TP3 stats

Romain PEREIRA

2 Avril 2018

1) Ajuster une loi de Bernouilli

```
N <- 10
X_N <- rbinom(n=N, size=1, prob=0.7)
X_N
```

```
##  [1] 1 1 0 1 1 1 0 1 1 1
```

(a) Une estimation simple et empirique est $p \simeq \frac{m}{N}$, avec m le nombre de 1 dans l'echantillon.

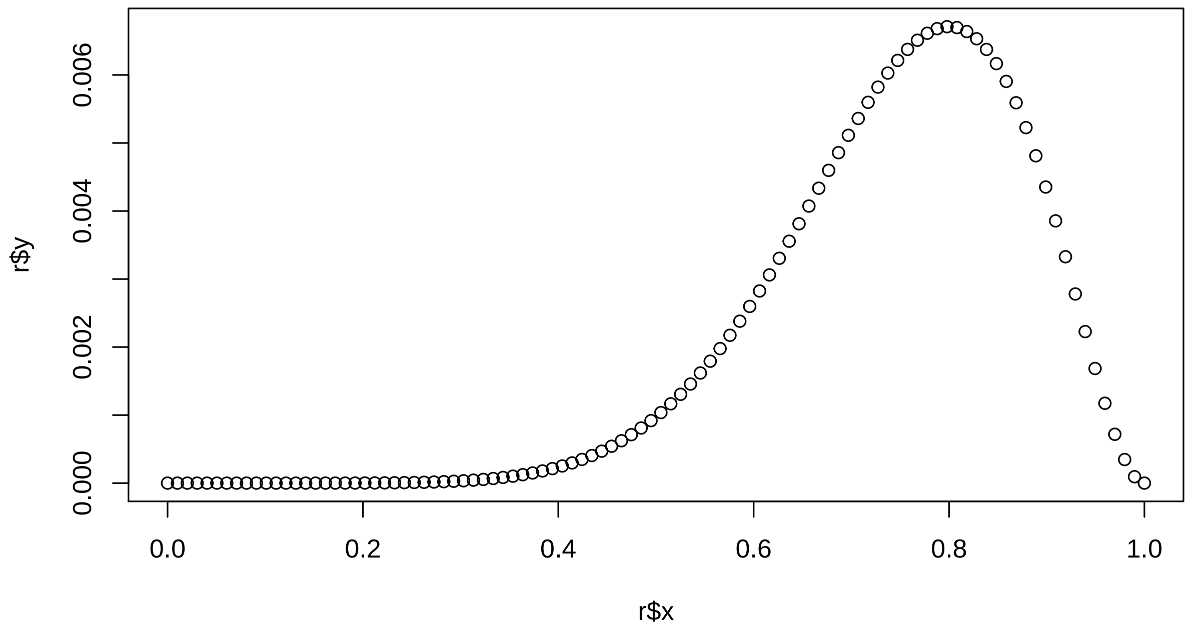
(b)

```
# fonction qui à un echantillon de Bernouilli, et une probabilité 'p',
# associe la vraisemblance d'un echantillon de Bernouilli
L_bern <- function(X_N, p) {
  s   <- sum(X_N)
  n   <- length(X_N)
  L_N <- (p**s) * ((1 - p)**(n - s))
  return (L_N)
}
L_bern(X_N, 0.5)
```

```
## [1] 0.0009765625
```

