TP 3 – SY02
Estimation et théorème central limite
Corrigé

Les questions/sections marquées par un  sont des questions plus ouvertes qui peuvent être abordées dans un deuxième temps.

1 Estimation par la méthode des moments

On modélise un phénomène par une variable aléatoire X suivant une loi uniforme sur l'intervalle $[0, a]$ avec a entier. Afin d'estimer le paramètre a, on cherche un estimateur avec la méthode des moments. On utilisera la fonction suivante pour générer un échantillon de taille n :

```
runifa <- function(n) {
  if(!exists("param")) param <<- sample(10:20, 1)
  runif(n, min = 0, max = param)
}
```

(1) Sachant que $\mathbb{E}(X) = \dfrac{a}{2}$, proposer un estimateur du paramètre a.

On inverse le système en exprimant les paramètres en fonction des moments. On trouve

$$a = 2\,\mathbb{E}(X).$$

On propose donc l'estimateur $\widehat{a} = 2\overline{X}$.

(2) Créer une fonction `estim` qui prend en argument un échantillon de taille quelconque issu de X et renvoie l'estimation du paramètre a.

```
>runifa <- function(n) {
+if (!exists("param"))
+  param <<- sample(10:20, 1)
+runif(n, min = 0, max = param)
+}
>estim <- function(x) {
+2 * mean(x)
+}
```

③ Lancer plusieurs fois la fonction précédente avec un échantillon de taille 100. Quel semble être le paramètre a ?

```
>n <- 100
>estim(runifa(n))
[1] 15.77133
>estim(runifa(n))
[1] 16.16566
>estim(runifa(n))
[1] 14.46068
>estim(runifa(n))
[1] 14.51449
```

Le paramètre entier inconnu semble être ici 15 (il est différent à chaque nouvelle session).

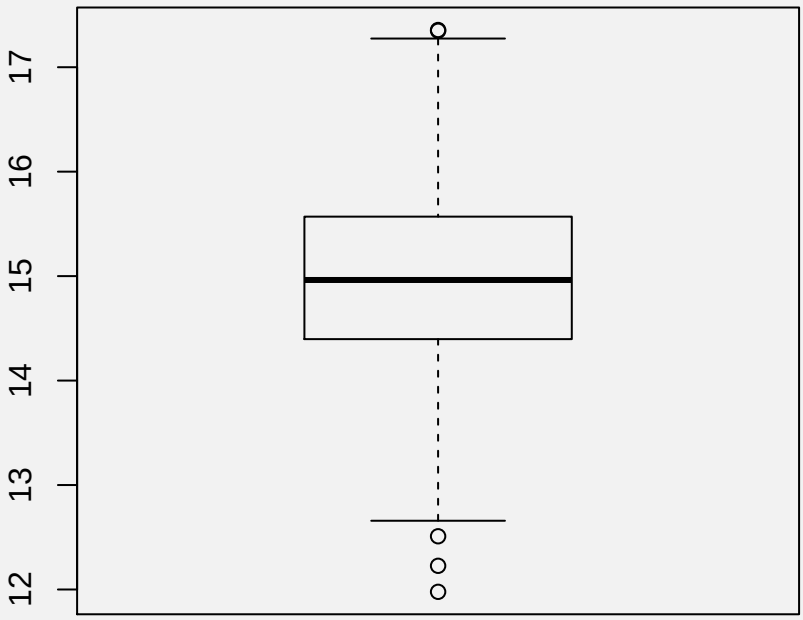
Au lieu d'exécuter manuellement plusieurs fois l'instruction `estim(runifa(n))`, on va utiliser une fonction `R` qui le fait à notre place et accumule les différents résultats : il s'agit de la fonction **`replicate`** qu'on utilise comme suit :

```
a <- replicate(1000, estim(runifa(n)))
```

Le vecteur `a` stocke les 1000 résultats de l'instruction `estim(runifa(n))`.

④ Faire un diagramme en boîte de 1000 estimations successives de `a`. Quel est le paramètre inconnu ? Vérifier qu'il est en accord avec le vrai paramètre choisi au hasard, utilisé pour générer les observations et stocké dans `param`.

```
>a <- replicate(1000, estim(runifa(n)))
>boxplot(a)
```



Le paramètre recherché est donc 15.

```
>param
[1] 15
```

⑤ On admet que les moments de X sont :

