Fiche TD avec le logiciel  : tdr76

# Calcul symbolique

J.R. Lobry

Comment calculer une dérivée, application aux fonctions de sensibilité

## 1 Introduction

Attention,  n'est pas fait pour faire du calcul formel. Il vaut mieux pour cela utiliser le logiciel libre `Maxima` (`http://maxima.sourceforge.net/`) ou un de ses concurrent dans le monde commercial (Mathematica ou Maple).

## 2 Calcul de dérivées

On peut calculer les dérivées d'expression du langage, par exemple :

```
D(exp = expression(x^2), name = "x")
```

```
2 * x
```

Voyons si  connaît les dérivées usuelles :

### 2.1 $(x^n)' = nx^{n-1},\ n \in \mathbb{R}$

```
D(expression(x^n), "x")
```

```
x^(n - 1) * n
```

### 2.2 $(\sqrt{x})' = \frac{1}{2\sqrt{x}}$

```
D(expression(sqrt(x)), "x")
```

```
0.5 * x^-0.5
```

## 2.3 $\left(\frac{1}{x}\right)' = \frac{-1}{x^2}$

```
D(expression(1/x), "x")
```

```
-(1/x^2)
```

## 2.4 $(\mathrm{e}^{ax})' = a\mathrm{e}^{ax},\ a \neq 0$

```
D(expression(exp(a * x)), "x")
```

```
exp(a * x) * a
```

## 2.5 $(\ln(x))' = \frac{1}{x}$

```
D(expression(log(x)), "x")
```

```
1/x
```

## 2.6 $(\cos x)' = -\sin x$

```
D(expression(cos(x)), "x")
```

```
-sin(x)
```

## 2.7 $(\sin x)' = \cos x$

```
D(expression(sin(x)), "x")
```

```
cos(x)
```

## 2.8 $(\tan x)' = 1 + \tan^2 x = \frac{1}{\cos^2 x}$

```
D(expression(tan(x)), "x")
```

```
1/cos(x)^2
```

## 2.9 $(\cosh x)' = \sinh x$

```
D(expression(cosh(x)), "x")
```

```
sinh(x)
```

## 2.10 $(\sinh x)' = \cosh x$

```
D(expression(sinh(x)), "x")
```

```
cosh(x)
```

## 2.11 $(\tanh x)' = 1 - \tanh^2 x = \frac{1}{\cosh^2 x}$

```
D(expression(tanh(x)), "x")
```

```
1/cosh(x)^2
```

## 2.12 $(\arcsin x)' = \frac{1}{\sqrt{1-x^2}}$

```
D(expression(asin(x)), "x")
```

```
1/sqrt(1 - x^2)
```

## 2.13 $(\arccos x)' = \frac{-1}{\sqrt{1-x^2}}$

```
D(expression(acos(x)), "x")
```

```
-(1/sqrt(1 - x^2))
```

## 2.14 $(\arctan x)' = \frac{1}{1+x^2}$

```
D(expression(atan(x)), "x")
```

```
1/(1 + x^2)
```

## 2.15 $(\alpha f(x) + \beta g(x))' = \alpha f'(x) + \beta g'(x)$

```
D(expression(alpha * cos(x) + beta * sin(x)), "x")
```

```
beta * cos(x) - alpha * sin(x)
```

## 2.16 $(f(x)g(x))' = f'(x)g(x) + f(x)g'(x)$

```
D(expression(cos(x) * sin(x)), "x")
```

```
cos(x) * cos(x) - sin(x) * sin(x)
```

**2.17** $\left(\frac{1}{f(x)}\right)' = \frac{-f'(x)}{(f(x))^2}$

```
D(expression(1/sin(x)), "x")
```

```
-(cos(x)/sin(x)^2)
```

**2.18** $\left(\frac{f(x)}{g(x)}\right)' = \frac{-f'(x)g(x)-f(x)g'(x)}{(g(x))^2}$

```
D(expression(cos(x)/sin(x)), "x")
```

```
-(sin(x)/sin(x) + cos(x) * cos(x)/sin(x)^2)
```

**2.19** $((g \circ f)(x))' = f'(x)g'(f(x))$

```
D(expression(cos(sin(x))), "x")
```

```
-(sin(sin(x)) * cos(x))
```

Donc R connaît ses classiques.

