Méthodes de Monte Carlo - TP1, solution

Master parcours SSD - UE Statistique Computationnelle

Septembre 2019

1 Simulation de variables aléatoires

Exercice 1 (utilisation des fonctions de R)

Le logiciel `R` *permet de simuler des nombres aléatoires selon de nombreuses distributions usuelles (voir la liste complète au sujet "Distributions" dans l'aide* `R`*, via la commande* `>?Distributions`*).*

1. *Générer des n-échantillons de taille croissante selon les lois normale, exponentielle, et beta.*
2. *Comparer les distributions empiriques obtenues et la distribution théorique.*

On pourra utiliser les fonctions `hist` *et* `density/plot.density` *de R.*

Rappels :

— *la distribution normale* $\mathcal{N}(\mu, \sigma)$ *de moyenne* μ *et variance* σ^2 *a pour densité*

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp \frac{-(x-\mu)^2}{2\sigma^2}, \ \textit{pour} \quad x \in \mathbb{R}.$$

— *la distribution exponentielle* $\mathcal{E}(\lambda)$ *de taux* λ *a pour densité*

$$f(x) = \lambda \exp -\lambda x, \ \textit{pour} \quad x \in \mathbb{R}^+.$$

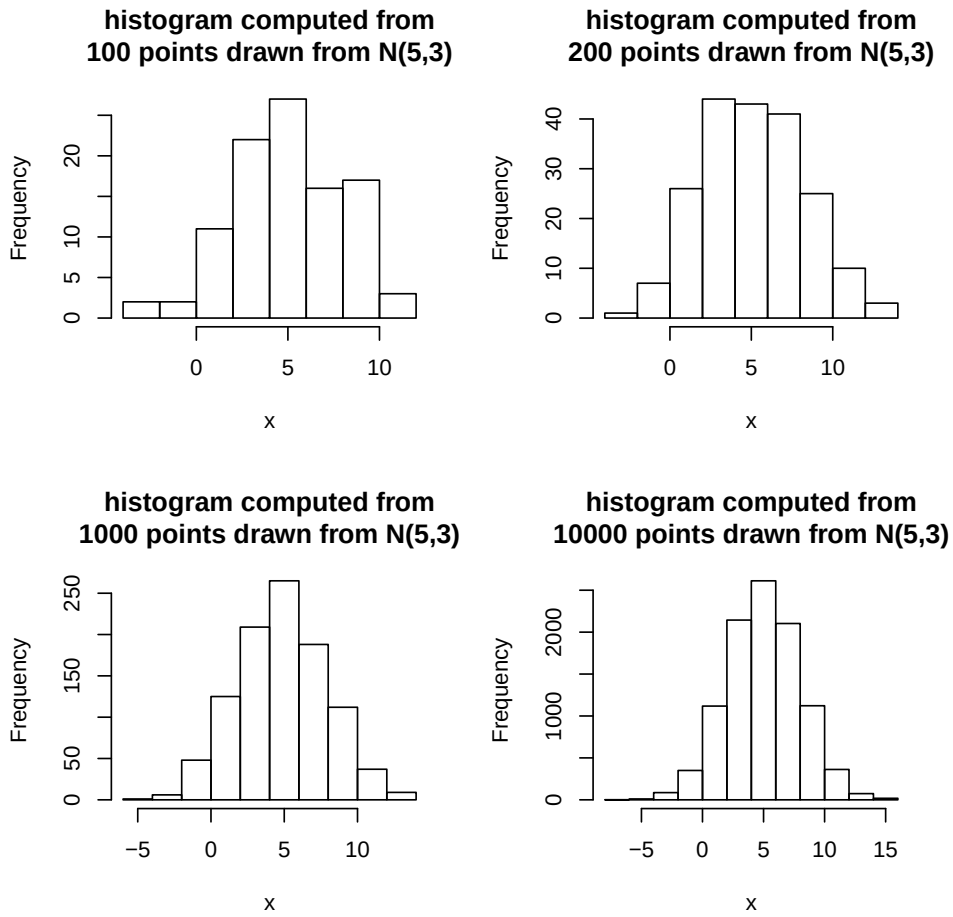
— *la distribution* $\beta(\alpha, \beta)$ *de paramètres de forme* $\alpha > 0$ *et* $\beta > 0$ *a pour densité*

$$f(x) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1}(1-x)^{\beta-1}, \ \textit{où} \ \ B(\alpha, \beta) \quad \textit{est la fonction bêta, pour} \quad x \in [0, 1].$$

On détaille l'exemple pour la loi normale. Le code suivant tire des échantillons de taille croissante et représente leurs histogrammes.