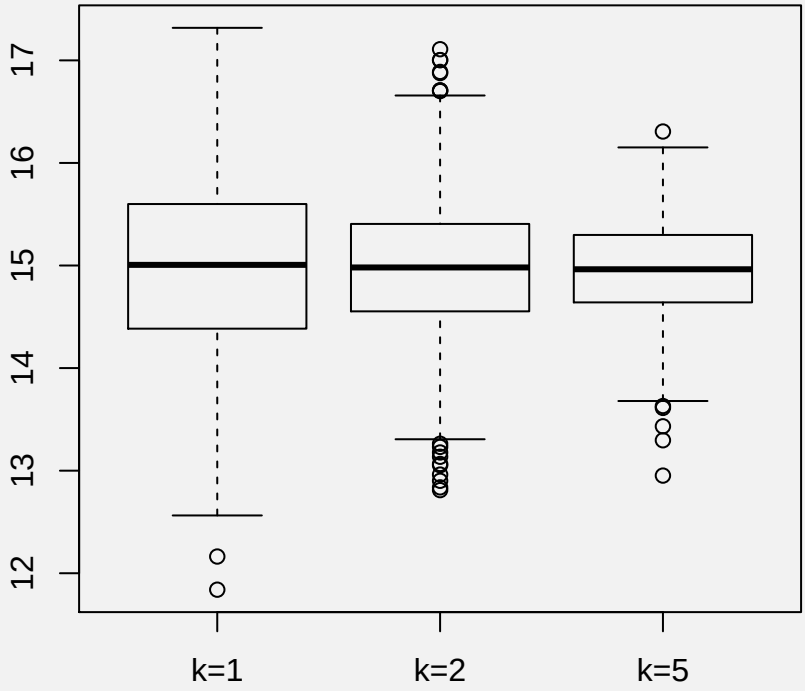
$$m_k = \mathbb{E}(X^k) = \frac{a^k}{k+1} \quad \text{pour tout} \quad k \geq 1.$$

Trouver pour chaque moment l'estimateur $\hat{a}_k$ correspondant. Refaire l'étude précédente. Quel est l'estimateur le plus précis ?

Les estimateurs sont :

$$\hat{a}_k = ((k+1)\hat{m}_k)^{(1/k)}$$

```
>estimk <- function(x, k) {
+((k + 1) * mean(x^k))^(1/k)
+}
>a1 <- replicate(1000, estimk(runifa(n), 1))
>a2 <- replicate(1000, estimk(runifa(n), 2))
>a5 <- replicate(1000, estimk(runifa(n), 5))
>boxplot(a1, a2, a5, names = c("k=1", "k=2", "k=5"))
```



La variance des estimateurs baisse. Lorsque k est grand, on trouve l'estimateur $\max_i X_i$.

2 Théorème central limite

On souhaite vérifier expérimentalement le théorème central limite. Pour cela, on va choisir une variable aléatoire X de loi quelconque, d'espérance μ et d'écart-type σ. On utilisera la fonction suivante pour générer un échantillon de loi inconnue de taille n :

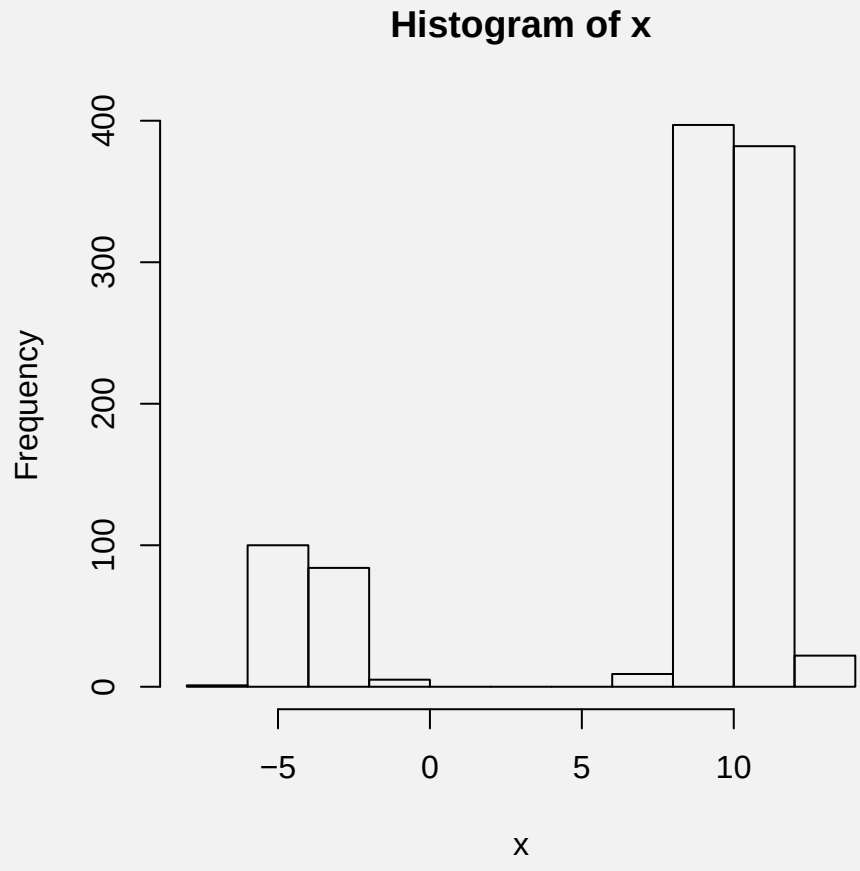
```
runknown <- function(n) {
  bn <- rbinom(n, 1, 0.2)
  bn * rnorm(n, mean=-4, sd=1) + (1 - bn) * rnorm(n, mean=10, sd=1)
}
```