(c)

```
# calcul des valeurs 'f(x)' pour 'x' dans [x1, xn] sur 'n' points
# affiche la courbe obtenu
# renvoie le maximum atteint
optimiser_parametre <- function(f, x1, xn, n){
  # vecteurs pour le plot
  x <- c() # vecteur discretisant [xmin, xmax] (paramètre)
  y <- c() # vecteur des ordonnées (vraissemblances)

  # pour chaque x dans [x1, xn], avec une discretisation à n points...
  step <- (xn - x1) / (n - 1) # pas de discretisation
  m    <- 1 # l'indice pour lequel le maximum est atteint
  for (i in 1:n) {
    x[i] <- x1 + (i - 1) * step
    y[i] <- f(x[i])
    m    <- if (y[i] > y[m]) i else m
  }
  return (list("x"=x, "y"=y, "m"=m))
}

r <- optimiser_parametre(function(x) { return (L_bern(X_N, x)) }, 0, 1, 100)
plot(r$x, r$y)
```




```
r$x[r$m] # parametre pour lequel la vraissemblance maximum est atteinte
```

```
## [1] 0.7979798
```

```
r$y[r$m] # vraissemblance maximal
```

```
## [1] 0.006710035
```

On remarque que la vraisemblance est une fonction continue, admettant un maximum. (d)

```
optimize(function(p) { L_bern(X_N, p) }, interval=c(0, 1), maximum=TRUE)
```

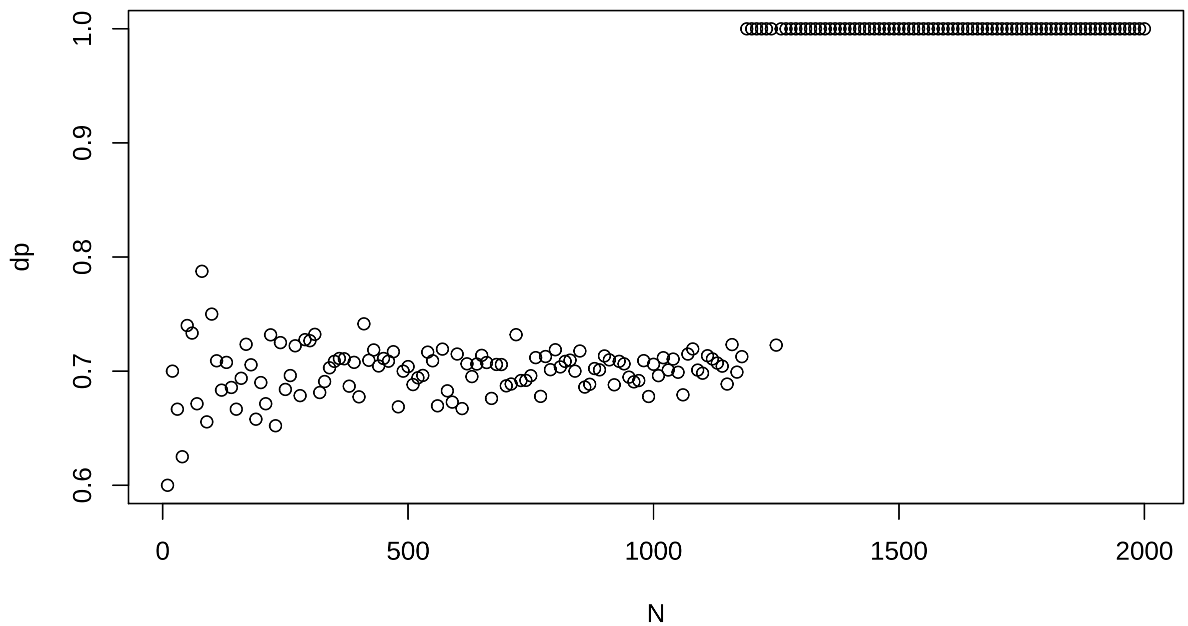
```
## $maximum
## [1] 0.799982
##
## $objective
## [1] 0.006710886
```

Remarque: on trouve une valeur proche de celle obtenu avec notre discretisation en (c).

