LSTAT 2040 - TP 6 : Solutions

L'estimateur du maximum de vraisemblance et optimisation numérique

Exercice 1

On commence par calculer la vraisemblance profilée en α donnée par

$$\ell_n^p(\alpha) = \ell_n(\alpha, \hat{\beta}(\alpha)),$$

où $\hat{\beta}(\alpha) = \text{argmax}_\beta \, \ell_n(\alpha, \beta)$.

Par différentiation, on voit que la densité sous-jacente aux données est ici donnée par

$$f(x; \alpha, \beta) = \frac{\alpha}{\beta^\alpha} x^{\alpha-1} \mathrm{I}(0 \leq x \leq \beta).$$

Il suit alors directement que pour n'importe quel $\alpha > 0$, la vraisemblance des données est une fonction décroissante de β pour $\beta \geq X_{(n)}$ et vaut 0 sinon. On déduit que pour tout $\alpha > 0$,

$$\hat{\beta}(\alpha) = \hat{\beta} = X_{(n)}.$$

Ainsi, on a

$$\ell_n^p(\alpha) = n \left\{ \log(\alpha) - \alpha \log(X_{(n)}) \right\} + (\alpha - 1) \sum_{i=1}^n \log(X_i).$$

En résolvant la condition de premier ordre associée et en observant que la fonction ℓ_n^p est strictement concave en n'importe quel point, on trouve que son maximum est atteint en

$$\hat{\alpha} = \frac{n}{\sum_{i=1}^n \log(X_{(n)}/X_i)}.$$

On déduit que le MLE de (α, β) est donné par

$$(\hat{\alpha}, \hat{\beta}) = \left(\frac{n}{\sum_{i=1}^n \log(X_{(n)}/X_i)}, X_{(n)} \right).$$

Exercice 2

(a) Après avoir exécuté le code dans l'énoncé, que l'on rappelle ici par souci de simplicité,

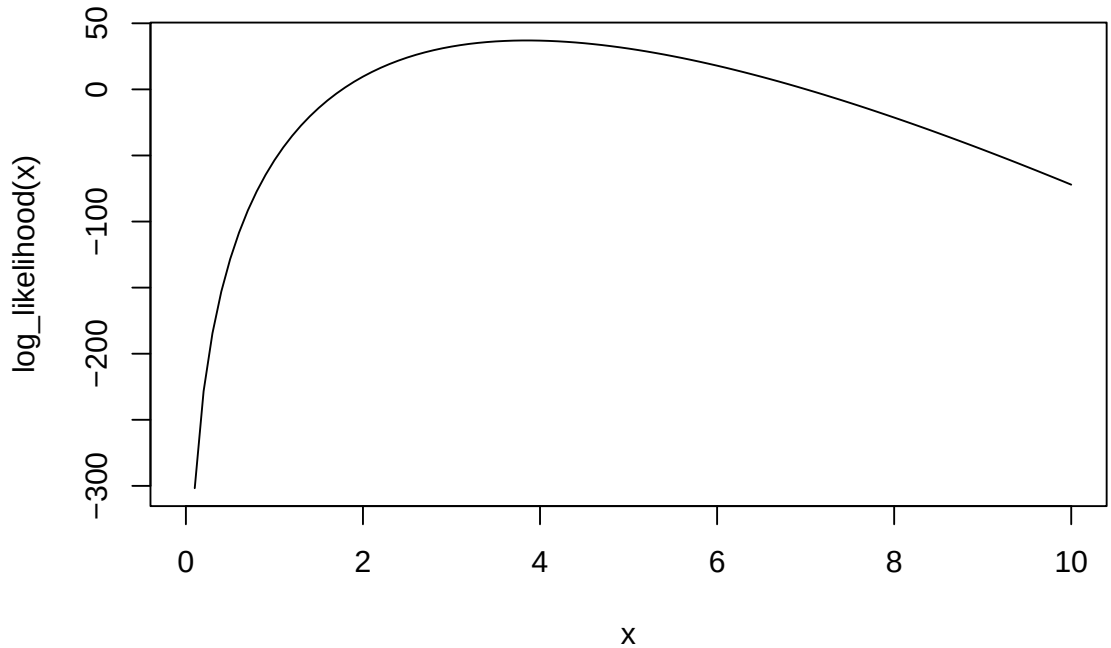
```
theta_true <- 4
set.seed(2023)
n <- 100
X <- rbeta(n, theta_true, 2)
```