⑥ Vérifier expérimentalement que l'espérance de X vaut 7.2 et que l'écart-type vaut $\sqrt{32.36}$.

```
>runknown <- function(n) {
+bn <- rbinom(n, 1, 0.2)
+bn * rnorm(n, mean = -4, sd = 1) + (1 - bn) * rnorm(n, mean = 10, sd = 1)
+}
>n <- 1000
>x <- runknown(n)
>mean(x)
[1] 7.285805
>sd(x)
[1] 5.594852
>sqrt(32.36)
[1] 5.688585
```

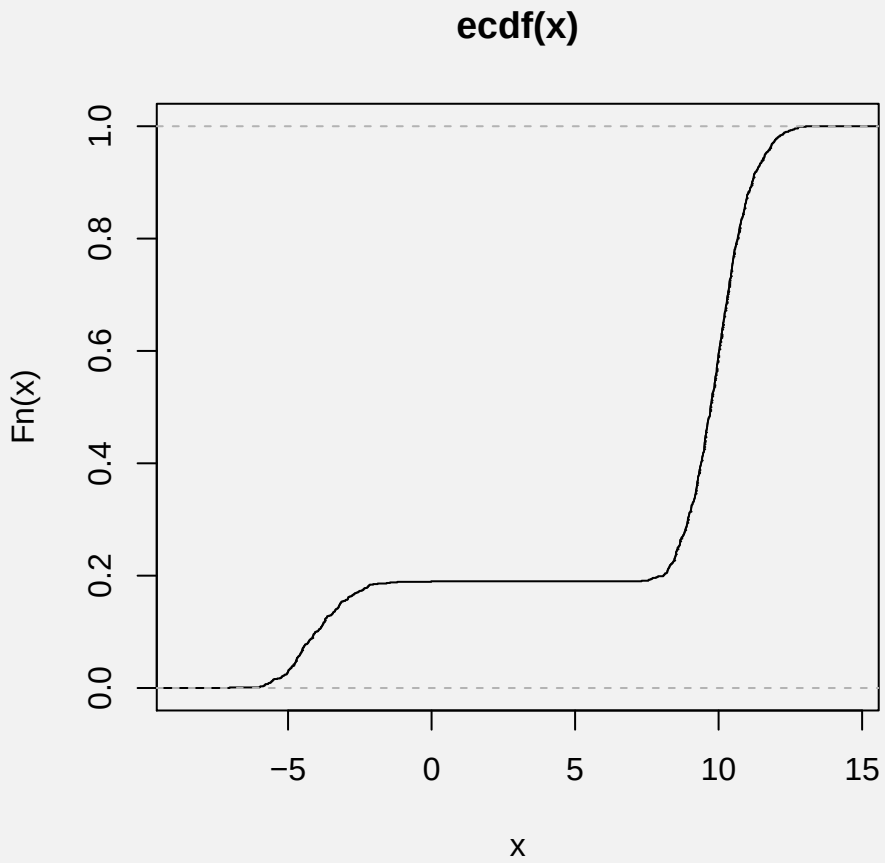
⑦ Tracer un histogramme d'un échantillon de taille 1000.

```
>n <- 1000
>x <- runknown(n)
>hist(x)
```



⑧ Tracer la fonction de répartition empirique de la loi inconnue avec un échantillon de taille 1000 à l'aide des fonctions `ecdf` et `plot`.

```
>plot(ecdf(x))
```



La fonction `ecdf` est un peu spéciale : elle renvoie elle-même une fonction. Pour calculer la fonction de répartition empirique en $x = 2$, on écrit donc

```
>ecdf(x)(2)
[1] 0.19
```

Le théorème central limite dit que si on a n variables aléatoires échantillons indépendantes $X_1, \ldots, X_n$ de loi parente celle de X, alors la variable aléatoire suivante,

$$T = \frac{\overline{X} - \mu}{\sigma/\sqrt{n}},$$

converge en loi lorsque n augmente vers une loi normale centrée réduite. On va donc vérifier que des réalisations issues de la loi de T sont distribuées comme une gaussienne centrée réduite.

