# Projet Monte Carlo

Ibrahim

2023-06-04

Exercice 1:

1-) Vu que nous avons la fonction de repartition de la loi de Gumbel alors pour la simulation nous allons utiliser la methode d'inverse de la fonction de repartition.

Voici les étapes pour simuler une variable aléatoire suivant la loi de Gumbel :

Etape 1: Générer un nombre aléatoire U suivant une distribution uniforme sur l'intervalle [0, 1].

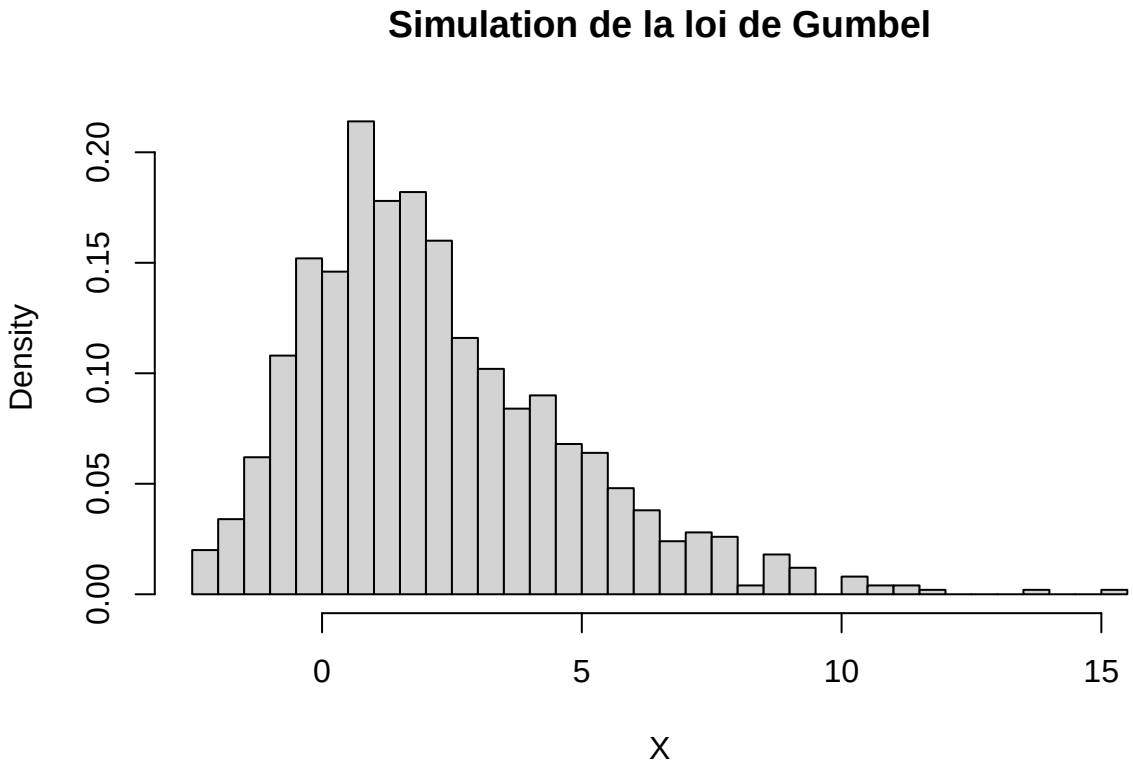
Etape 2: Calculer la variable aléatoire X en utilisant la formule : X = $\mu$ - $\beta$ * ln(-ln(U)) , où $\mu$ est la position de la queue de distribution et $\beta$ est le paramètre d'échelle de la loi de Gumbel.

Etape 3 : La variable X ainsi obtenue suit une loi de Gumbel avec les paramètres $\mu$ et $\beta$.