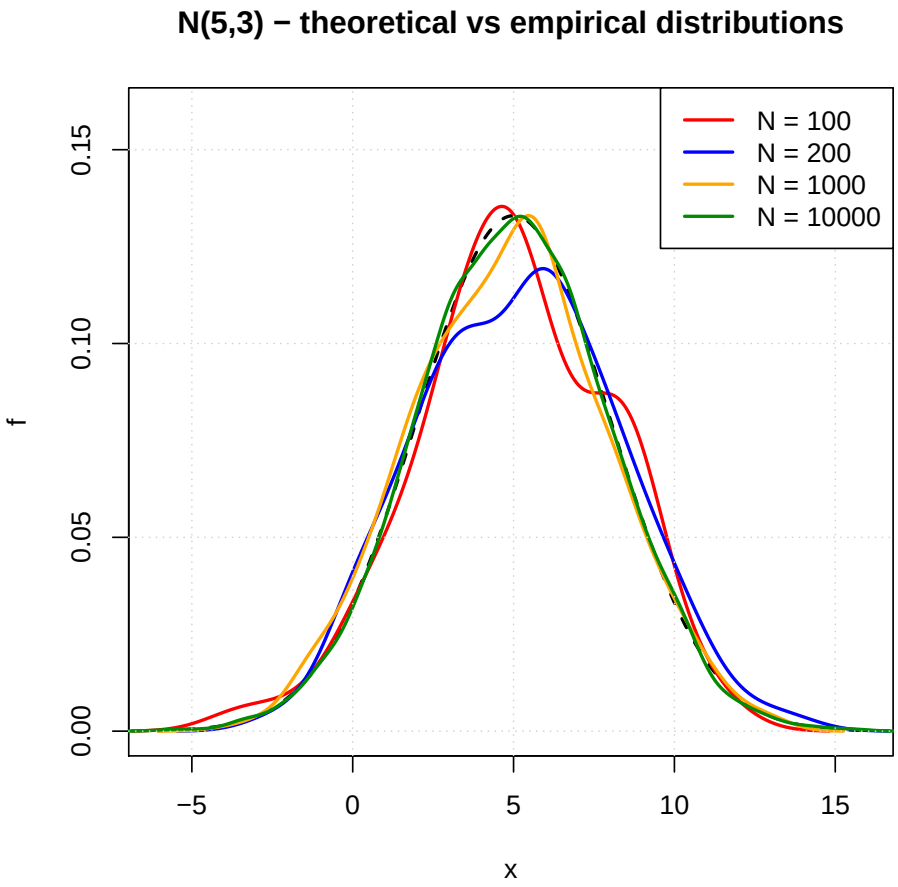
```
> ##############################
> #### STARTING EXERCICE 1 ####
> ##############################
> set.seed(20)
```

```
> # choose parameters
> mu = 5
> sigma = 3
> N = c(100, 200, 1000, 10000)
> # draw numbers
> x = list()
> for(i in 1:length(N)){
+   x[[i]] = rnorm(N[i], mean = mu, sd = sigma)
+ }
> # plot histograms
> par(mfrow = c(2,2))
> for(i in 1:length(N)){
+   hist(x[[i]], xlab = "x", main = paste("histogram computed from\n", N[i], " points drawn
+ }
```