(9) À partir de l'échantillon de taille n de loi celle de X, calculer une seule réalisation de la variable aléatoire T.

```
>mu <- 7.2
>sigma <- sqrt(32.36)
>(mean(x) - mu)/(sigma/sqrt(n))
[1] 0.9155237
```

(10) Réitérer l'opération précédente plusieurs fois en regénérant à chaque fois un nouvel échantillon de taille n de X. À quoi semble ressembler la distribution obtenue ?

```
>x <- runknown(1000)
>(mean(x) - mu)/(sigma/sqrt(n))
[1] 1.482528
>x <- runknown(1000)
>(mean(x) - mu)/(sigma/sqrt(n))
[1] 0.9107382
>x <- runknown(1000)
>(mean(x) - mu)/(sigma/sqrt(n))
[1] 1.902527
>x <- runknown(1000)
>(mean(x) - mu)/(sigma/sqrt(n))
[1] 0.1926825
```

La distribution semble se concentrer autour de 0 avec un faible écart type.

Pour générer un nombre quelconque de réalisations de T, on va à nouveau utiliser la fonction **replicate**. Il faut d'abord créer une fonction qui regroupe le calcul de la réalisation et retourne le résulat.

```
random.T <- function(n) {
  # Générer le vecteur x de taille n
  # Calculer une réalisation de la loi T
  return(valeur)
}
```

Pour avoir une réalisation de la loi T, il suffit maintenant d'appeler la fonction `random.T` comme ceci

```
random.T(n)
```

Ensuite, pour appeler `random.T` un nombre quelconque de fois et stocker les différents résulats dans un vecteur, on exécute

```
t.10 <- replicate(10, random.T(n))
```

qui stocke dans le vecteur `t.10` 10 appels successifs de la fonction `random.T` avec l'argument n.

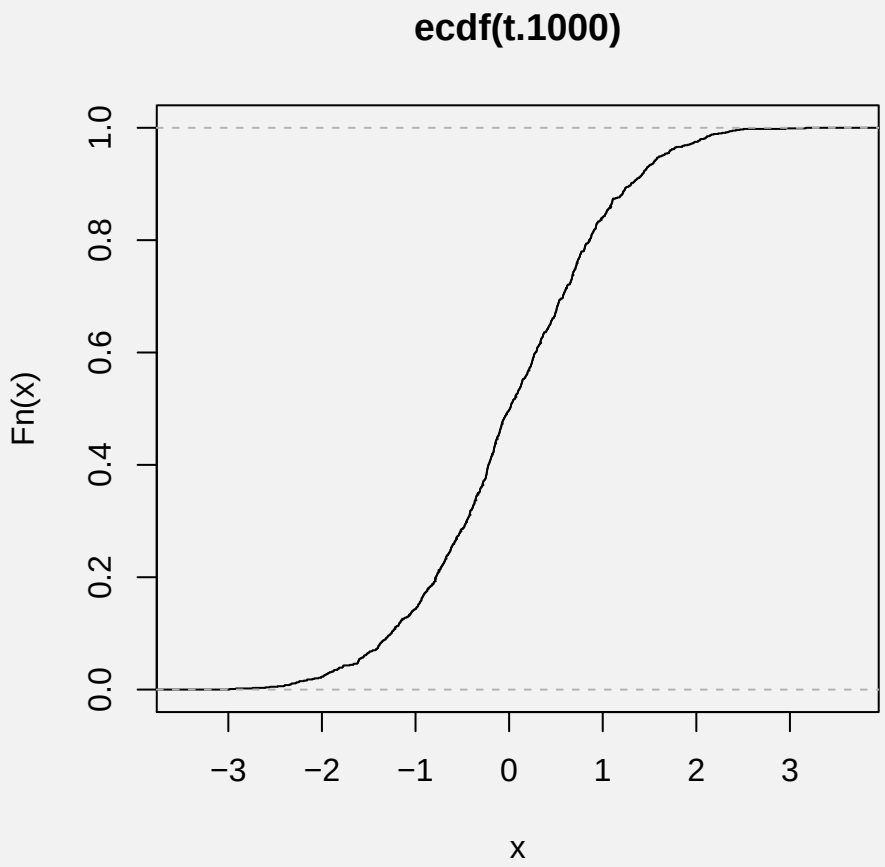
(11) Définir la fonction `random.T` et générer un vecteur `t.1000` contenant 1000 réalisations de la variable aléatoire T. Vérifier que la moyenne empirique et la variance empirique sont en accord avec une loi normale centrée réduite.

```
>random.T <- function(n) {
+x <- runknown(n)
+(mean(x) - mu)/(sigma/sqrt(n))
+}
>t.1000 <- replicate(1000, random.T(n))
>mean(t.1000)
[1] -0.007138456
>var(t.1000)
[1] 1.048013
```

La moyenne empirique semble nulle et la variance empirique égale à 1. On observe bien des estimations compatibles avec une loi normale centrée réduite.

(12) Tracer la fonction de répartition empirique de l'échantillon `t.1000`. Comparer avec celle d'un échantillon `x` vu à la question 8.