2-) Donnons le code R associé.

```r
simulate_gumbel <- function(n, mu, beta) {
  U <- runif(n)
  X <- mu - beta * log(-log(U))
  return(X)
}

n <- 1000
mu <- 1
beta <- 2
X <- simulate_gumbel(n, mu, beta)
hist(X, breaks = 30, freq = FALSE, main = "Simulation de la loi de Gumbel")
```

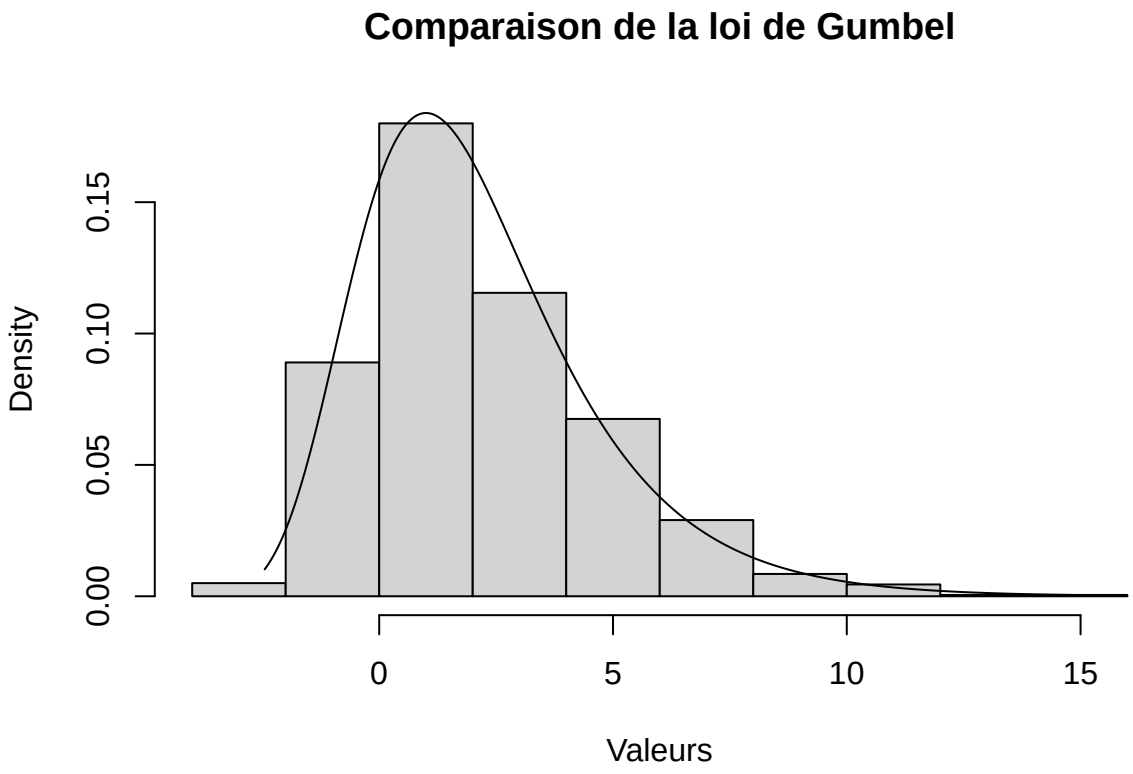



3-) Validons notre méthode de simulation:

```r
# Distribution théorique de la loi de Gumbel
hist(X, probability = TRUE, main = "Comparaison de la loi de Gumbel", xlab = "Valeurs")
t <- seq(min(X), max(X), by = 0.01)

# Simulation avec la fonction dgev de la bibliothèque evd
library(evd)
y <- dgumbel(t,loc = 1,scale = 2)

lines(t,y)
```




Exercice 2 :

1-) La methode que je propose pour calculer I est la methode de monte carlos classique.

$I = \int_0^1 \sin(\sqrt{x})\, dx$

$I = \int_{\mathbb{R}} \sin(\sqrt{x}) \cdot \mathbb{1}_{[0,1]}(x)\, dx$

$I - hat = \frac{1}{n} \sum_{k=1}^{n} \sin(\sqrt{X_k})$, avec $X_1, X_n \sim U(0,1)$ iid

2-) Ecrivons un programme R qui calcule I

```
f <- function(x) {
  sin(sqrt(x))
}

I <- integrate(f, 0, 1)$value

n=10000
X = runif(n)
z = sin(sqrt(X))
I_hat = mean(z)
print(I_hat)
```

```
## [1] 0.6035041
```

3-) Traçons l'evolution de l'estimation de I et de l'intervalle de confiance au niveau de confiance 95%