## 2.20 Utiliser une dérivée symbolique

La fonction `D()` renvoie un objet de mode `call` que l'on peut évaluer avec la fonction `eval()` :

```
dsin <- D(expression(sin(x)), "x")
dsin
```

```
cos(x)
```

```
mode(dsin)
```

```
[1] "call"
```

```
x <- pi
eval(dsin)
```

```
[1] -1
```

```
x <- seq(-1, 1, length = 10)
eval(dsin)
```

```
 [1] 0.5403023 0.7124746 0.8496076 0.9449569 0.9938335 0.9938335 0.9449569 0.8496076
 [9] 0.7124746 0.5403023
```

On peut transformer cet objet de mode `call` en une fonction ainsi :

```
fdsin <- function(x) eval(dsin)
fdsin(pi)
```

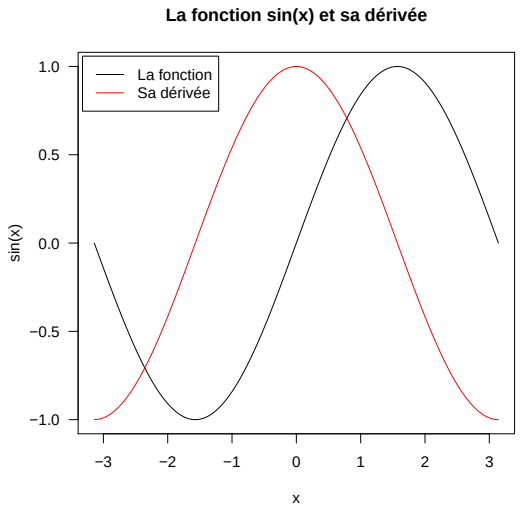
```
[1] -1
```

```
fdsin(x)
```

```
[1] 0.5403023 0.7124746 0.8496076 0.9449569 0.9938335 0.9938335 0.9449569 0.8496076
[9] 0.7124746 0.5403023
```

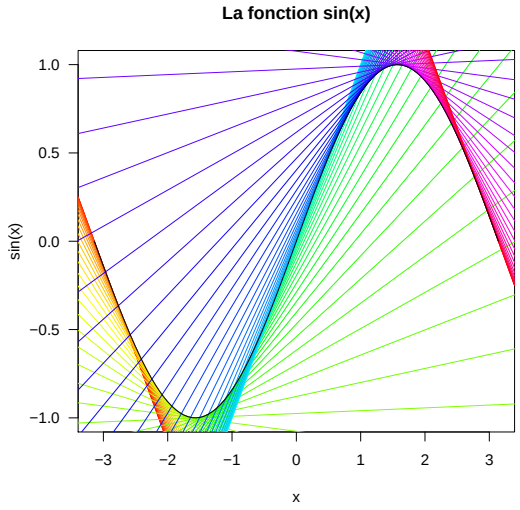
Représenter la fonction et sa dérivée :

```
x <- seq(-pi, pi, length = 100)
plot(x, sin(x), type = "l", las = 1, main = "La fonction sin(x) et sa dérivée")
points(x, fdsin(x), type = "l", col = "red")
legend("topleft", inset = 0.01, lty = 1, col = c("black", "red"),
    legend = c("La fonction", "Sa dérivée"))
```




Représenter les tangentes à la courbe :

```
n <- 100
x <- seq(-pi, pi, length = n)
plot(x, sin(x), type = "l", las = 1, main = "La fonction sin(x)")
for (i in 1:n) {
    abline(a = sin(x[i]) - fdsin(x[i]) * x[i], b = fdsin(x[i]),
        col = rainbow(n)[i])
}
points(x, sin(x), type = "l", col = "black")
```