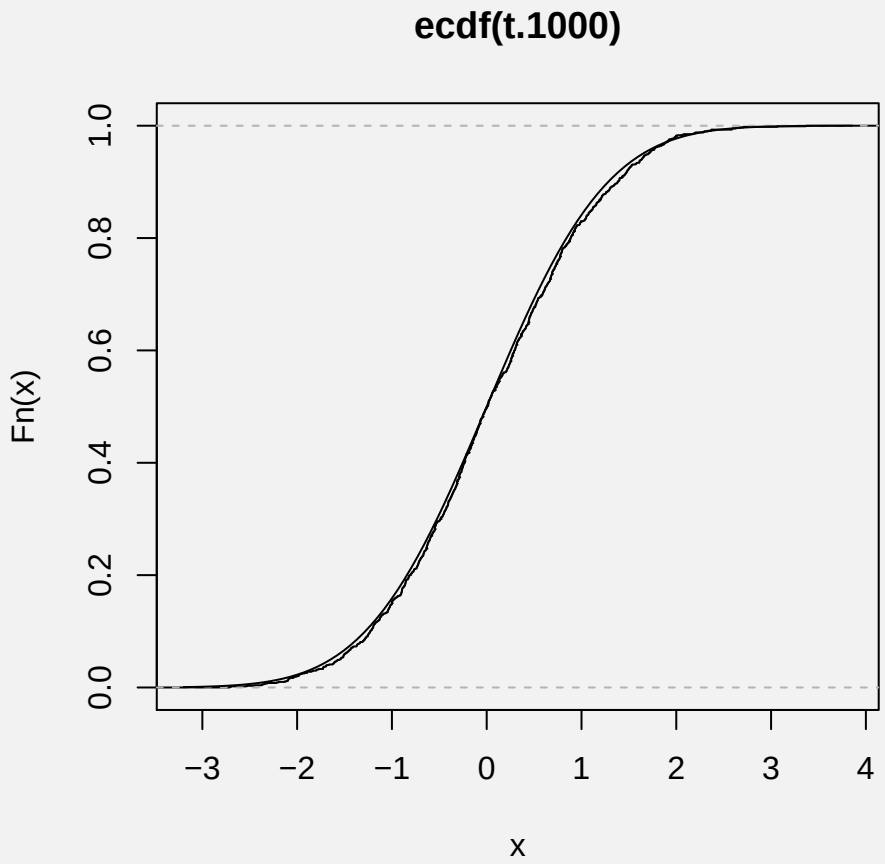
```
>t.1000 <- replicate(1000, random.T(n))
>plot(ecdf(t.1000))
```



La fonction de répartition de T pour $n = 1000$ est lissée et ressemble à celle d'une gaussienne centrée réduite.

(13) Avec l'instruction `curve(pnorm, add=TRUE)`, superposer au graphe précédent, la fonction de répartition théorique d'une gaussienne centrée réduite. Qu'observez-vous ?

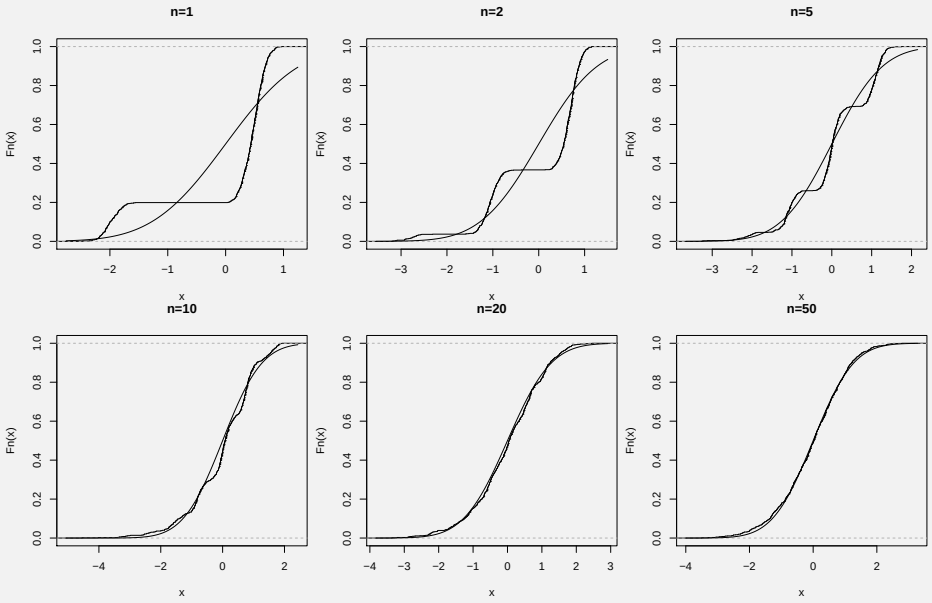
```
>t.1000 <- replicate(1000, random.T(n))
>plot(ecdf(t.1000))
>curve(pnorm, add = TRUE)
```



La variable aléatoire T est distribuée comme une variable normale centrée réduite.

