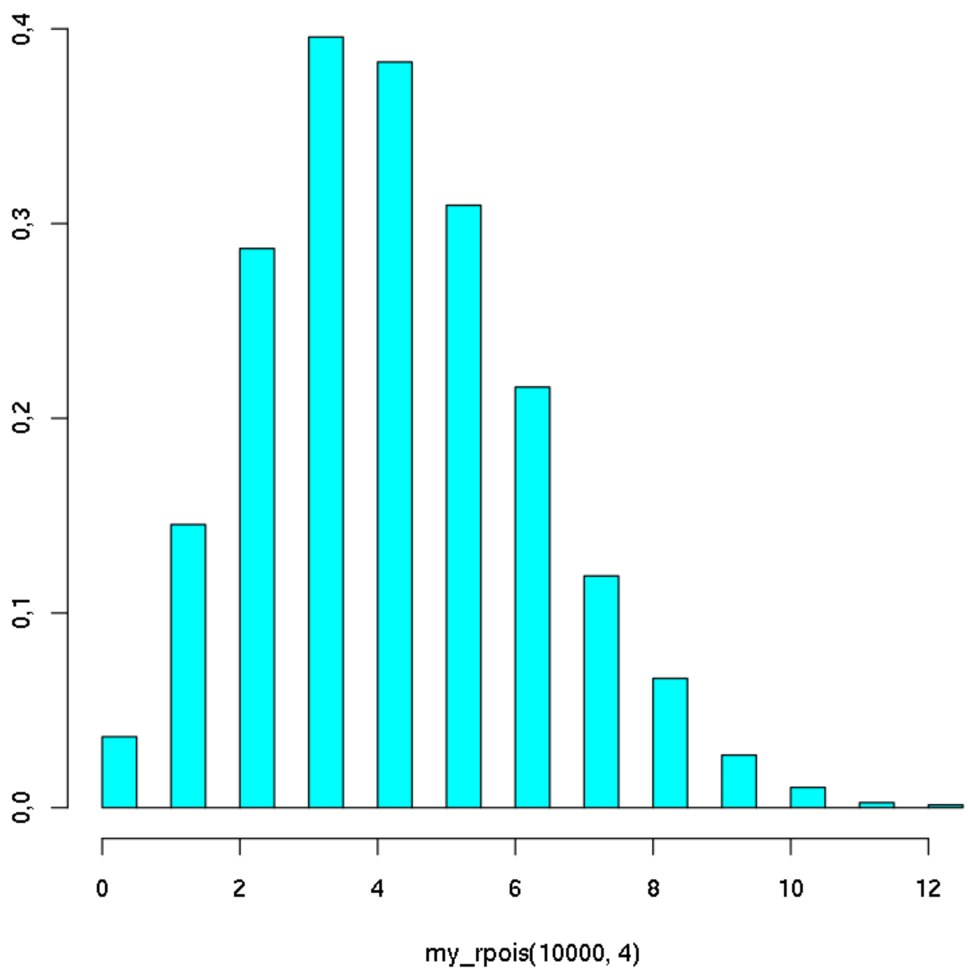
2 Exercice 3

Question 1

```
In [13]: # Fonction inverse de la repartition de la loi de poisson
         my_rpois_one = function(lambda){
             yunif = runif(1, 0, 1)
             xpois = 0
             while(yunif>ppois(xpois, lambda = 4)){xpois = xpois+1}
             return(xpois)
         }
```

In [14]:
```
# Fonction rpois
my_rpois = function(n, lambda){
    vect = c()
    for(i in 1:n){
        vect = c(vect, my_rpois_one(lambda))
    }
    return(vect)
}
```