# 3 Exemple d'application au calcul des fonctions de sensibilité

## 3.1 Une première expérience

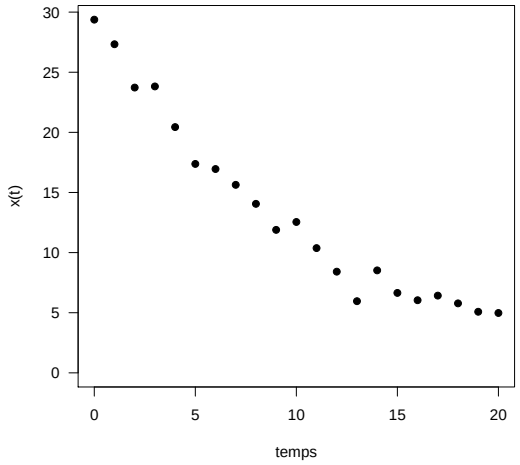
Soit un modèle exponentiel décroissant :

$$x(t) = x_0 \mathrm{e}^{-\mu t}$$

On fait une première expérience :

```
set.seed(1)
bruit <- rnorm(21)
data <- 30 * exp(-0.1 * (0:20)) + bruit
```

```
plot(0:20, data, pch = 19, xlab = "temps", ylab = "x(t)", las = 1,
    ylim = c(0, max(data)))
```




On veut estimer les paramètres $x_0$ et $\mu$ du modèle à partir de ces données. On définit une fonction qui retourne la somme des carrés des écarts :

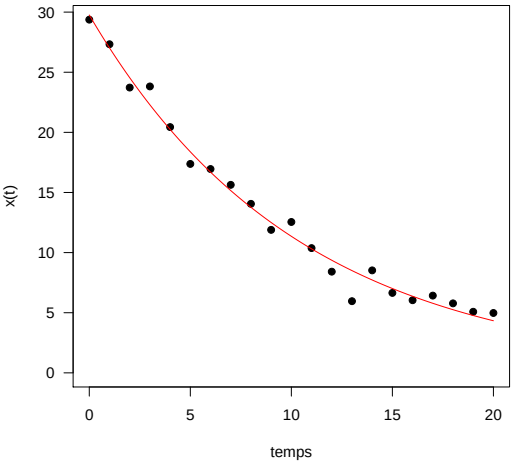
```
sce <- function(p) {
    x0 <- p[1]
    mu <- p[2]
    sum((data - x0 * exp(-mu * (0:20)))^2)
}
```

On cherche la valeur des paramètres qui la minimise :

```
nlmfit <- nlm(f = sce, p = c(10, 0.1))
nlmfit
```

```
$minimum
[1] 15.93465

$estimate
[1] 29.7606677  0.0963585

$gradient
[1] -1.735956e-05  1.287420e-03

$code
[1] 2

$iterations
[1] 15
```

```
plot(0:20, data, pch = 19, xlab = "temps", ylab = "x(t)", las = 1,
    ylim = c(0, max(data)))
tseq <- seq(0, 20, length = 100)
points(tseq, nlmfit$estimate[1] * exp(-nlmfit$estimate[2] * tseq),
    type = "l", col = "red")
```




Une région de confiance pour la valeur des paramètres avec un risque de première espèce $\alpha$ est donnée [1] par l'ensemble des valeurs des paramètres telles que la somme des carrés des résidus n'excède pas un seuil donné,

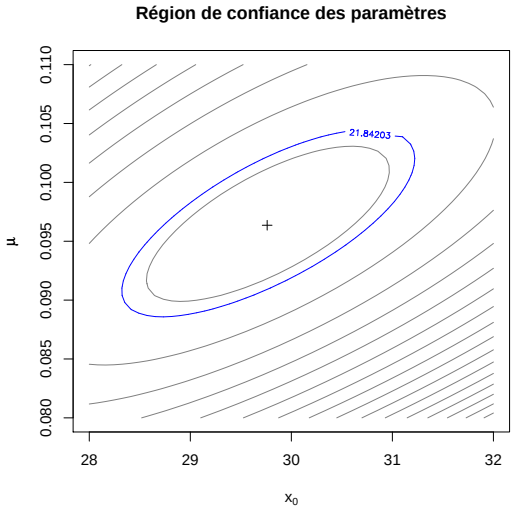
$$\theta / S(\theta) \leq S(\hat{\theta})(1 + \frac{p}{n-p} F^{\alpha}_{p;n-p})$$