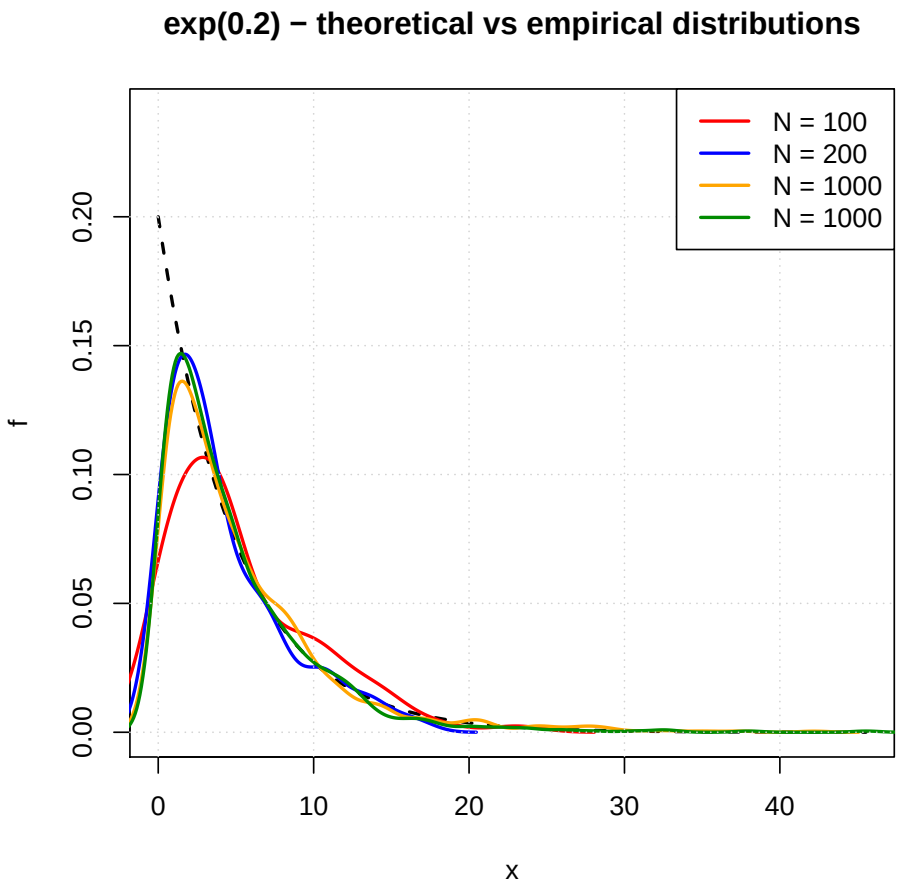


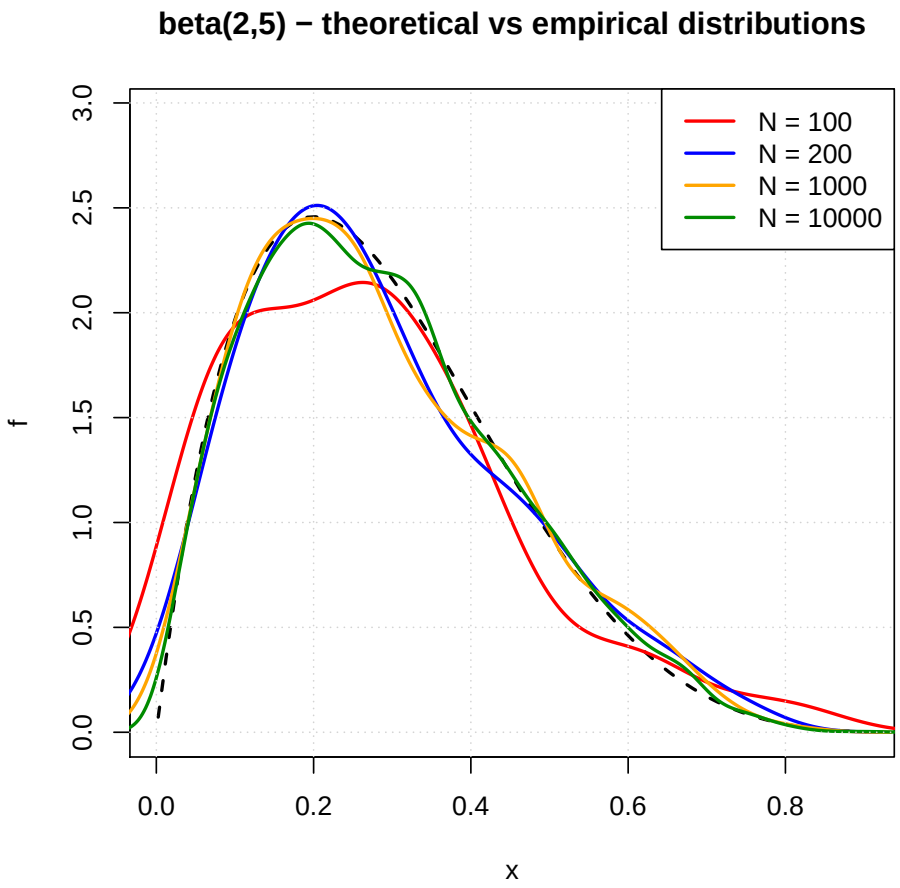
Le code suivant compare la distribution théorique aux densités empiriques obtenues par la fonction `density` de `R`.

```
> # show theoretical density
> x.range = range(unlist(x))
> x.grid = seq(x.range[1], x.range[2], by = 0.01)
> f = 1/(sigma*sqrt(2*pi)) * exp( -(x.grid-mu)^2/(2*sigma^2) )
> # compare theoretical to empirical densities
> cols = c("red","blue","orange","green4","gold")
> plot(x.grid, f, type = "l", lty = 2, lwd = 2, ylim = c(0, 1.2*max(f)), xlab = "x", main =
> for(i in 1:length(x)){
+   lines(density(x[[i]]), col = cols[i], lwd = 2)
+ }
> grid()
> legend("topright",  paste("N =",N), col = cols, lwd = 2, bg = "white")
```



Les deux figures suivantes affichent les densités obtenues pour les lois exponentielles et beta. On note que l'estimation de la densité par la méthode des noyaux (telle que réalisée par la fonction `density`) n'est pas satisfaisante pour la distribution exponentielle.


exp(0.2) − theoretical vs empirical distributions
N = 100
N = 200
N = 1000
N = 1000
f
0.00
0.05
0.10
0.15
0.20
0
10
20
30
40
x



Exercice 2 (simulation par la méthode d'inversion)