In [15]:
```
library(MASS)
truehist(my_rpois(10000, 4))
```

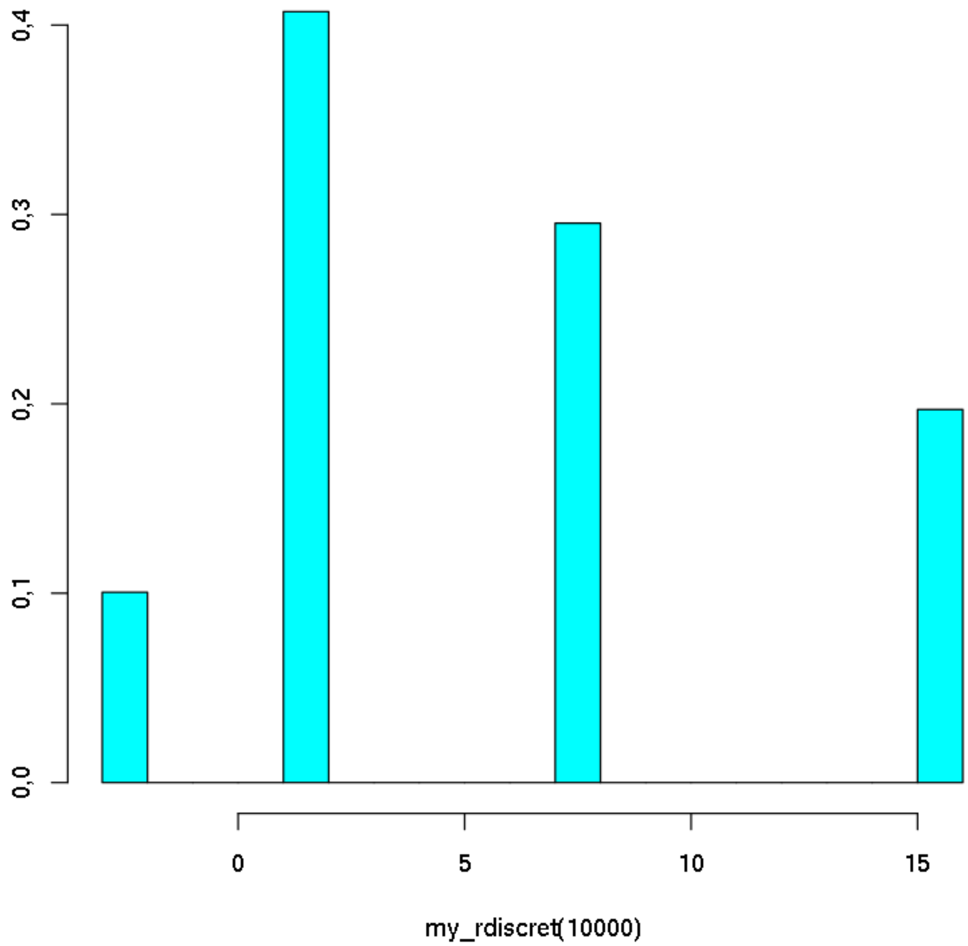


Question 2

In [16]:
```
# Fonction inverse
xdisc = c(-3, 1.3, 7, 15.2)
ydisc = c(0.1, 0.4, 0.3, 0.2)
my_rdiscret_one = function(){
    yunif=runif(1, 0, 1)
    i = 1
    while(yunif > cumsum(ydisc[1:i])[i]){i = i+1}
    return(xdisc[i])
}
```

In [17]:
```
# Fonction rdiscret
my_rdiscret = function(n){
    vect=c()
    for(i in 1:n){
        vect=c(vect, my_rdiscret_one())
    }
    return(vect)
}
```

In [18]:
```
library(MASS)
truehist(my_rdiscret(10000))
```



Question 3

- Densité `dlaplace()`

```
g(x) = 1/2*exp(x) si x <= 0
g(x) = 1/2*exp(-x) si x >= 0
```

- Répartition `plaplace()`

```
G(x) = 1/2*exp(x) si x <= 0
G(x) = 1 - 1/2*exp(-x) si x >= 0
```

- Quantile `qlaplace()`

```
G^(-1)(x) = ln(2p) si 0<=p<=1/2
G^(-1)(x) = -ln(2*(1-p)) si 1/2<=p<=1
```