où $p$ est le nombre de paramètres du modèle, $n$ le nombre de points disponibles dans le jeu de données, et $\hat{\theta}$ le vecteur des valeurs des paramètres tel que le critère soit minimal.

```
scemin <- nlmfit$minimum
n <- length(data)
p <- 2
alpha = 0.05
seuil <- scemin * (1 + (p * qf(p = 1 - alpha, df1 = p, df2 = n -
    p))/(n - p))
seuil
```

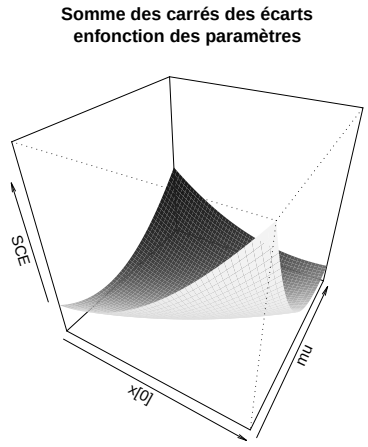
```
[1] 21.84203
```

```
x0seq <- seq(from = 28, to = 32, length = 50)
museq <- seq(from = 0.08, to = 0.11, length = 50)
scegrid <- matrix(nrow = length(x0seq), ncol = length(museq))
for (i in 1:length(x0seq)) {
    for (j in 1:length(museq)) {
        scegrid[i, j] <- sce(c(x0seq[i], museq[j]))
    }
}
contour(x = x0seq, y = museq, z = scegrid, drawlabels = FALSE, nlevels = 20,
    col = grey(0.5), xlab = expression(x[0]), ylab = expression(mu),
    main = "Région de confiance des paramètres")
points(nlmfit$estimate[1], nlmfit$estimate[2], pch = 3)
contour(x = x0seq, y = museq, z = scegrid, levels = seuil, add = TRUE,
    col = "blue")
```




Vue en perspective :

```
persp(x = x0seq, y = museq, z = scegrid, theta = 30, phi = 35, shade = TRUE,
    border = NA, xlab = expression(x[0]), ylab = expression(mu),
    zlab = "SCE", main = "Somme des carrés des écarts\nenfonction des paramètres")
```




## 3.2 Les fonctions de sensibilité

Les fonctions de sensibilité sont les dérivées partielles par rapport aux paramètres du modèle :

$$s_{x_0}(t) = \frac{\partial}{\partial x_0}\left(x_0 \mathrm{e}^{-\mu t}\right)$$

$$s_{\mu}(t) = \frac{\partial}{\partial \mu}\left(x_0 \mathrm{e}^{-\mu t}\right)$$

Elles sont particulièrement pénibles à calculer à la main parce que l'on ne dérive pas par rapport à la variable habituelle, il est alors extrêmement facile de

se tromper en appliquant un automatisme inadéquat. Dans ce genre de situation, le recours à un calculateur symbolique est particulièrement intéressant.

```
sx0 <- D(expression(x0 * exp(-mu * t)), "x0")
sx0
```

```
exp(-mu * t)
```

```
smu <- D(expression(x0 * exp(-mu * t)), "mu")
smu
```

```
-(x0 * (exp(-mu * t) * t))
```