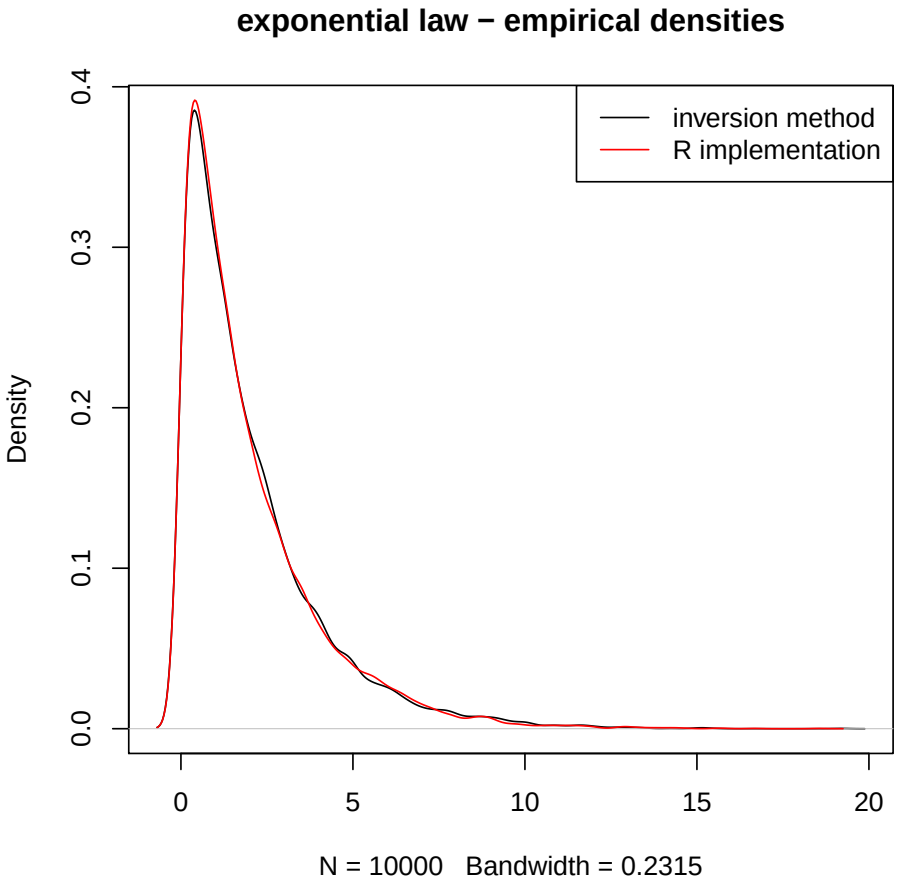
Simuler une variable suivant une loi exponentielle de paramètre 0.5 par la méthode d'inversion et comparer à la méthode proposée par R.

Pour simuler par la méthode d'inversion il faut calculer la réciproque de la fonction de répartition de la variable d'intérêt. La densité de la loi exponentielle de paramètre λ est $f(x) = \lambda \exp(-\lambda x)$. On retrouve facilement que sa fonction de répartition est $F(x) = 1 - \exp(-\lambda x)$. Pour trouver sa fonction réciproque on résoud $u = F(x)$ selon x. On trouve $x = -\frac{\log(1-u)}{\lambda} = F^{-1}(u)$. Par conséquent, si $U \to \mathcal{U}(0,1)$ alors $-\frac{\log(1-U)}{\lambda} \to \mathcal{E}(\lambda)$. On note néanmoins que que si $U \to \mathcal{U}(0,1)$ alors $(1-U) \to \mathcal{U}(0,1)$. Il suffit donc de tirer $U \to \mathcal{U}(0,1)$, et d'utiliser $-\frac{\log(U)}{\lambda}$.

Le code ci-desous illlustre cette méthode et compare les densités (empiriques) obtenues en tirant de cette manière et directement via la fonction `rexp` implémentée dans `R`.

```
> ##############################
> #### STARTING EXERCICE 2 ####
```

```
> ############################
> n = 10000
> lambda = 0.5
> x = runif(n, 0, 1)
> y = -log(x)/lambda
> plot(density(y), main = "exponential law - empirical densities")
> y2 = rexp(n, lambda)
> lines(density(y2), col = 2)
> legend("topright", c("inversion method","R implementation"), col = c(1,2), lwd = 1)
```



Exercice 3 (simulation par la méthode du rejet)