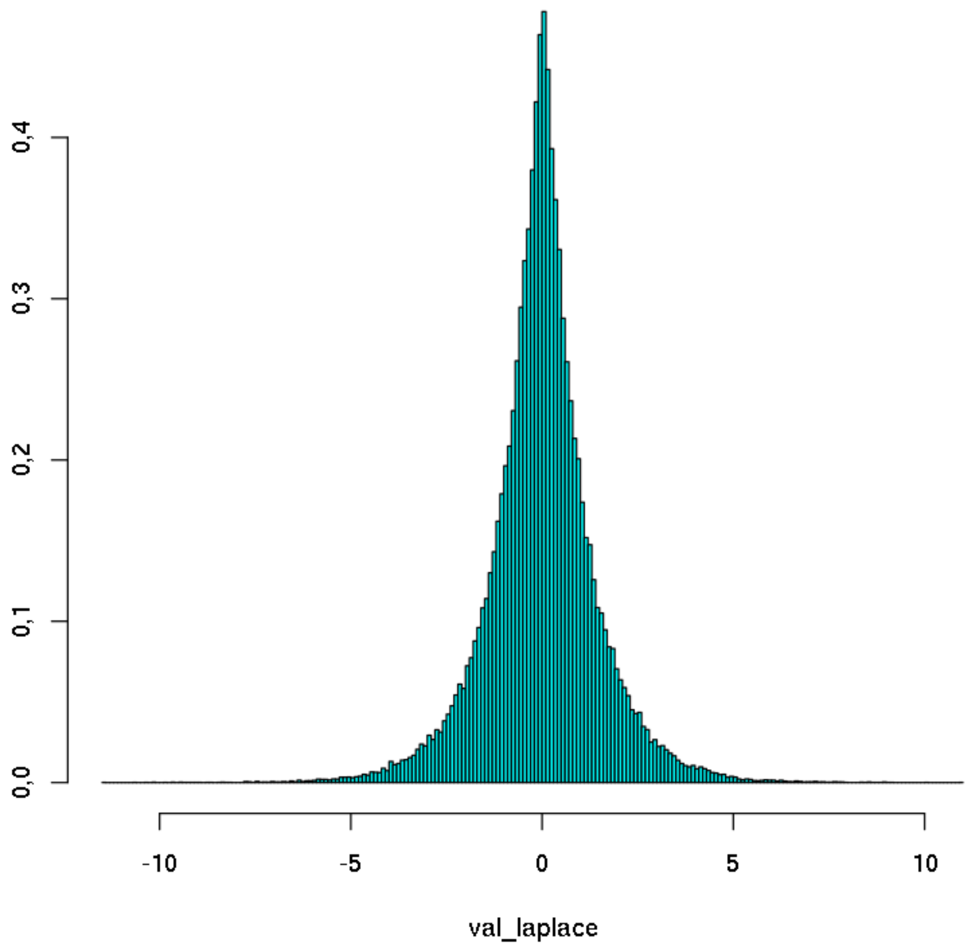
```
In [19]: my_dlaplace <- function(x){
             a = rep(0, length(x))
             a[x<=0]=1/2*exp(x[which(x<=0)])
             a[x>0]=1/2*exp(-x[which(x>0)])
             return(a)
         }
```

```
In [20]: my_plaplace <- function(x){
             a = rep(0, length(x))
             a[x<=0] = 1/2*exp(x[which(x<=0)])
             a[x>0] = 1 - 1/2*exp(-x[which(x>0)])
             return(a)
         }
```

```
In [21]: my_qlaplace <- function(p){
             a = rep(0, length(p))
             a[p>=0 & p<=1/2] = log(2*p[which(p>=0 & p<=1/2)])
             a[p>=1/2 & p<=1] = -log(2*(1-p[which(p>=1/2 & p<=1)]))
             return(a)
         }
```

```
In [22]: my_rlaplace <- function(n){
             return(my_qlaplace(runif(n, 0, 1)))
         }
```

```
In [23]: # Histograme des valeurs obtenues avec my_rlaplace
         library(MASS)
         val_laplace = my_rlaplace(100000)
         truehist(val_laplace)
```



Question 4 La fonction g est symétrique. Ainsi, rechercher le maximum de $h(x) = \frac{f(x)}{g(x)}$ sur $\mathbb{R}$ revient à le chercher pour $x \geq 0$. On a donc :

$$g(x) = \frac{1}{2} exp(-x)$$

$$f(x) = \frac{1}{\sqrt{2\pi}} exp(\frac{-x^2}{2})$$

$$h(x) = \frac{f(x)}{g(x)} = \frac{\frac{1}{2} exp(-x)}{\frac{1}{\sqrt{2\pi}} exp(\frac{-x^2}{2})}$$

$$h(x) = \sqrt{\frac{2}{\pi}} exp(x - \frac{x^2}{2})$$

En dérivant h, on obtient :

$$\frac{\mathrm{d}h}{\mathrm{d}x} = \sqrt{\frac{2}{\pi}}(1-x)exp(x-\frac{x^2}{2})$$