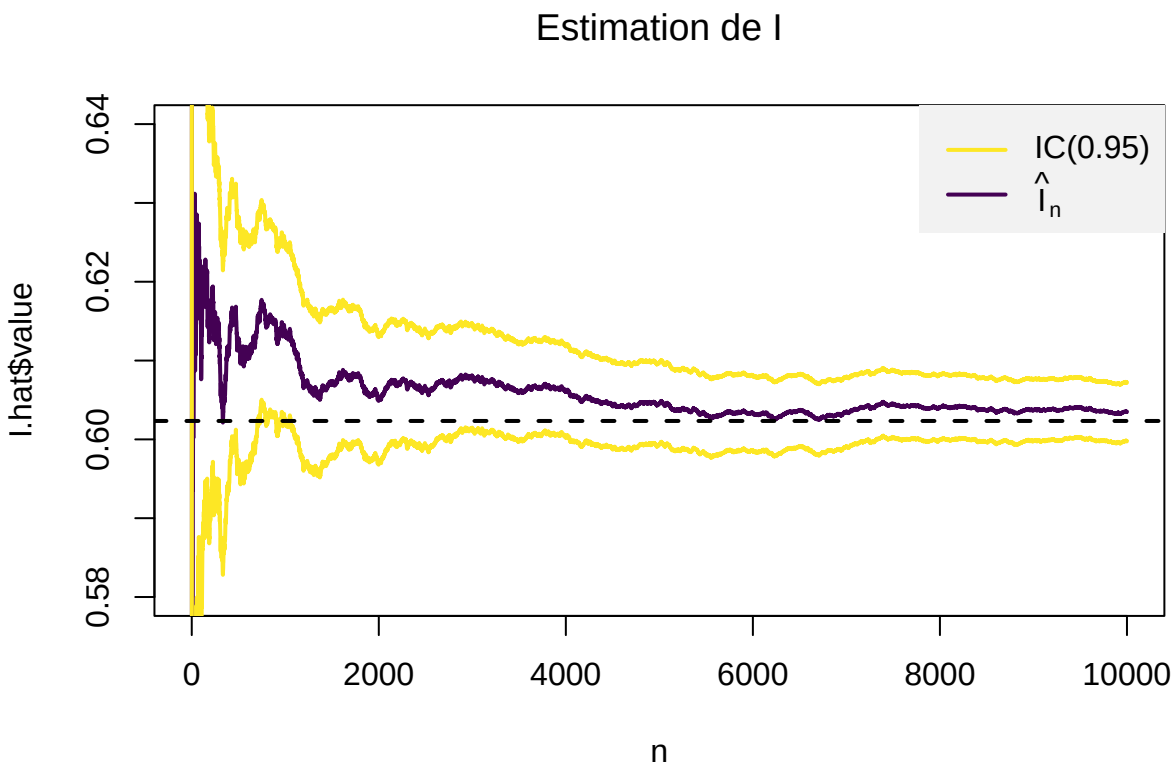
```
MC.estim.evol <- function(y, level = 0.95) {
  n <- length(y)
  delta <- cumsum(y) / (1:n)
```

```r
  s2 <- (cumsum(y^2) - (1:n) * delta^2) / (0:(n - 1))
  s2 <- c(0, s2[-1]) / (1:n)
  b.IC <- qnorm(0.5 * (level + 1)) * sqrt(s2)
  return(data.frame(value = delta, var = s2, IC.inf = delta - b.IC, IC.sup = delta + b.IC, level = level
}

I.hat <- MC.estim.evol(z)
I.hat[n, ]
```

```
##           value          var    IC.inf    IC.sup level
## 10000 0.6035041 3.601357e-06 0.5997847 0.6072236  0.95
```

```r
library(viridisLite)
c.pal = viridis(2)
plot(1:n, I.hat$value, type = "l", lwd = 2, col = c.pal[1],
main = expression(paste("Estimation de I")), xlab = "n",
ylim = range(0.58, 0.64))
lines(I.hat$IC.inf, col = c.pal[2], lwd = 2)
lines(I.hat$IC.sup, col = c.pal[2], lwd = 2)
abline(h = I, lty = 2, lwd = 2)
legend("topright", c("IC(0.95)", expression(hat(I)[n])),
col = c.pal[2:1], lwd = 2, box.lty = 0, bg = "gray95")
```




4-) Trouvons un nombre de boucle n :

Pour trouver un nombre de boucles n tel que la méthode d'approximation approche I à 0,01 près avec une probabilité≥ 0,99, nous pouvons utiliser l'inégalité de Chebyshev.