on peut définir, après calculs, la fonction de log-vraisemblance comme suit :

```
log_likelihood <- function(theta){
  value <- n*(log(theta)+log(1+theta))+(theta-1)*sum(log(X))+sum(log(1-X))
  return(value)
}
```

Ensuite, il suffit de faire un graphique de celle-ci pour une plage de valeurs raisonnables pour θ. Un moyen simple d'obtenir un tel graphique est d'avoir recourt à la fonction `curve()` de `R` :

```r
curve(log_likelihood, from = 0, to = 10)
```



On constate alors que la fonction de log-vraisemblance semble bien concave dans ce cas. Un calcul simple montre que pour tout $\theta > 0$,

$$\partial_\theta^2 \ell_n(\theta) = -n \frac{2\theta^2 + 2\theta + 1}{\theta^2(1+\theta)^2} < 0,$$

ce qui garantit la concavité de la log-vraisemblance.

(b) On sait que

$$\mu = \mathrm{E}[X_1] = \frac{\theta}{\theta + 2} =: g(\theta),$$

où $g : (0, \infty) \to (0, 1)$ est inversible d'inverse donnée par $y \in (0, 1) \mapsto g^{-1}(y) = 2y/(1 - y)$. Il suit que l'estimateur des moments de θ est donné par

$$\hat{\theta}_{\mathrm{MM}} := g^{-1}(\overline{X}_n) = \frac{2\overline{X}_n}{1 - \overline{X}_n}.$$

On le calcule sur les données via le code

```r
theta_MoM <- 2*mean(X)/(1-mean(X))
theta_MoM
```

```
## [1] 3.770508
```

Notons que l'estimateur des moments est déjà assez proche de la vraie valeur du paramètre sous-jacente à la génération des données.

(c) Afin d'appliquer l'algorithme de Newton–Raphson, nous aurons besoin de la dérivée première et seconde de la log-vraisemblance :

```r
#Dérivée première :
grad_logLik <- function(theta){
  value <- n*(1/theta + 1/(theta+1)) + sum(log(X))
  return(value)
}
#Dérivée seconde :
hess_logLik <- function(theta){
  value <- -n*(1/theta^2 + 1/(theta+1)^2)
  return(value)
}
```

L'algorithme est alors donné par :

```r
NR_optim <- function(theta0 = theta_MoM, tol = 10^(-6), trace = FALSE){
  diff <- 1
  theta <- theta0
  while(diff > tol){
    theta_old <- theta
    grad <- grad_logLik(theta_old)
    hess <- hess_logLik(theta_old)
    if(trace == TRUE) print(c(theta_old, log_likelihood(theta_old), grad, hess))
    theta <- theta_old - grad/hess
    diff <- abs(theta - theta_old)
  }
  return(theta)
}
```

Notons que la valeur par défaut de la valeur initiale de θ dans l'algorithme est l'estimateur des moments et que la tolérance par défaut entre deux itérations de l'algorithme est de 10^{-6}, comme demandé. L'argument `trace` est simplement là pour garder un oeil sur l'évolution de θ, de $\ell_n(\theta)$ et de ses dérivées lors de l'évolution de l'algorithme. Si l'on applique l'algorithme (en indiquant `trace=TRUE` afin de mieux visualiser la procédure), on trouve :

```r
theta_NR <- NR_optim(trace = TRUE)
```

```
## [1]   3.7705080  36.9774423   0.9075915 -11.4280740
## [1]   3.84992575  37.01394030   0.01723785 -10.99813944
## [1]  3.851493e+00  3.701395e+01  6.455917e-06 -1.098990e+01
```

```r
theta_NR
```

```
## [1] 3.851494
```

L'algorithme converge assez rapidement puisque la valeur initiale est très proche de la valeur de θ qui maximise la log-vraisemblance de nos données. Il suffit de trois itérations.