(e)

```
p    <- 0.7 # le paramètre réel de notre loi de bernouilli
N0   <- 10
Nn   <- 2000
n    <- 200
N    <- 0 * 1:n # vecteur des 'N'
dp   <- 0 * 1:n # vecteur des '| p - px |', où p = 0.7, et px la valeur d'optimize
step <- (Nn - N0) / (n - 1)
for (i in 1:n) {
  N[i]  <- as.integer(N0 + (i - 1) * step)
  X_Ni  <- rbinom(n=N[i], size=1, prob=p)
```

```
  px    <- optimize(function(px) { L_bern(X_Ni, px) }, interval=c(0, 1), maximum=TRUE)
  dp[i] <- px$maximum
}
plot(N, dp)
```




```
mean(dp)
```

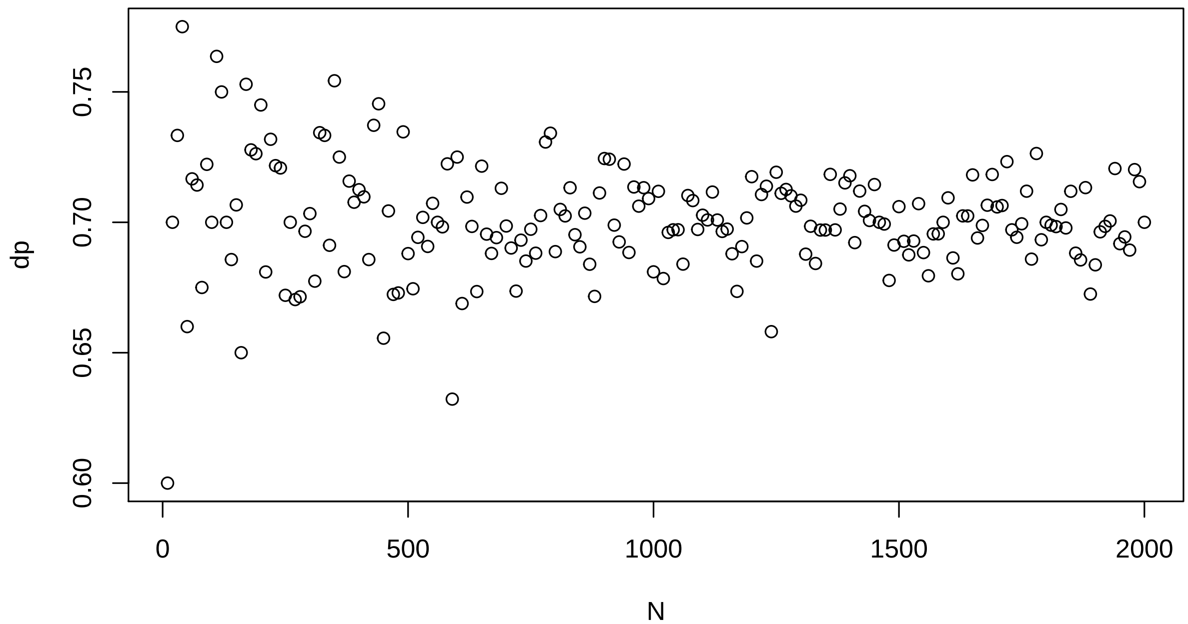
```
## [1] 0.8217109
```

On peut combattre l'instabilité des calculs en passons au log-vraissemblance.

Le produit devient alors une somme, et il n'y a plus d'instabilités dû au produit de probabilités.

L'étude est équivalent la même par la croissance stricte du *log*:

```
# fonction qui à un echantillon de Bernouilli, et une probabilité 'p',
# associe la log vraisemblance d'un echantillon de Bernouilli
log_L_bern <- function(X_N, p) {
  s   <- sum(X_N)
  n   <- length(X_N)
  L_N <- s * log(p) + (n - s) * log(1 - p)
  return (L_N)
}
# on recalcule le paramètre optimal avec la log-vraissemblance cette fois ci
for (i in 1:n) {
  N[i]  <- as.integer(N0 + (i - 1) * step)
  X_Ni  <- rbinom(n=N[i], size=1, prob=p)
  px    <- optimize(function(px) { log_L_bern(X_Ni, px) }, interval=c(0, 1), maximum=TRUE)
  dp[i] <- px$maximum
}
plot(N, dp)
```