L'inégalité de Chebyshev est une inégalité probabiliste qui relie la probabilité qu'une variable aléatoire s'écarte d'une certaine distance par rapport à sa moyenne. Elle s'applique dans ce cas puisque nous cherchons

à trouver un nombre de boucles n tel que notre estimation se rapproche de la valeur réelle I avec une certaine probabilité.

L'inégalité de Chebyshev peut s'écrire comme suit:

$$P(\|X - \mu\| \geq k\sigma) \leq \frac{1}{k^2}$$

Dans notre cas, X est notre estimation, $\mu$ est la vraie valeur I, k est l'écart maximal acceptable (0,01 dans notre cas) et $\sigma$ est l'écart-type de notre estimation.

```
Z <- qnorm(1 - 0.01)  # Quantile correspondant à une probabilité de 0,99
delta <- 0.01  # Tolérance de 0,01
sigma <- sqrt(I.hat$var)  # Écart-type de l'estimateur

# Calcul du nombre de boucles n
n <- ceiling((Z * sigma / delta)^2)
```

5-) Comparaison avec la valeur que nous obtenons :

```
n <- 150000
m <- 100
ans <- numeric(m)
for (i in 1:m) {
pi.hat <- MC.estim.evol(sin(sqrt(runif(n))))
ans[i] <- max(which(abs(I - I.hat$value) > delta)) + 1 }

quantile(ans, 0.99)
```

```
##  99%
## 1082
```

Exercice 3:

1-) En utilisant R, donnons la valeur de p :

$$p = \frac{1}{2} - \frac{1}{\pi} \cdot \arctan(2)$$

```
p <- 1/2 - 1/pi * atan(2)
print(p)
```

```
## [1] 0.1475836
```

2-) Déterminons l'estimateur de Monte Carlo usuel

$$\hat{p} = \frac{1}{n} \sum_{k=1}^{n} \mathbb{1}_{X_k \geq 2}$$

Avec $X_1, \ldots, X_n$ iid suivant une loi de Cauchy $C(0, 1)$.

```r
n <- 2000
x <- rcauchy(n)
p.hat.1 <- MC.estim.evol(x >= 2)
p.hat.1[n,]
```

```
##       value          var    IC.inf    IC.sup level
## 2000 0.1435 6.148462e-05 0.1281315 0.1588685  0.95
```

3-) Montrons:

$$p = \int_2^{\infty} \frac{1}{\pi(1+x^2)}\, dx$$

$$p = \int_2^{\infty} \frac{1}{\pi(1+x^2)}\, dx$$

$$p = \frac{1}{2} - \frac{1}{\pi}(\arctan(2) - \arctan(0))$$

$$p = \frac{1}{2} - \int_0^{2} \frac{1}{\pi(1+x^2)}\, dx$$

4-) Deduisons une nouvelle methode d'estimation

```r
n <- 2000

ft <- function(x) 1 / (pi * (1 + x^2))

# Génération d'échantillons aléatoires de x
x <- runif(n, min = 0, max = 2)

# Calcul de l'expression pour chaque échantillon
expression_values <- 1/2 - (1 / (pi * (1 + x^2)))

# Estimation de p en utilisant MC.estim.evol
p.hat.2 <- MC.estim.evol(expression_values, level = 0.95)

p.hat.2[n,]
```

```
##          value          var    IC.inf    IC.sup level
## 2000 0.3204442 3.637877e-06 0.3167059 0.3241825  0.95
```

5-) Montrons:

Nous allons faire un changement de variable

$$Posons : y = \frac{1}{x}$$

$$x = \frac{1}{y}$$

$$dx = \frac{-1}{y^2} dy$$

$$p = -\int_0^{\frac{1}{2}} \frac{1}{\pi(1+\frac{1}{y^2})} \cdot \frac{-1}{y^2}\, dy$$

$$p = \int_0^{\frac{1}{2}} \frac{1}{\pi(1+y^2)}\, dy$$

6-) Deduisons une nouvelle méthode d'estimation de p.