(d) On applique notre algorithme du point (c) avec $\theta = 10$ comme point intial :

```r
NR_optim(theta0 = 10, trace = TRUE)
```

```
## [1]  10.000000 -72.086600 -27.485252  -1.826446
## [1]  -5.048487        NaN -91.084666 -10.024726
## [1] -14.134487        NaN -61.264600  -1.080202
## [1] -70.85036704          NaN -49.41921838  -0.04041691
## [1] -1.293587e+03           NaN -4.673083e+01 -1.196121e-04
## [1] -3.919802e+05           NaN -4.657667e+01 -1.301676e-09
## [1] -3.578248e+10           NaN -4.657616e+01 -1.562029e-19
## [1] -2.981773e+20           NaN -4.657616e+01 -2.249474e-39
```

```
## [1] -2.070536e+40           NaN -4.657616e+01 -4.665137e-79
## [1]  -9.983878e+79            NaN  -4.657616e+01 -2.006465e-158
## [1] -2.321305e+159            NaN  -4.657616e+01   0.000000e+00
## [1]      -Inf       NaN -46.57616   0.00000
```

```
## Error in while (diff > tol) {: missing value where TRUE/FALSE needed
```

On constate que l'algorithme ne fonctionne plus. En effet, lors de la première itération, la dérivée première étant tellement négative (et la dérivée seconde étant négative également), la nouvelle valeur de θ obtenue devient négative, et la log-vraisemblance n'est pas définie pour de telles valeurs. Un moyen simple de corriger cela est d'adapter l'algorithme afin que, si une itération mène à une valeur négative de θ, on prend $\theta = 0.01$ (par exemple) pour l'itération d'après :

```r
NR_optim_2 <- function(theta0 = theta_MoM, tol = 10^(-6), trace = FALSE){
  diff <- 1
  theta <- theta0
  while(diff > tol){
    theta_old <- theta
    grad <- grad_logLik(theta_old)
    hess <- hess_logLik(theta_old)
    if(trace == TRUE) print(c(theta_old, log_likelihood(theta_old), grad, hess))
    theta <- theta_old - grad/hess
    if(theta <= 0) theta <- 0.01 #adaptation
    diff <- abs(theta - theta_old)
  }
  return(theta)
}
```

On applique le nouvel algorithme avec le même point initial :

```r
NR_optim_2(theta0 = 10, trace = TRUE)
```

```
## [1]  10.000000 -72.086600 -27.485252  -1.826446
## [1]        0.0100     -536.3608    10052.4337 -1000098.0296
## [1]  2.005145e-02 -4.662670e+02  5.038629e+03 -2.488148e+05
## [1]  4.030196e-02 -3.954345e+02  2.530818e+03 -6.165934e+04
## [1]  8.134714e-02 -3.232441e+02  1.275201e+03 -1.519729e+04
## [1]     0.1652569  -248.8013309   644.3604506 -3735.3327500
## [1]    0.337761 -171.546569  324.242928 -932.436860
## [1]    0.6854982  -93.8557256  158.6327916 -248.0078003
## [1]   1.325126 -25.564099  71.896753 -75.446144
## [1]   2.278081  18.581871  27.826086 -28.575059
## [1]   3.251870  34.826268   7.694434 -14.988031
## [1]   3.765242  36.972505   0.967847 -11.457479
## [1]   3.84971517  37.01393642   0.01955399 -10.99924678
## [1]  3.851493e+00  3.701395e+01  8.306520e-06 -1.098990e+01
```

```
## [1] 3.851494
```

Cela converge vers la même valeur qu'obtenue au point (c). Noter qu'un nombre largement supérieur d'itérations a été nécessaire à obtenir la convergence car le point $\theta = 0.01$ est fort éloigné du point qui maximise la log-vraisemblance (d'ailleurs, rien ne garantit à priori qu'il y ait convergence pour une vraisemblance générale). Ces observations ne peuvent que vous inciter à bien considérer le choix de votre point initial.