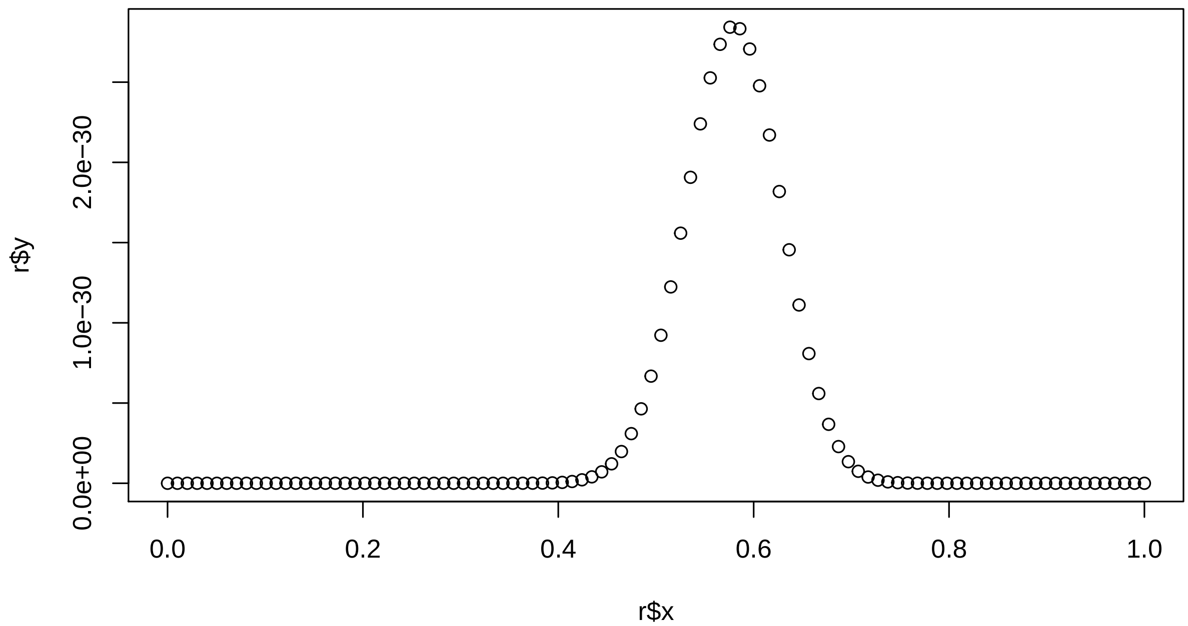



```r
mean(dp)
```

```
## [1] 0.7012504
```

(f)

```
df <- read.csv(file="./distribution_inconue_2_100_realisations.csv", header=TRUE)

r <- optimiser_parametre(function(x) { return (L_bern(df$x, x)) }, 0, 1, 100)
plot(r$x, r$y)
```




```
r$x[r$m] # parametre pour lequel la vraissemblance maximum est atteinte
```

```
## [1] 0.5757576
```

```
r$y[r$m] # vraissemblance maximal
```

```
## [1] 2.842447e-30
```

L'hypothèse que cette echantillon suit une loi de Bernouilli est possible, car l'enchantillon ne contient que 2 types de valeurs distinctes: 0 ou 1.

2) Ajuster une loi normale d'écart type connu

```
X_N <- rnorm(30, 2, 1)
```

(a)

```
# fonction qui à un echantillon Normal, et un moyenne 'mu',
# associe la vraissemblance d'un echantillon Normal
L_norm <- function(X_N, mu) {
  return (prod(dnorm(X_N, mu, 1)))
}
L_norm(X_N, 1.5)
```

```
## [1] 5.342489e-19
```

```r
L_norm(X_N, 2)
```