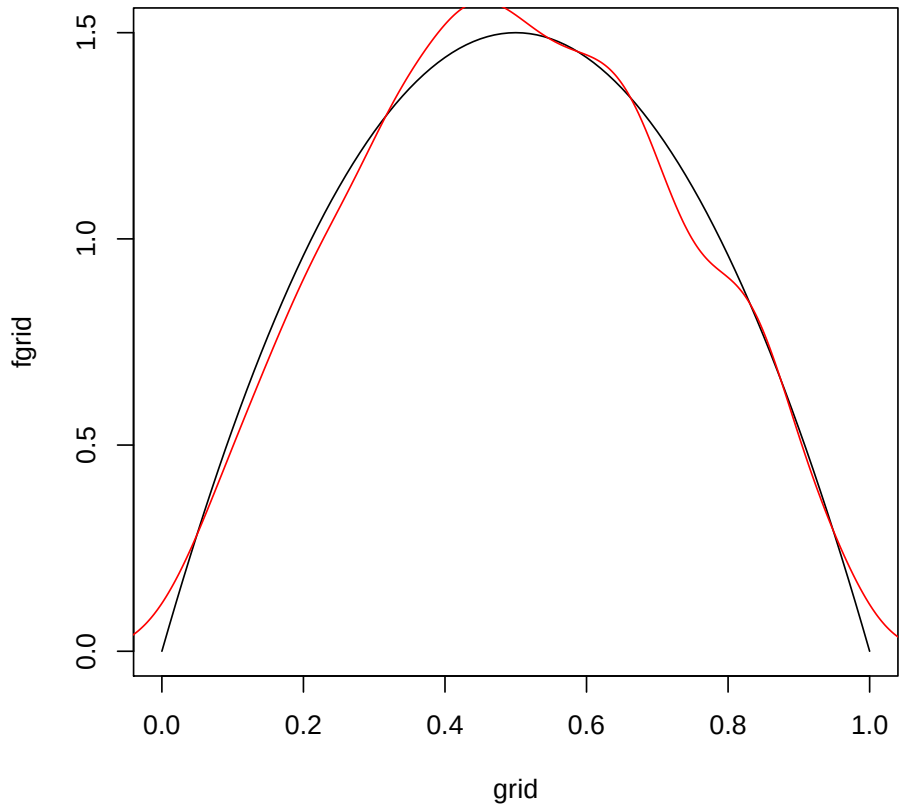
Soit la densité $f(x) = 6x(1-x)$ *définie pour* $x \in [0,1]$.

1. *Vérifier que* f *est bien une densité et trouver le plus petit* M *tel que* $f(x) \leq M$.
   Pour trouver M, on calcule la valeur maximale de f. Pour cela on annule la dérivée de f. Le maximum se trouve en $1/2$ et donc $M = f(1/2) = 1.5$.

2. *Simuler 1000 variables aléatoires distribuées selon f en utilisant la fonction $g(x) = M$ comme fonction majorante. Mesurer le taux de rejet et vérifier que la distribution empirique correspond bien à la distribution théorique.*

   Le code ci-dessous simule les données et compare la distribution empirique à la distribution théorique.

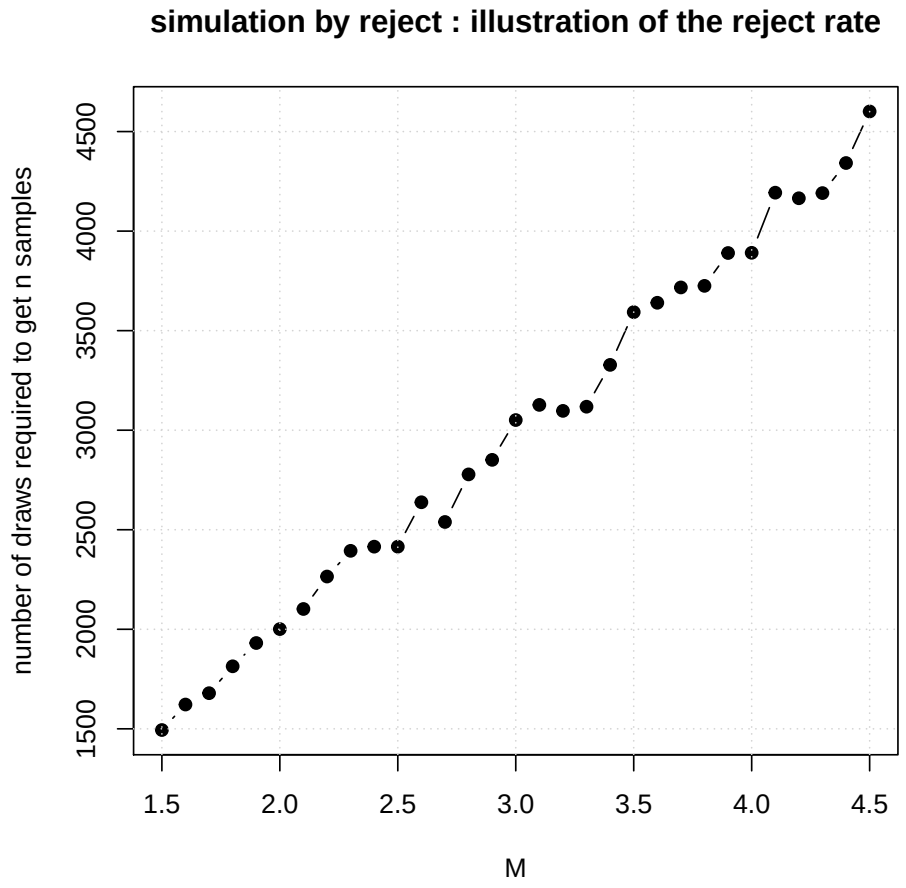
```
> ##############################
> #### STARTING EXERCICE 3 ####
> ##############################
> # draw samples
> x = c()
> cptr = 0
> M = 1.5
> n  = 1000
> while(length(x) < n){
+   a = runif(1, 0, 1)
+   b = runif(1, 0, 1)
+   if( b*M <= 6*a*(1-a) ){
+       x = c(x,a)
+   }
+   cptr = cptr + 1
+ }
> cat("number of draws required to get", n, "samples =", cptr, "\n")

number of draws required to get 1000 samples = 1458

> # compare the empirical and theoretical distributions
> grid = seq(0, 1, by = 0.01)
> fgrid = 6*grid*(1-grid)
> plot(grid, fgrid, type = "l")
> lines(density(x), col = 2)
```