(14) Superposer à nouveau les deux courbes pour des valeurs de n valant 1, 2, 5, 10, 20, 50. Qu'observez-vous ?

```
>Ts <- replicate(1000, random.T(1))
>plot(ecdf(Ts), main = "n=1")
>curve(pnorm, add = TRUE)
>Ts <- replicate(1000, random.T(2))
>plot(ecdf(Ts), main = "n=2")
>curve(pnorm, add = TRUE)
>Ts <- replicate(1000, random.T(5))
>plot(ecdf(Ts), main = "n=5")
>curve(pnorm, add = TRUE)
>Ts <- replicate(1000, random.T(10))
>plot(ecdf(Ts), main = "n=10")
>curve(pnorm, add = TRUE)
>Ts <- replicate(1000, random.T(20))
>plot(ecdf(Ts), main = "n=20")
>curve(pnorm, add = TRUE)
>Ts <- replicate(1000, random.T(50))
>plot(ecdf(Ts), main = "n=50")
>curve(pnorm, add = TRUE)
```



On observe une convergence simple de la fonction de répartition empirique vers la fonction de répartition d'une loi normale centrée réduite. On a donc bien convergence en loi.

3 Estimation par maximum de vraisemblance

On considère la loi exponentielle de paramètre $\lambda > 0$ et de densité

$$f_\lambda(x) = \begin{cases} \lambda e^{-\lambda x} & \text{si } x \geq 0 \\ 0 & \text{sinon,} \end{cases}$$

disponible en `R` avec la function `dexp`. On fixe le paramètre $\lambda = 3$ et on suppose qu'on dispose d'un échantillon iid $x_1, \ldots, x_n$ issu de cette loi. On cherche à présent à estimer le paramètre λ qu'on suppose à présent inconnu avec la méthode du maximum de vraisemblance.

(15) Créer la fonction `f` de densité qui prend en argument un paramètre λ et un échantillon x et renvoie les densités aux points `x` pour la loi exponentielle de paramètre λ.

```
>f <- function(lambda, x) {
+dexp(x, rate = lambda)
+}
```

16 Créer la fonction `L` de vraisemblance qui prend en argument un paramètre λ et un échantillon x et renvoie la vraisemblance du paramètre λ par rapport aux données `x`. On pourra utiliser la fonction **`prod`**.

```
>L <- function(lambda, x) {
+prod(f(lambda, x))
+}
```

17 Créer la fonction `logL` de log–vraisemblance qui prend en argument un paramètre λ et un échantillon x et renvoie la log–vraisemblance du paramètre λ par rapport aux données `x`. Pour des raisons de stabilité, on utilisera le fait que le logarithme d'un produit est la somme des logarithmes [1].

```
>logL <- function(lambda, x) {
+sum(log(f(lambda, x)))
+}
```

18 Stocker dans la variable `x` un échantillon de taille $n = 100$ suivant une loi exponentielle de paramètre $\lambda = 3$. D'après la vraisemblance, quel est le paramètre le plus probable entre $\lambda = 3.1$ et $\lambda = 2.8$?

```
>n <- 100
>x <- rexp(n, rate = 3)
>logL(3.1, x)
[1] 9.436964
>logL(2.8, x)
[1] 9.294493
```

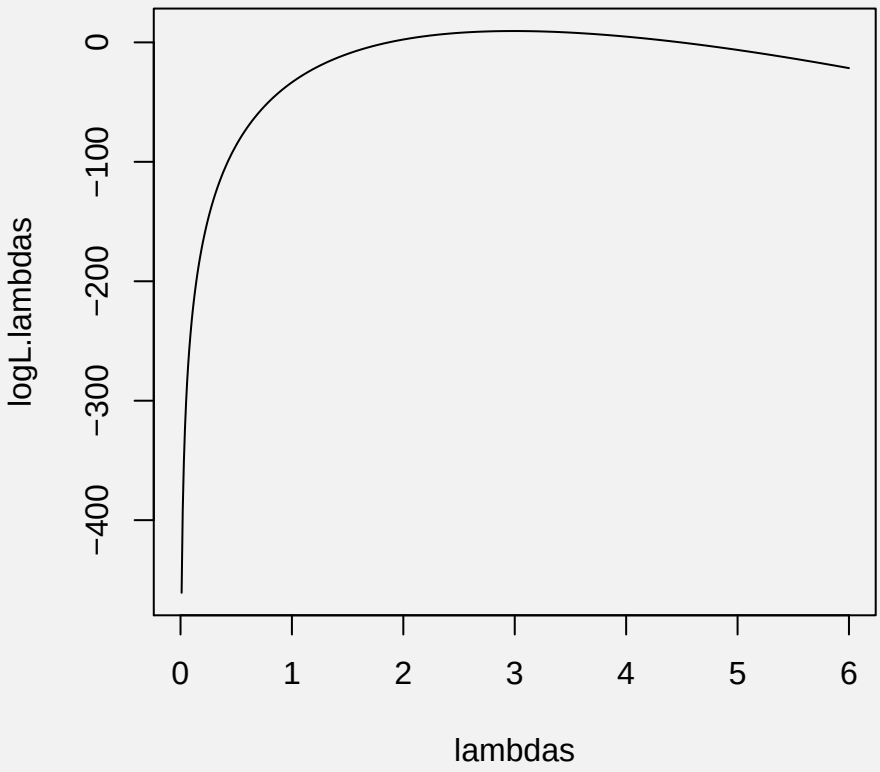
$\lambda = 3.1$ est le paramètre le plus probable pour l'échantillon x.