Le signe de la dérivée de h est :

$$\frac{\mathrm{d}h}{\mathrm{d}x}(1) = 0$$

$$\frac{\mathrm{d}h}{\mathrm{d}x}(x) \geq 0$$

$$\frac{\mathrm{d}h}{\mathrm{d}x}(x) \leq 0$$

Ainsi, le maximum de h pour $x \geq 0$ est atteint en $x = 1$, et vaut

$$f(1) = \sqrt{\frac{2exp(1)}{\pi}} = m$$

```
In [24]: # methode de rejet
         m = sqrt(2*exp(1)/pi)
         my_rnorm <- function(n){
             xi = my_rlaplace(n)
             ui = runif(n)
             return(xi[which(ui <= dnorm(xi)/(m*my_dlaplace(xi)))])
         }
```

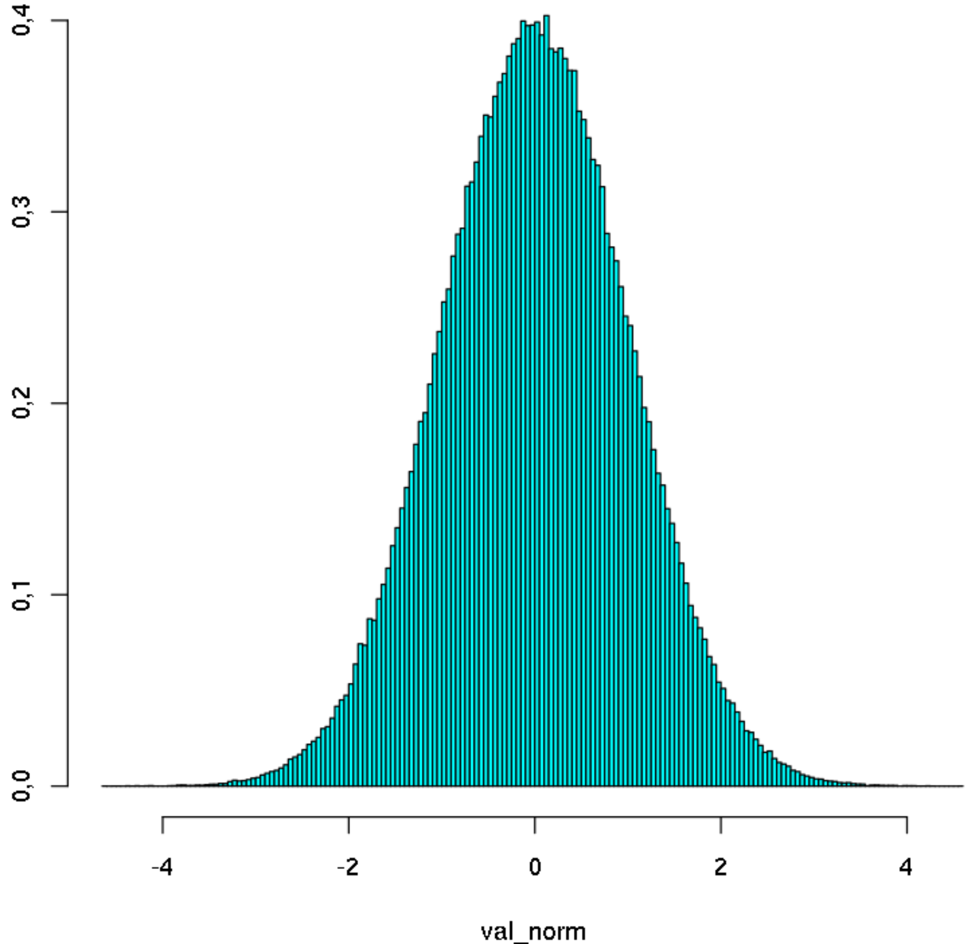
```
In [25]: # taux de rejet calculé
         length(my_rnorm(10000))/10000
```

0,7591

```
In [26]: # taux de rejet attendu
         1/m
```

0,76017345053314

```
In [27]: # Histogramme des valeurs obtenues avec my_rnorm
         library(MASS)
         val_norm = my_rnorm(500000)
         truehist(val_norm)
```


0,4
0,3
0,2
0,1
0,0
-4
-2
0
2
4
val_norm