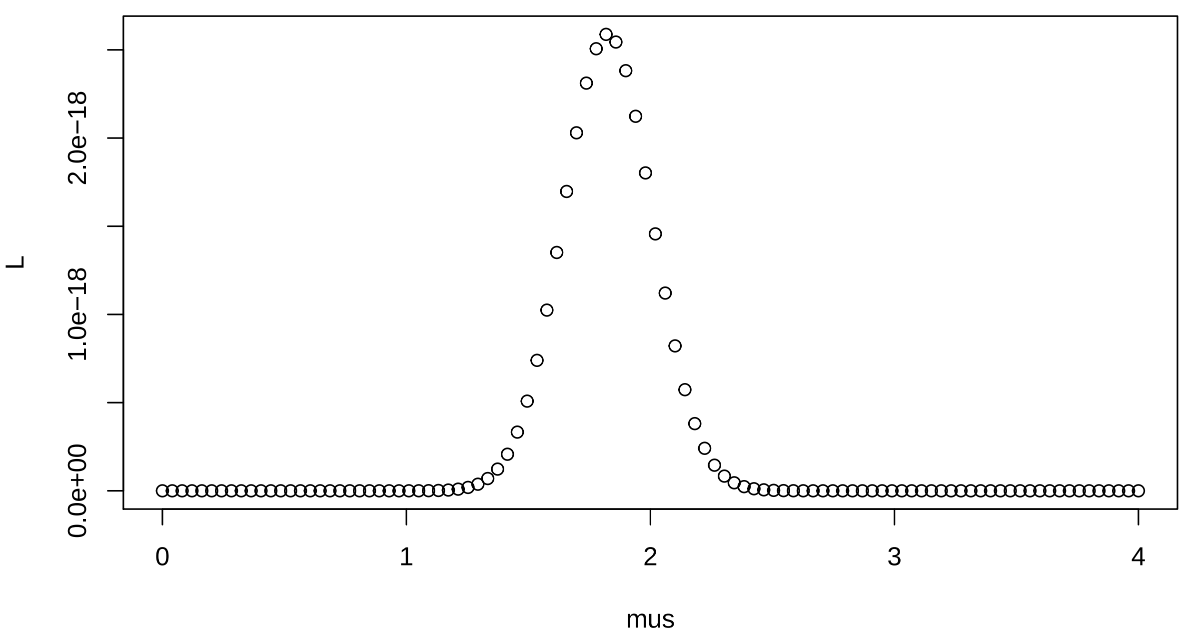
```
## [1] 1.630326e-18
```

```r
L_norm(X_N, 2.5)
```

```
## [1] 2.751672e-21
```

(b)

```
mus <- c() # vecteur discretisant [0, 4]
L   <- c() # vecteur des vraisemblances
# pour chaque p dans [0, 4], avec une discretisation à n points...
n    <- 100 # taille de la discretisation
mu0  <- 0 # [p0, pn]
muN  <- 4
step <- (muN - mu0) / (n - 1) # pas de discretisation
m    <- 1 # l'indice pour lequel le maximum est atteint
for (i in 1:n) {
  mus[i] <- mu0 + (i - 1) * step
  L[i]  <- L_norm(X_N, mus[i])
  m     <- if (L[i] > L[m]) i else m
}
plot(mus, L)
```




```
c("mu" = mus[m], "vraissemblance"=L[m])
```

```
##              mu vraissemblance
##    1.818182e+00   2.587906e-18
```

On regarde que la vraissemblance est d'autant plus élévé que le paramètre testé se rapproche du vrai paramètre. (ici $\mu = 2$)