(e) Puisque la log-vraisemblance est strictement concave, il suffit de résoudre l'équation de premier ordre associée pour trouver le MLE. Cela conduit à une équation du second ordre en θ dont l'unique solution

positive est donnée par

$$\hat{\theta}_{\text{MLE}} := \frac{-2 - \overline{\log(X)}_n + \sqrt{4 + \overline{\log(X)}_n^2}}{2\overline{\log(X)}_n},$$

où $\overline{\log(X)}_n = n^{-1}\sum_{i=1}^n \log(X_i)$. On peut alors le calculer sur nos données :

```
theta_MLE <- (-2-mean(log(X))-sqrt(4+mean(log(X))^2))/(2*mean(log(X)))
theta_MLE
```

```
## [1] 3.851494
```

A 6 décimales près, il n'y a aucune différence entre le MLE théorique calculé sur nos données et le résultat obtenu numériquement. En effet, l'erreur entre le vrai paramètre sous-jacent à la génération de nos données et nos estimations est presque uniquement statistique :

```
#erreur numérique absolue
abs(theta_NR - theta_MLE)
```

```
## [1] 8.260059e-14
```

```
#erreur statistique absolue
abs(theta_MLE - theta_true)
```

```
## [1] 0.1485063
```

(f) Si l'on dispose d'une forme fermée pour l'information de fisher $I_n(\theta)$, ce qui est le cas ici, on peut simplement calculer cette information en le MLE (numérique si l'analytique n'est pas disponible). Un bref calcul montre que

```
I_n <- function(theta){
  value <- n*(2*theta^2 + 2*theta +1)/(theta^2*(theta+1)^2)
  return(value)
}
```

Dès lors, la variance asymptotique estimée du MLE est donnée par :

```
1/I_n(theta_MLE)
```

```
## [1] 0.09099264
```

L'autre méthode consiste à utiliser l'information de Fisher observée sur les données, qui est un estimateur consistant de la vraie information de Fisher. Celle-ci est donnée par :

```
-1/hess_logLik(theta = theta_MLE)
```

```
## [1] 0.09099264
```

Ici, les deux quantités sont exactement les mêmes car la dérivée seconde de la log-vraisemblance n'est pas aléatoire et le fait de prendre l'espérance ou une moyenne arithmétique ne change donc rien. Cela ne sera pas le cas en général. Comme on connait ici la valeur du vrai paramètre, on peut comparer nos résultats avec la vraie variance asymptotique

```
1/I_n(4)
```

```
## [1] 0.09756098
```

et constater que notre estimation est très proche de la réalité.

Exercice 3

(a) On sait de l'exercice précédent que

$$\mu = \mathrm{E}[X_1] = \frac{\theta}{\theta+2} =: g(\theta),$$

où $g : (0,\infty) \to (0,1)$ est inversible. Il suit que le MLE de μ est donné par

$$\hat{\mu}_{\mathrm{MLE}} = g(\hat{\theta}_{\mathrm{MLE}}).$$

De plus, on sait également de l'exercice précédent que

$$\sqrt{n}(\hat{\theta}_{\mathrm{MLE}} - \theta) \xrightarrow{d} \mathrm{N}\left(0, \frac{\theta^2(\theta+1)^2}{2\theta^2+2\theta+1}\right),$$

et il suit donc de la méthode delta avec $g'(\theta) = 2/(\theta+2)^2$ que

$$\sqrt{n}(\hat{\mu}_{\mathrm{MLE}} - \mu) \xrightarrow{d} \mathrm{N}\left(0, \frac{4\theta^2(\theta+1)^2}{(\theta+2)^4(2\theta^2+2\theta+1)}\right).$$

(b) Le CLT donne

$$\sqrt{n}(\overline{X}_n - \mu) \xrightarrow{d} \mathrm{N}\left(0, \frac{2\theta}{(\theta+2)^2(\theta+3)}\right).$$

Un petit calcul montre alors que l'inégalité

$$\frac{\mathrm{AVar}[\overline{X}_n]}{\mathrm{AVar}[\hat{\mu}]} > 1$$

est satisfaite, montrant ainsi que le MLE possède une variance asymptotique plus faible que la moyenne empirique, en accord avec la théorie.