3. *Tracer l'évolution du taux de rejet en fonction de M, quand M varie de sa valeur minimale à 3 fois sa valeur minimale.*

```
> # same exemple for increasing values of M
> M.grid = seq(M, 3*M, by = 0.1)
> cptr.grid = c()
> for(M2 in M.grid){
+   x = c()
+   cptr = 0
+   while(length(x) < n){
+     a = runif(1, 0, 1)
+     b  = runif(1, 0, 1)
+     if( b*M2 <= 6*a*(1-a) ){
+       x = c(x,a)
+     }
+     cptr = cptr + 1
+   }
```

```
+   cptr.grid = c(cptr.grid, cptr)
+ }
> plot(M.grid,  cptr.grid, xlab = "M", ylab = "number of draws required to get n sampl
> grid()
```



2 Méthodes MC pour l'intégration

Exercice 4 (estimation de π)

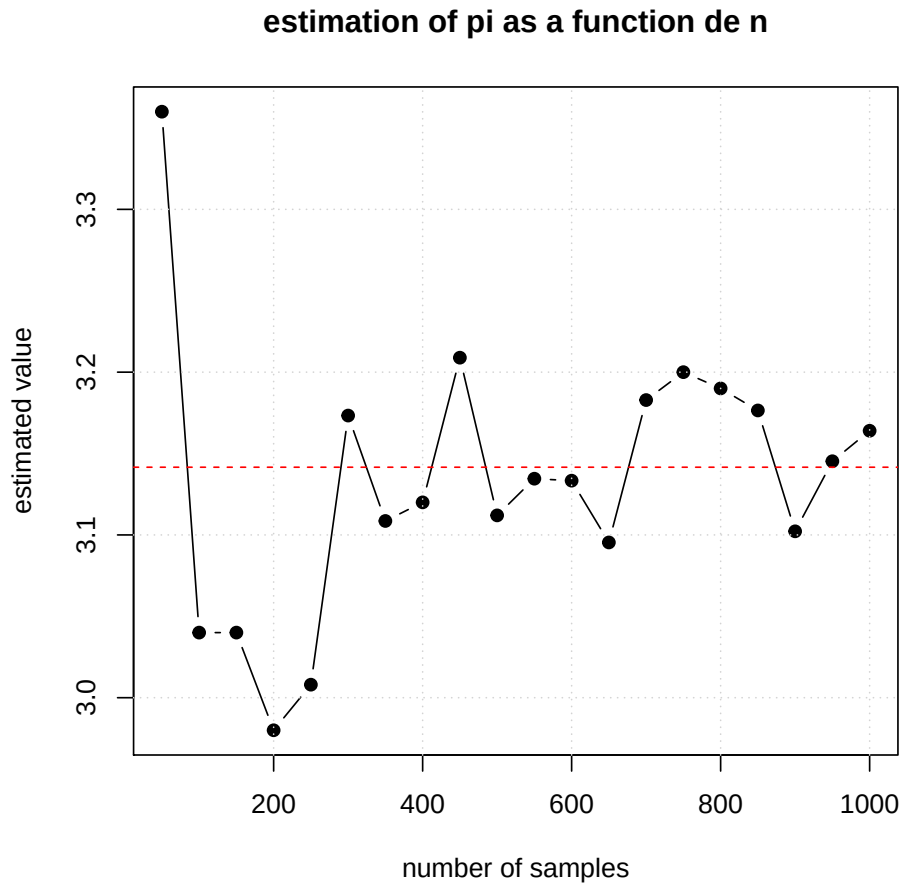
1. *Estimer π avec l'approche décrite dans le cours à partir de $n = 50$ points.*

```
> ##############################
> #### STARTING EXERCICE 4 ####
> ##############################
> n = 50
> x = runif(n, 0, 1)
> y = runif(n, 0, 1)
```

```
> pi_hat = 4*mean(x^2+y^2 <= 1)
> cat("estimation of pi from", n, "samples =", round(pi_hat, digits=4))

estimation of pi from 50 samples = 2.96
```