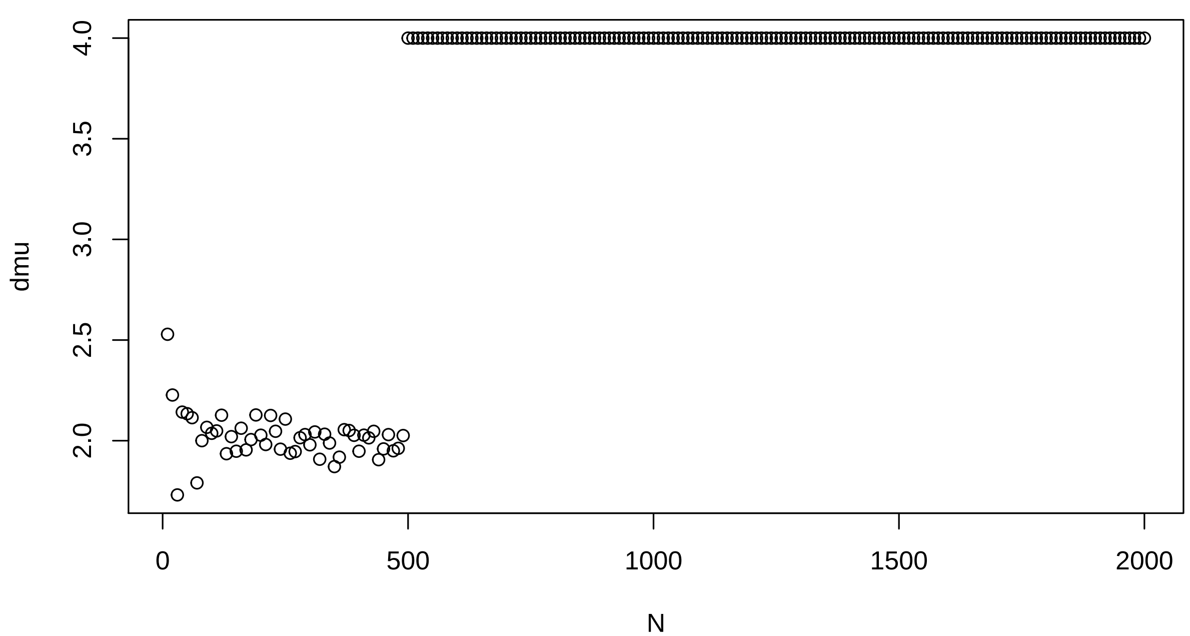
(c)

```r
optimize(function(mu) { L_norm(X_N, mu) }, interval=c(mu0, muN), maximum=TRUE)
```

```
## $maximum
## [1] 1.824376
##
## $objective
## [1] 2.589397e-18
```

(d)

```r
mu   <- 2 # le paramètre réel de notre loi
N0   <- 10
Nn   <- 2000
n    <- 200
N    <- 0 * 1:n # vecteur des 'N'
dmu  <- 0 * 1:n # vecteur des '| mu - mux |', où mu = 2, et px la valeur d'optimize
step <- (Nn - N0) / (n - 1)
for (i in 1:n) {
  N[i]   <- as.integer(N0 + (i - 1) * step)
  X_Ni   <- rnorm(n=N[i], 2, 1)
  mux    <- optimize(function(mux) { L_norm(X_Ni, mux) }, interval=c(0, 4), maximum=TRUE)
  dmu[i] <- mux$maximum
}
plot(N, dmu)
```