Exercice 4

(a) Après avoir exécuté le code dans l'énoncé, que l'on rappelle ici par souci de simplicité,

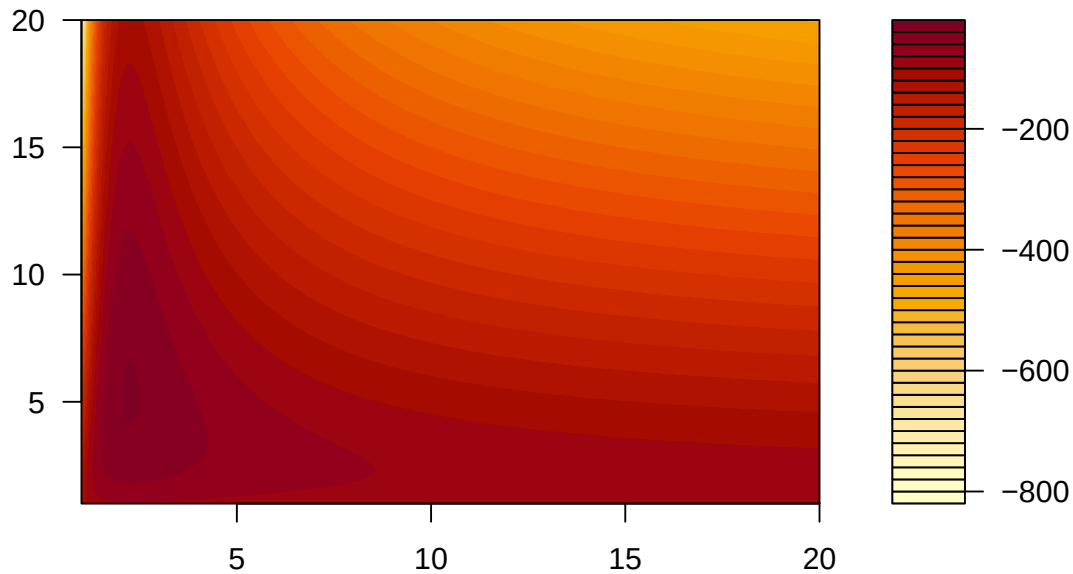
```r
mu_true <- 2
lambda_true <- 5
set.seed(2023)
n <- 100
# install.packages("statmod") # exécuter si le package "statmod" n'est pas installé
library(statmod)
X <- rinvgauss(n, mean = mu_true, shape = lambda_true)
```

on peut définir, après calculs, la fonction de log-vraisemblance comme suit :

```r
log_likelihood <- function(mu, lambda){
  value <- (n/2)*(log(lambda)-log(2/pi))-1.5*sum(log(X))-lambda/(2*mu^2)*sum((X-mu)^2/X)
  return(value)
}
```

Puisque la log-vraisemblance est bivariée dans ce cas, on propose de la représenter en deux dimensions via un graphique de ses courbes de niveaux. On aurait également pu réaliser un graphique 3D, mais ce type de représentation nous semble moins informative. Le graphique des courbes de niveaux peut être obtenu comme suit :

```r
log_likelihood_vec <- Vectorize(FUN = log_likelihood) #vectorisation pour appliquer outer()
mu_grid <- seq(from = 1, to = 20, by = 0.01)
lambda_grid <- seq(from = 1, to = 20, by = 0.01)
log_likelihood_grid <- outer(X = mu_grid, Y = lambda_grid, FUN = log_likelihood_vec)
filled.contour(x = mu_grid, y = lambda_grid, z = log_likelihood_grid, nlevels = 30)
```