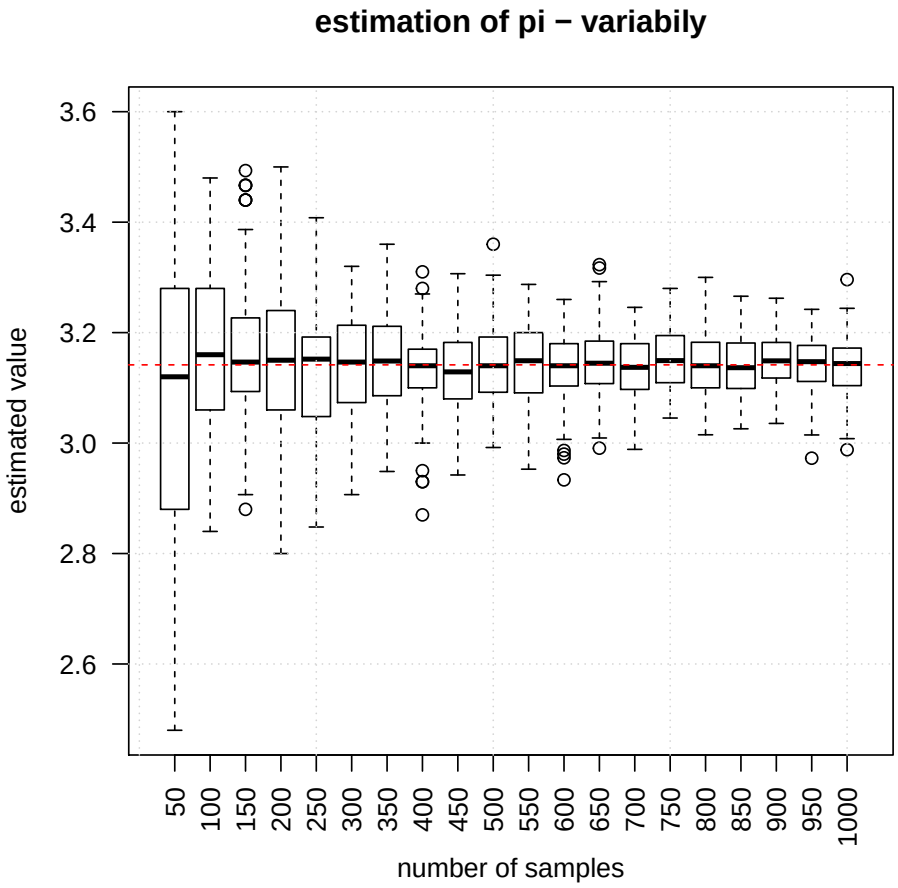
2. *Reproduire cette expérience en faisant varier n de 50 à 1000.*

```
> N = seq(50, 1000, by=50)
> pi_hat = numeric(length(N))
> for(i in 1:length(N)){
+   x = runif(N[i], 0, 1)
+   y = runif(N[i], 0, 1)
+   pi_hat[i] = 4*mean(x^2+y^2 <= 1)
+ }
> plot(N, pi_hat, type = "b", pch = 19, xlab = "number of samples", ylab = "estimated
> title("estimation of pi as a function de n")
> grid()
> abline(h = pi, lty = 2, col = 2)
```



3. *Reproduire cette seconde expérience 100 fois et visualiser la variabilité du résultat. On pourra par exemple utiliser la fonction* `boxplot` *de* `R`.

```
> N = seq(50, 1000, by=50)
> m  = 100
> pi_hat = matrix(0, nrow = m, ncol = length(N))
> for(i in 1:length(N)){
+   pi_hat[,i] = replicate(m, expr = {
+            x = runif(N[i], 0, 1)
+            y = runif(N[i], 0, 1)
+          4*mean(x^2+y^2 <= 1)
+   })
+ }
> boxplot(pi_hat, names = N, las = 2, xlab = "number of samples", ylab = "estimated value")
> title("estimation of pi - variabily")
> grid()
> abline(h = pi, lty = 2, col = "red")
```



Exercice 5

Proposer deux manières d'appproximer $I = \int_0^1 \cos(x^3) \exp(-x) dx$ *par la méthode de Monte-Carlo.*

On peut par exemple utiliser l'approche classique basée sur des tirages selon la loi uniforme $\mathcal{U}(0,1)$:

```
> ############################
> #### STARTING EXERCICE 5 ####
> ############################
> # 1) via simulation  de loi uniforme
> m = 1000
> x = runif(m)
> Ihat.1 = mean( cos(x^3)*exp(-x) )
> cat("estimation based on uniform samples =", Ihat.1, "\n")

estimation based on uniform samples = 0.6045338
```

On peut également imaginer tirer selon la loi exponentielle de paramètre $\lambda = 1$. Sa densité faut en effet $f(x) = \exp(-x)$, et I peut donc s'écrire comme $I = \int_0^1 \cos(x^3) f(x) dx$. On ne peut néanmoins pas directement considérer la moyenne empirique de la fonction $\cos(x^3)$ selon des exemples tirés selon $\mathcal{E}(1)$, car l'intégrale est définie entre 0 et 1 alors que le support de $\mathcal{E}(1)$ est $\mathbb{R}^+$. Il faut donc ré-écrire I comme $I = \int_0^{\mathbb{R}^+} \cos(x^3) \mathbf{1}(x \in [0,1]) f(x) dx$, où la fonction $\mathbf{1}(.)$ vaut 1 si son argument est vrai et zéro sinon. Le code ci-dessous implémente ce schéma.

```
> # 2) via loi exponentielle
> y = rexp(m)
> Ihat.2 = mean(cos(y^3)*(y<=1))
> cat("estimation based on eponential samples =", Ihat.2, "\n")

estimation based on eponential samples = 0.595028
```

Exercice 6

On s'intéresse à $I = \int_0^1 \sin(\sqrt{x}) dx$.