19 Que fait le bout de code suivant

```
lambdas <- seq(0, 6, 0.01)
logL.lambdas <- sapply(lambdas, function(lambda) logL(lambda, x))
plot(lambdas, logL.lambdas, type = "l")
```

1. Comparer `log(1e-200 * 1e-200)` et `log(1e-200) + log(1e-200)`

```
>lambdas <- seq(0, 6, 0.01)
>logL.lambdas <- sapply(lambdas, function(lambda) logL(lambda, x))
>plot(lambdas, logL.lambdas, type = "l")
```



Le bout de code calcule la log–vraisemblance des `lambdas` et affiche le graphe correspondant. Le maximum de cette fonction de vraisemblance est atteint vers $\lambda = 3$.

La méthode du maximum de vraisemblance consiste à trouver le paramètre λ qui maximise la fonction `logL`. Pour ce faire, nous allons utiliser la fonction `optimize` de `R`. Supposons qu'on souhaite calculer le maximum de la fonction $g(x) = -(x-\pi)^2$ (qui vaut 0 et est atteint pour $x = \pi$). On définit d'abord la fonction g en `R` :

```
>g <- function(x) {
+  -(x - pi)^2
+}
```

On appelle ensuite la fonction `optimize` comme suit :

```
>(opt <- optimize(g, lower = -10, upper = 10, maximum = TRUE))
$maximum
[1] 3.141593

$objective
[1] 0
```

Les arguments `upper` et `lower` fixe l'intervalle de recherche et `maximum` spécifie que l'on recherche un maximum et non un minimum.

La fonction retourne un objet que l'on a pas encore rencontré : une liste nommée. Il s'agit d'une liste contenant des objets qui sont accessibles en fournissant leur nom.

```
>opt$maximum
[1] 3.141593
>opt$objective
[1] 0
```

Attention, si on veut optimiser une fonction à plusieurs arguments (par exemple une fonction de vraisemblance), il ne faut pas oublier de fixer dans la fonction `optimize` les arguments sur lesquels ne porte pas l'optimisation. À titre d'exemple, on pourra consulter la section « Examples » de l'aide sur la fonction `optimize`.

(20) Calculer le paramètre λ le plus vraisemblable par rapport à l'échantillon `x`.

```
>x <- rexp(n, rate = 3)
>opt <- optimize(logL, lower = 0, upper = 6, maximum = TRUE, x = x)
>opt$maximum
[1] 3.040364
```

Le paramètre le plus vraisemblable de la loi exponentielle ayant généré `x` est donc 3.0403643

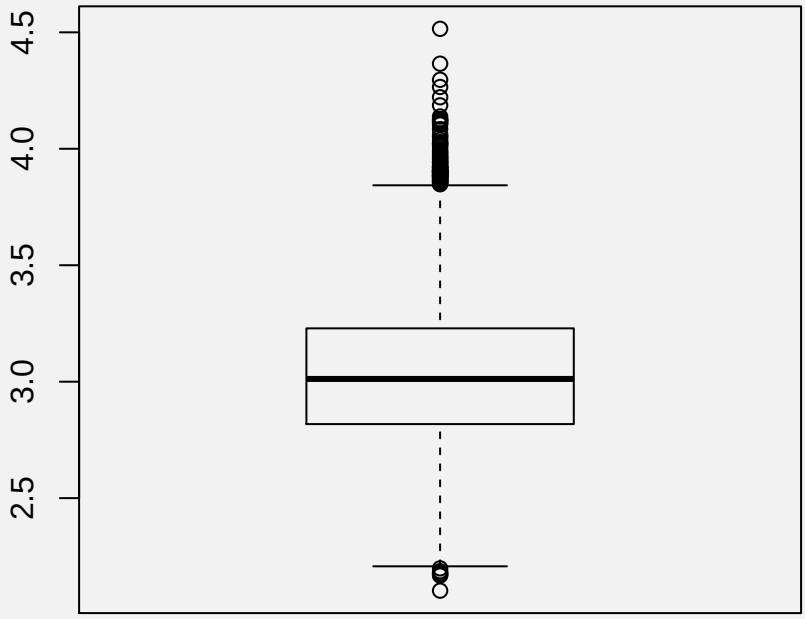
De la même manière que dans la section précédente, on cherche maintenant à recommencer cette estimation pour différents échantillons `x`.