Nous allons faire une estimation par la methode preferentielle:

$$p = \int_{\mathbb{R}} \frac{1}{2} \left( \frac{1}{\pi(1+y^2)} \right) \cdot 2 \cdot \mathbb{1}_{[0,\frac{1}{2}]}(y)\, dy$$

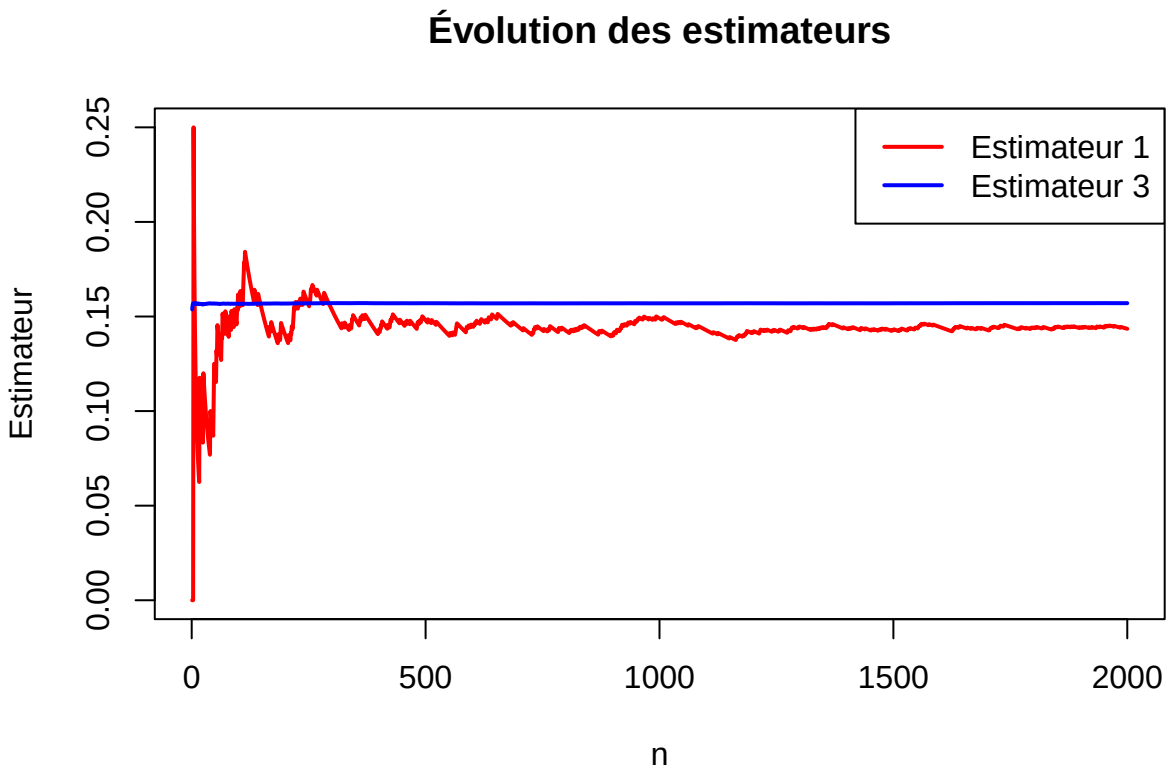
$$\hat{p} = \frac{1}{n} \sum_{k=1}^{n} \frac{1}{2\pi(1+Y_k^2)}, \quad \text{avec } Y_1, ..., Y_n \sim \mathrm{U}(0, \frac{1}{2}), \text{ indépendants identiquement distribués}$$

```
u <- runif(n, 0, 0.2)
p.hat.3 <- MC.estim.evol(1/(2 * pi * (1 + u * u)))
p.hat.3[n, ]
```

```
##          value          var    IC.inf    IC.sup level
## 2000 0.1570467 1.692505e-09 0.1569661 0.1571274  0.95
```

7-) Traçons sur un même graphique l'évolution de l'estimateur

```
n <- length(p.hat.1$value)
plot(1:n, p.hat.1$value, type = "l", lwd = 2, col = "red",
     main = "Évolution des estimateurs", xlab = "n",
     ylab = "Estimateur", ylim = range(p.hat.1$value, p.hat.3$value))
lines(1:n, p.hat.3$value, col = "blue", lwd = 2)
legend("topright", legend = c("Estimateur 1","Estimateur 3"),
       col = c("red","blue"), lwd = 2)
```