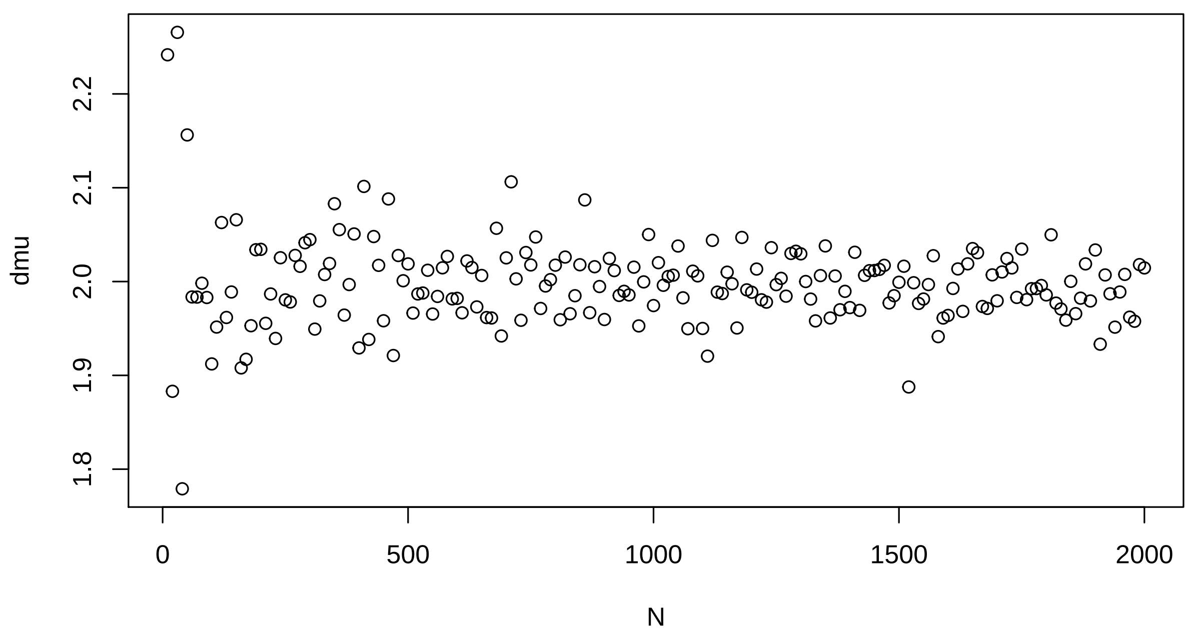

```r
mean(dmu)
```

```
## [1] 3.514525
```

On obtient alors des instabilités à cause du produit dans le calcul de la vraissemblance.

Passons à la log-vraissemblance:

```r
# fonction qui à un echantillon Normal, et un moyenne 'mu',
# associe la log vraissemblance d'un echantillon Normal
log_L_norm <- function(X_N, mu) {
  #return (prod(dnorm(X_N, mu, 1)))
  return (sum(log(dnorm(X_N, mu, 1))))
}

for (i in 1:n) {
  N[i]  <- as.integer(N0 + (i - 1) * step)
  X_Ni  <- rnorm(n=N[i], 2, 1)
  mux   <- optimize(function(mux) { log_L_norm(X_Ni, mux) }, interval=c(0, 4), maximum=TRUE)
  dmu[i] <- mux$maximum
}
plot(N, dmu)
```




```r
mean(dmu)
```

```
## [1] 1.997827
```