On a donc :

$$s_{x_0}(t) = \mathrm{e}^{-\mu t}$$
$$s_{\mu}(t) = -x_0 t \mathrm{e}^{-\mu t}$$

Les fonctions de sensibilité expriment l'intensité de la variation du modèle à une modification de la valeur des paramètres. Par exemple, $s_{x_0}(t)$ est maximale pour $t = 0$ et nulle pour $t \to +\infty$, ce qui est assez logique pour la condition initiale $x_0$. Pour bien estimer $x_0$ on a donc intérêt à bien échantillonner au début de l'expérience. Les choses sont plus amusante pour $s_{\mu}(t)$. Comment se comporte cette fonction au cours du temps ?

```
smup <- D(smu, "t")
smup
```

```
-(x0 * (exp(-mu * t) - exp(-mu * t) * mu * t))
```

$$(s_{\mu}(t))' = -(x_0(\mathrm{e}^{-\mu t} - \mu t \mathrm{e}^{-\mu t})) = -x_0 \mathrm{e}^{-\mu t}(1 - \mu t)$$

On voit ici les limites du calculateur symbolique pour simplifier les expressions. La dérivée première s'annule pour $t = \frac{1}{\mu}$, vérifions que c'est un extremum avec la dérivée seconde :

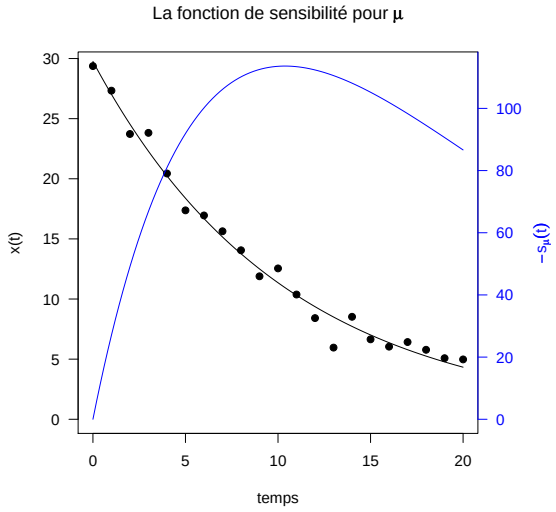
```
smus <- D(smup, "t")
smus
```

```
x0 * (exp(-mu * t) * mu + (exp(-mu * t) * mu - exp(-mu * t) *
    mu * mu * t))
```

$$(s_{\mu}(t))'' = x_0(\mathrm{e}^{-\mu t}\mu + (\mathrm{e}^{-\mu t}\mu - \mathrm{e}^{-\mu t}\mu\mu t)) = x_0 \mu \mathrm{e}^{-\mu t}(2 - \mu t)$$

On a donc un point d'inflexion pour $t = \frac{2}{\mu}$ et le signe de la dérivée seconde est inchangé au passage de $t = \frac{1}{\mu}$, on a donc bien un extremum. Représentons le modèle et la fonction de sensibilité pour $\mu$ :

```
par(mar = c(5, 4, 4, 4) + 0.1)
plot(0:20, data, pch = 19, xlab = "temps", ylab = "x(t)", las = 1,
    ylim = c(0, max(data)), main = expression(paste("La fonction de sensibilité pour ",
        mu)))
tseq <- seq(0, 20, length = 100)
x0 <- nlmfit$estimate[1]
mu <- nlmfit$estimate[2]
points(tseq, x0 * exp(-mu * tseq), type = "l")
fsmu <- function(t) eval(smu)
y <- -fsmu(tseq)
yscale <- max(data)/max(y)
lines(x = tseq, y = yscale * y, type = "l", col = "blue")
axis(side = 4, at = yscale * pretty(y), labels = pretty(y), las = 1,
    col = "blue", col.axis = "blue")
mtext(text = expression(-s[mu](t)), line = 3, side = 4, col = "blue")
```

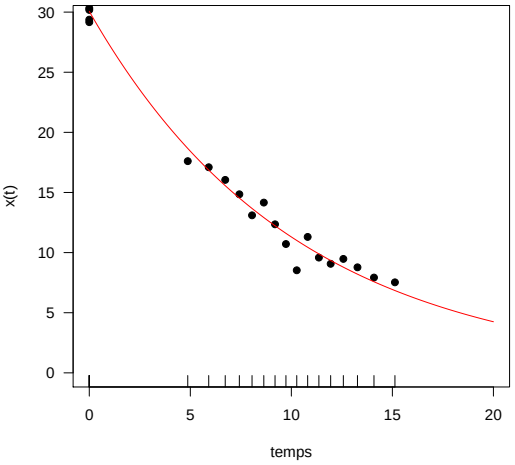



On a donc intérêt d'échantillonner au voisinage de $t = 10$ pour bien estimer le paramètre $\mu$.