Le maximum de la fonction semble être atteint dans le coin inférieur gauche du graphique et la fonction semble décroissante quand on s'éloigne de ce point. Ainsi, la représentation graphique tend à nous faire penser que la log-vraisemblance est effectivement strictement concave. Afin de vérifier cela analytiquement, on calcule la matrice Hessienne de la log-vraisemblance ℓ_n qui est donnée par :

$$\nabla^2 \ell_n(\mu, \lambda) = \begin{pmatrix} \frac{-n\lambda}{\mu^3}\left(\frac{3\overline{X}_n}{\mu} - 2\right) & \frac{n}{\mu^2}\left(\frac{\overline{X}_n}{\mu} - 1\right) \\ \frac{n}{\mu^2}\left(\frac{\overline{X}_n}{\mu} - 1\right) & \frac{-n}{2\lambda^2} \end{pmatrix}.$$

Ainsi, par le critère du syllabus, la log-vraisemblance n'est pas strictement concave puisque le déterminant n'est pas positif pour toute valeur de (μ, λ). Par contre, on voit facilement que la matrice Hessienne est définie négative pour $\mu = \overline{X}_n$ (qui est le MLE de μ, voir point (d)), et il suit, par une extension multivariée de la note sur l'optimisation, qu'il existe un unique maximum global donné par l'unique solution aux équations de premier ordre.

(b) On sait de la théorie que

$$(\mu_1, \mu_2) := (\mathrm{E}[X_1], \mathrm{E}[X_1^2]) = (\mu, \mu^3/\lambda + \mu^2) =: g(\mu, \lambda),$$

où $g : (0, \infty)^2 \to D := \{(y_1, y_2) \in (0, \infty)^2 : y_2 > y_1^2\}$ est inversible d'inverse donné par $(y_1, y_2) \in D \mapsto g^{-1}(y_1, y_2) = (y_1, y_1^3/(y_2 - y_1^2))$. Il suit que l'estimateur des moments de (μ, λ) est donné

$$(\hat{\mu}_{\mathrm{MM}}, \hat{\lambda}_{\mathrm{MM}}) = g^{-1}(\overline{X}_n, \overline{X^2}_n) = \left(\overline{X}_n, \overline{X}_n^3/(\overline{X^2}_n - \overline{X}_n^2)\right).$$

Le calcul sur l'échantillon donne :

```
param_MoM <- c(mean(X), mean(X)^3/(mean(X^2) - mean(X)^2))
param_MoM
```

```
## [1] 2.243875 5.513912
```

(c) Afin d'utiliser la fonction `optim()`, puisque cette dernière effectue une minimisation, on définit l'opposé de la fonction de log-vraisemblance, qui attendra son minimum au point qui maximise la log-vraisemblance :

```
n_log_likelihood <- function(param){
  value <- -log_likelihood(mu = param[1], lambda = param[2])
  return(value)
}
```

Notez qu'on définit la fonction comme fonction d'un unique paramètre multivarié afin d'utiliser `optim()`.

```
optim(par = param_MoM, fn = n_log_likelihood, method = "L-BFGS-B")
```

```
## $par
## [1] 2.243875 5.276266
##
## $value
## [1] 38.76751
##
## $counts
## function gradient
##        8        8
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

L'algorithme converge puisque l'élément "convergence" de la liste obtenue en réponse à l'utilisation de `optim()` contient le chiffre 0. La valeur obtenue pour le maximum de vraisemblance de nos données est alors donné par

```
param_optim <- optim(par = param_MoM, fn = n_log_likelihood, method = "L-BFGS-B")$par
param_optim
```

```
## [1] 2.243875 5.276266
```

Notons que l'estimateur des moments était déjà très proche du MLE dans ce cas-ci.

(d) Comme discuté au point (a), le MLE est ici donné par l'unique solution des équations de premier ordre. Un calcul montre que celles-ci sont données par

$$\hat{\mu}_{\text{MLE}} = \overline{X}_n \qquad \text{et} \qquad \hat{\lambda}_{\text{MM}} = \frac{n}{\sum_{i=1}^{n}\left(\frac{1}{X_i} - \frac{1}{\overline{X}_n}\right)}.$$

On peut alors calculer ceux-ci sur nos données et mesurer la source de l'erreur entre les vrais paramètres sous-jacents à la génération de nos données et les sorties de notre algorithme d'optimisation :