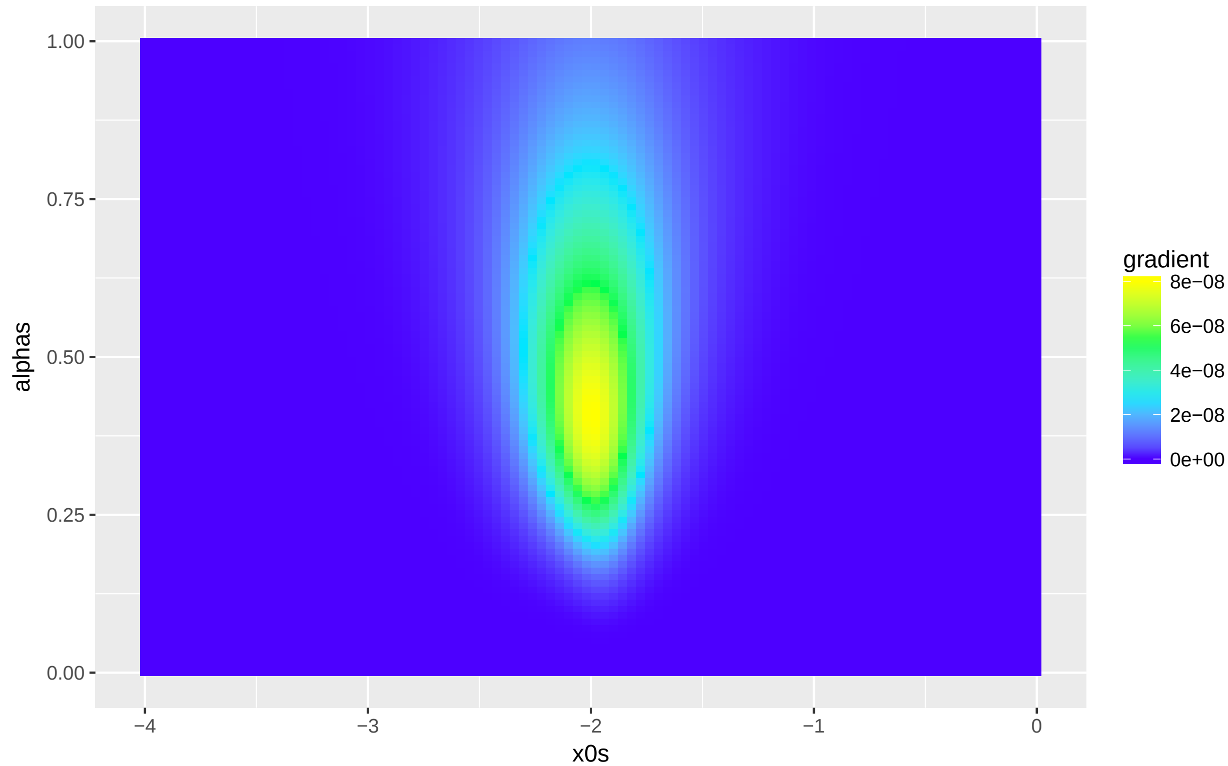
3) Ajuster une loi à plusieurs paramètres

```
# taille des enchantillons
N <- 100

# generation d'une distribution exponentielle
lambda         <- 2
L              <- 4
exponentielle  <- rexp(n=N, rate=lambda) * exp(lambda * L)

# generation d'une distribution exponentielle
x0     <- -2
alpha  <- 0.4
cauchy <- rcauchy(N, x0, alpha)

# log-vraissemblance pour une distribution exponentielle
L_exp <- function(X_N, lambda, L) {
  return (sum(log(lambda) - lambda * (X_N - L)))
}

# log-vraissemblance pour une distribution de cauchy
L_cauchy <- function(X_N, x0, alpha) {
  pi <- 3.1418
  return (sum(log(alpha / pi) - log((X_N - x0)**2 + alpha**2)))
}
```

Nous allons chercher à déterminer les paramètres de notre loi de Cauchy, en comparant la vraissemblance de notre echantillon pour une discretisation de ses 2 paramètres.

```
# import de la bibliotheque
library(ggplot2)

 # nombre de points de discretisation pour 1 dimension
n <- 100

# generation du dataframe
x0s            <- rep(seq(-4.0, 0.0, (0.0 - -4.0) / (n - 1)), times=n)
alphas         <- rep(seq(0.0, 1.0, (1.0 - 0.0) / (n - 1)), each=n)
vraissemblance <- c()
gradient       <- c()

# i pour lequel le maximum est atteint
m <- 1

for (i in 1:(n*n)) {
  vraissemblance[i] <- L_cauchy(cauchy, x0s[i], alphas[i])
  # une transformation qui préserve le maximum, mais qui permet d'avoir une meilleure image
  gradient[i]       <- exp(vraissemblance[i] * 0.1)
  m <- if (vraissemblance[i] > vraissemblance[m]) i else m
}

df <- data.frame(x0s, alphas, vraissemblance, gradient)

ggplot(df, aes(x0s, alphas)) +
```

```r
    geom_raster(aes(fill = gradient)) +
    scale_fill_gradientn(colours = topo.colors(4))
```




```r
# vrais paramètres
c("x0" = x0, "alpha_max" = alpha)
```

```
##        x0 alpha_max
##      -2.0       0.4
```

```r
# valeurs des paramètres pour lesquels la vraissemblance est maximal:
c("x0_max" = x0s[m], "alpha_max" = alphas[m])
```

```
##     x0_max  alpha_max
## -1.9797980  0.4040404
```