1. *Proposer une méthode MC pour calculer* I *et l'implémenter.*
   On peut applique un schéma standard basé sur la loi uniforme $\mathcal{U}(0,1)$.

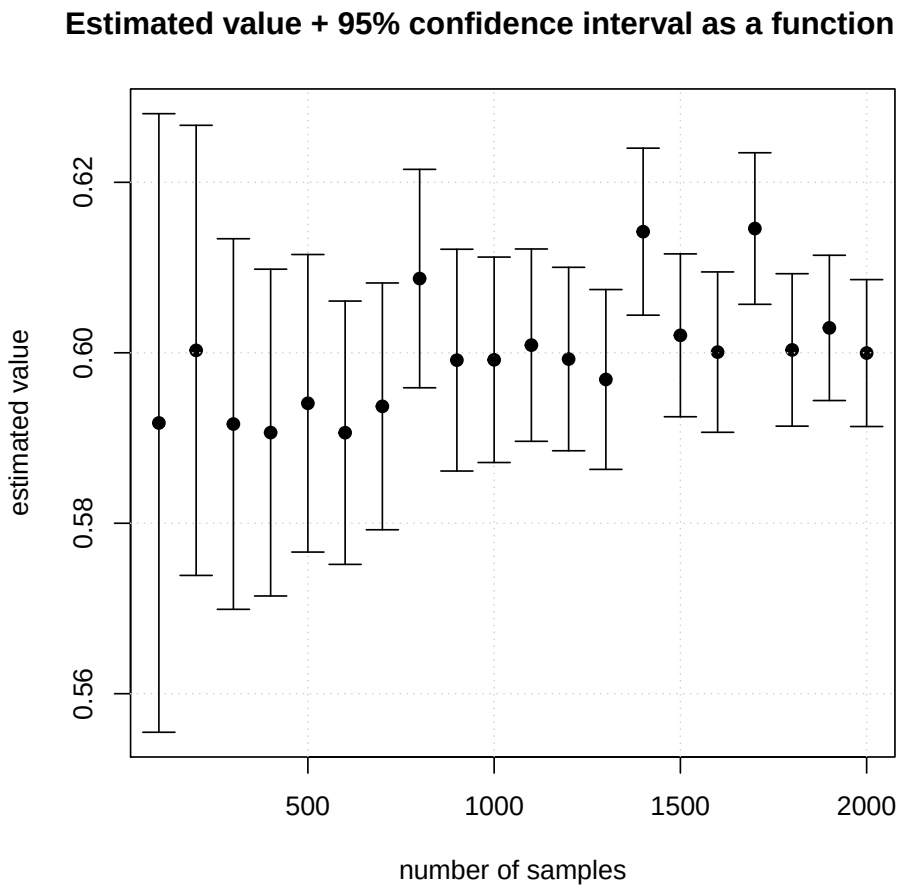
```
> ############################
> #### STARTING EXERCICE 6 ####
> ############################
> n = 1000
> x = runif(n, 0, 1)
> I.hat = mean( sin(sqrt(x)) )
> cat("estimated value based on  1000 samples =", I.hat, "\n")
```

```
estimated value based on  1000 samples = 0.6076633
```

2. *Tracer l'évolution de l'estimateur et de son intervalle de confiance à 95% en fonction du nombre de tirages.*

   Le code ci-dessous réalise cette analyse pour n allant de 100 à 2000.

```
> N = seq(100, 2000, by = 100)
> alpha = 0.05
> I.hat = c()
> I.conf = c()
> for(n in N){
+   x = runif(n, 0,1)
+   gx = sin(sqrt(x))
+   I = mean( gx )
+   I1 = I - qnorm(1-0.5*alpha)*sqrt(var(gx)/n)
+   I2 = I + qnorm(1-0.5*alpha)*sqrt(var(gx)/n)
+   # store
+   I.hat = c(I.hat,I)
+   I.conf = cbind(I.conf, c(I1,I2))
+ }
> plot(N, I.hat, ylim = range(I.conf), xlab = "number of samples", ylab = "estimated v
> title("Estimated value + 95% confidence interval as a function of n")
> grid()
> arrows(N, I.conf[1,], N, I.conf[2,], length = 0.1, angle = 90, code = 3)
```


Estimated value + 95% confidence interval as a function of n
estimated value
0.56
0.58
0.60
0.62
500
1000
1500
2000
number of samples