```
# Calcul analytique du MLE
param_MLE <- c(mean(X), n/(sum(1/X) - n/mean(X)))
param_MLE
```

```
## [1] 2.243875 5.276266
```

```
# Erreur numérique absolue
abs(param_MLE-param_optim)
```

```
## [1] 4.456594e-07 6.329793e-08
```

```r
# Erreur statistique absolue
abs(param_MLE-c(2,5))
```

```
## [1] 0.2438746 0.2762657
```

L'erreur est principalement statistique, dûe au faible nombre de données sous-jacentes à notre vraisemblance.

(e)
```r
# On peut récuperer la hessienne au point param_optim via le paramètre "hessian" :
hessian <- optim(par = param_MoM, fn = n_log_likelihood,
                 method = "L-BFGS-B", hessian = TRUE)$hessian
hessian
```

```
##              [,1]         [,2]
## [1,] 4.670156e+01 8.881784e-10
## [2,] 8.881784e-10 1.796043e+00
```

```r
# On déduit l'inverse de l'information de Fisher. Attention à ne pas mettre de "-"
# car ici la fonction passée dans optim() est déjà l'opposé de la log-lik
solve(hessian)
```

```
##               [,1]          [,2]
## [1,]  2.141256e-02 -1.058893e-11
## [2,] -1.058893e-11  5.567796e-01
```

(f) La matrice d'information de Fisher I_n est donnée par

$$I_n(\mu, \lambda) = \begin{pmatrix} \frac{n\lambda}{\mu^3} & 0 \\ 0 & \frac{n}{2\lambda^2} \end{pmatrix},$$

d'où l'on déduit que pour les vraies valeurs des paramètres, la variance asymptotique du MLE est donnée par

$$I_n^{-1}(2,5) = \begin{pmatrix} 0.04 & 0 \\ 0 & 0.5 \end{pmatrix}.$$

L'estimation de la variance asymptotique obtenue au point (e) est relativement proche de la vérité, comme on peut le vérifier, par exemple, en comparant les deux matrices par un calcul de la norme de Frobenius entre les deux matrices, qui n'est rien d'autre que la norme Euclidienne du vecteur qui contient les différences ponctuelles entre les éléments des deux matrices :

```r
info_true <- matrix(data = c(0.04,0,0,0.5), nrow = 2, byrow = TRUE)
norm(solve(hessian)-info_true, type = "F")
```