(21) Créer une fonction `sim.EMV` qui ne prend aucun argument, définit un échantillon et calcule le maximum de vraisemblance associé à cet échantillon.

```
> sim.EMV <- function() {
+ x <- rexp(n, 3)
+
+ # Maximization de la log-vraisemblance
+ opt <- optimize(logL, lower = 0, upper = 10, maximum = TRUE, x = x)
+ opt$maximum
+ }
> sim.EMV()
[1] 2.55151
> sim.EMV()
[1] 2.97365
```

(22) Créer un vecteur contenant le résultat de 10000 simulations avec la fonction **`replicate`** et `sim.EMV`.

```
>sim.EMV.10000 <- replicate(10000, sim.EMV())
>boxplot(sim.EMV.10000)
```



(23) Estimer l'espérance et la variance de cet estimateur en calculant la moyenne empirique et la variance empirique de réalisations de cet estimateur.

```
>mean(sim.EMV.10000)
[1] 3.032272
>var(sim.EMV.10000)
[1] 0.09399897
```

(24) Comparer l'estimation du biais avec le biais théorique dont on admet qu'il vaut

$$\frac{n}{n-1}\lambda - \lambda.$$

```
># Estimation du biais
>mean(sim.EMV.10000) - 3
[1] 0.03227171
># Valeur théorique du biais
>n/(n - 1) * 3 - 3
[1] 0.03030303
```

4 Information de Fisher

Nous allons à présent calculer une valeur approchée de l'information de Fisher pour le paramètre $\lambda = 3$. Pour calculer l'information de Fisher, il faut pouvoir calculer la dérivée

de la log–vraisemblance au point $\lambda = 3$. Pour ce faire, nous allons faire appel à une fonction appartenant à une bibliothèque qui n'est pas installée par défaut. Pour installer une bibliothèque il suffit d'exécuter l'instruction

```
install.packages("ma-bibliothèque")
```

puis, pour charger la bibliothèque, on exécute

```
library(ma-bibliothèque)
```

(25) Installer puis charger la bibliothèque `pracma`.

```
>install.packages("pracma")
>library(pracma)
```

Nous allons utiliser la fonction `grad` de la bibliothèque `pracma`. Elle calcule la dérivée d'une fonction en un point. Si on reprend l'exemple précédent avec la fonction g :

```
>grad(g, 0)
[1] 6.283185
```

On trouve bien $g'(0) = 2\pi$.

(26) Créer une fonction `sim.Fisher` sans argument qui génère un échantillon `x` et calcule l'information de Fisher de cet échantillon pour $\lambda = 3$.

```
> sim.Fisher <- function() {
+ x <- rexp(n, rate = 3)
+
+ # Log-vraisemblance par rapport à x
+ logLx <- function(lambda) logL(lambda, x)
+
+ # Information de Fisher
+ (grad(logLx, 3))^2
+ }
```

(27) Encore une fois, utiliser `sim.Fisher` et `replicate` pour simuler 1000 fois le calcul de l'information de Fisher d'un échantillon et donner une estimation de l'information de Fisher.

```
>(inf.Fisher <- mean(replicate(10000, sim.Fisher())))
[1] 11.12836
```

(28) Comparer la valeur théorique de l'information de Fisher donnée par,

$$I_n(\lambda) = \frac{n}{\lambda^2}.$$

```
>n/3^2
[1] 11.11111
```

(29) Sachant que l'estimateur du maximum de vraisemblance est asymptotiquement normal, estimer sa variance à l'aide de l'information de Fisher. Comparer le résultat obtenu avec la variance empirique de l'estimateur.

```
>(1/inf.Fisher)
[1] 0.08986048
>var(sim.EMV.10000)
[1] 0.09399897
```

(30) Retrouver expérimentalement le fait que,

$$I_n(\lambda) = -\mathbb{E}\left[\frac{\partial^2 \log \mathcal{L}}{\partial \lambda^2}(\lambda; X_1, \ldots, X_n)\right].$$

On pourra d'abord définir la fonction `grad2` prenant en argument une fonction `f` et un point `x` et qui calcule la dérivée seconde de la fonction `f` au point `x`.

```
> n <- 100
> grad2 <- function(f, x) {
+ df <- function(x) {
+   grad(f, x)
+ }
+ grad(df, x)
+ }
> sim.Fisher <- function() {
+ x <- rexp(n, 3)
+
+ # Log-vraisemblance par rapport à x
+ logLx <- function(lambda) logL(lambda, x)
+
+ # Information de Fisher
+ grad2(logLx, 3)
+ }
> (-mean(replicate(1000, sim.Fisher())))
[1] 11.11114
```