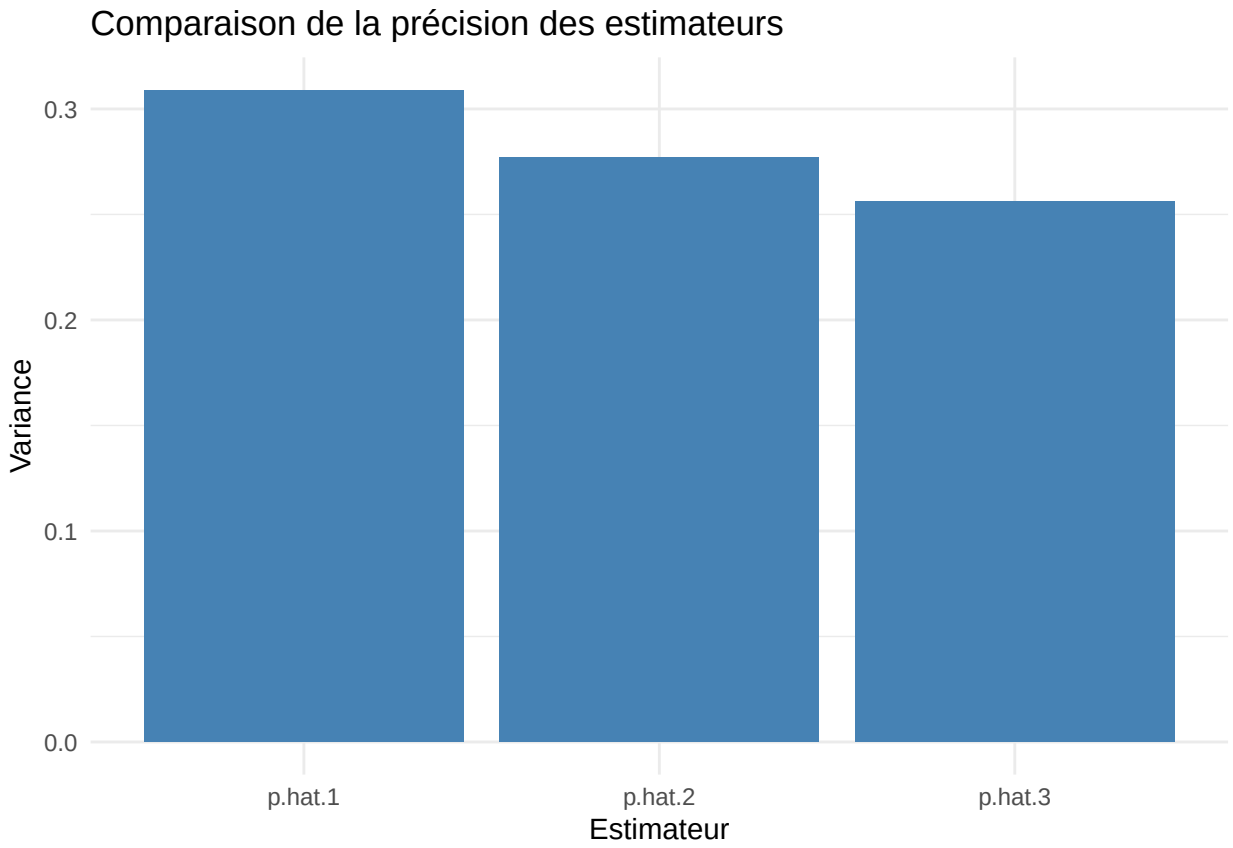



8-) Comparons la précision des différents estimateurs obtenus (en terme de réduction de variance).

Pour comparer la précision des différents estimateurs en termes de réduction de variance, nous pouvons calculer l'écart-type de chaque estimateur et les comparer entre eux. Une plus petite valeur d'écart-type indique une plus grande précision et une réduction de variance plus importante.

```
library(ggplot2)
data <- data.frame(Estimateur = c("p.hat.1","p.hat.2", "p.hat.3"),
                   Variance = c(p.hat.1$var,p.hat.2$var, p.hat.3$var))

ggplot(data, aes(x = Estimateur, y = Variance)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(x = "Estimateur", y = "Variance", title = "Comparaison de la précision des estimateurs") +
  theme_minimal()
```


Comparaison de la précision des estimateurs
Variance
0.3
0.2
0.1
0.0
p.hat.1
p.hat.2
p.hat.3
Estimateur