```
## [1] 0.05974457
```

Exercice 5

(a) Rappelons que la vraisemblance profilée en k est simplement donnée par

$$\ell_n^p(k) = \ell_n(k, \hat{\theta}(k)),$$

où $\hat{\theta}(k) = \operatorname{argmax}_\theta \ell_n(k, \theta)$. On calcule que

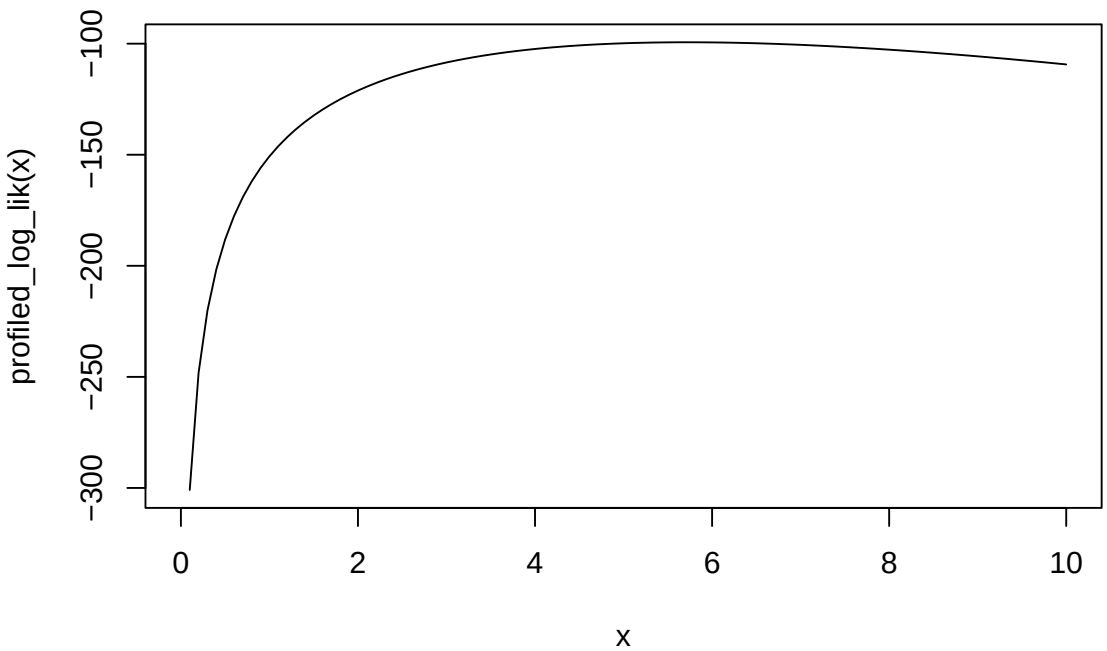
$$\hat{\theta}(k) = \frac{\overline{X}_n}{k},$$

d'où l'on déduit la forme annoncée pour ℓ_n^p. Après avoir exécuté le code dans l'énoncé, que l'on rappelle ici par souci de simplicité,

```r
k_true <- 5
theta_true <- 1/3
set.seed(2023)
n <- 100
X <- rgamma(n, shape = k_true, scale = theta_true)
```

on peut faire un graphique de la fonction de log-vraisemblance profilée comme suit :

```r
# Définition de la fonction :
profiled_log_lik <- function(k){
  value <- -n*log(gamma(k))-n*k*log(mean(X)/k)+(k-1)*sum(log(X))-n*k
  return(value)
}
# Graphe de la fonction :
curve(profiled_log_lik, from = 0, to = 10)
```



(b) Rappelons que l'estimateur des moments de k est donné par

```r
k_MM <- mean(X)^2/(mean(X^2)-mean(X)^2)
k_MM
```

```
## [1] 5.316416
```

Puisque la fonction d'optimisation sous-jacente à la fonction `mle()` est `optim()`, il faut encore une fois donner l'opposé de la fonction de log-vraisemblance profilée comme argument :

```r
n_profiled_log_lik <- function(k){
  value <- -profiled_log_lik(k)
  return(value)
}
```

```r
library(stats4) # chargement du package
k_mle_optim <- mle(n_profiled_log_lik, start = k_MM, method = "L-BFGS-B")
summary(k_mle_optim)
```

```
## Maximum likelihood estimation
##
## Call:
## mle(minuslogl = n_profiled_log_lik, start = k_MM, method = "L-BFGS-B")
##
## Coefficients:
##      Estimate Std. Error
## [1,] 5.715636  0.7858558
##
## -2 log L: 198.7328
```

Noter que la sortie de la fonction `mle()` n'est plus une liste comme la sortie de `optim()` mais un objet de la class `mle`, d'où le changement de syntaxe. La sortie nous fournit également une estimation de l'écart-type de l'estimateur obtenu, via une estimation de sa variance asymptotique basée sur l'information de Fisher obsevée.

(c) Au vue des calculs du point (a), on a que le MLE de (k, θ) est donné par

$$(\hat{k}_{\text{MLE}}, \hat{\theta}_{\text{MLE}}) = (\hat{k}, \hat{\theta}((\hat{k})),$$

où $\hat{k}$ est la valeur de k qui maximise la log-vraisemblance profilée en k. Cela donne :

```r
param_MLE <- c(k_mle_optim@coef, mean(X)/k_mle_optim@coef)
param_MLE
```

```
## [1] 5.7156360 0.2905569
```