## 3.3 Une deuxième expérience

On décide cette fois d'échantillonner la où c'est intéressant. On garde le même nombre total de points, et le même modèle d'erreur.

```
temps <- c(rep(0, 5), qnorm(mean = 10, sd = 4, p = seq(0.1, 0.9,
    length = 16)))
data2 <- 30 * exp(-0.1 * temps) + bruit
plot(temps, data2, pch = 19, xlab = "temps", ylab = "x(t)", las = 1,
    ylim = c(0, max(data)), xlim = c(0, 20))
rug(temps)
sce2 <- function(p) {
    x0 <- p[1]
    mu <- p[2]
    sum((data2 - x0 * exp(-mu * temps))^2)
}
nlmfit2 <- nlm(f = sce2, p = c(30, 0.1))
tseq <- seq(0, 20, length = 100)
points(tseq, nlmfit2$estimate[1] * exp(-nlmfit2$estimate[2] * tseq),
    type = "l", col = "red")
```

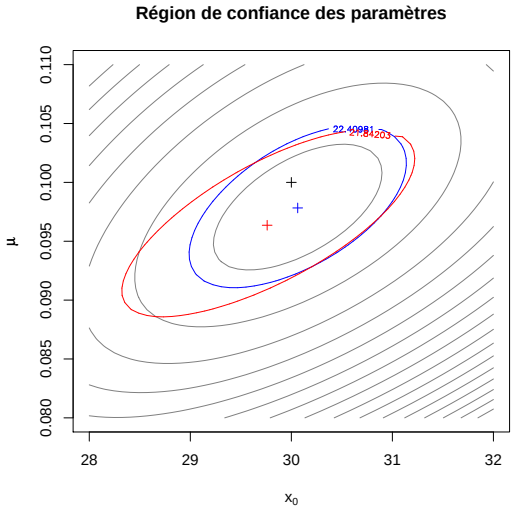



Représentons la régions de confiance :

```
scemin2 <- nlmfit2$minimum
n <- length(data2)
p <- 2
alpha = 0.05
seuil2 <- scemin2 * (1 + (p * qf(p = 1 - alpha, df1 = p, df2 = n -
    p))/(n - p))
seuil2
```

```
[1] 22.40981
```

```
scegrid2 <- matrix(nrow = length(x0seq), ncol = length(museq))
for (i in 1:length(x0seq)) {
    for (j in 1:length(museq)) {
        scegrid2[i, j] <- sce2(c(x0seq[i], museq[j]))
    }
}
contour(x = x0seq, y = museq, z = scegrid2, drawlabels = FALSE,
    nlevels = 20, col = grey(0.5), xlab = expression(x[0]), ylab = expression(mu),
    main = "Région de confiance des paramètres")
points(nlmfit2$estimate[1], nlmfit2$estimate[2], pch = 3, col = "blue")
contour(x = x0seq, y = museq, z = scegrid2, levels = seuil2, add = TRUE,
    col = "blue")
points(nlmfit$estimate[1], nlmfit$estimate[2], pch = 3, col = "red")
points(30, 0.1, pch = 3)
contour(x = x0seq, y = museq, z = scegrid, levels = seuil, add = TRUE,
    col = "red")
```




La nouvelle région de confiance en bleu est plus petite que la précédente en rouge, on a gagné en précision. Les nouvelles valeurs estimées pour les paramètres (croix bleue) sont plus proche des vraies valeurs (croix noire), on a gagné en exactitude. *Tout ceci pour un coût expérimental strictement identique.*

# Références

[1] E.M.L. Beale. Confidence regions in non-linear estimation. *Journal of the Royal Statistical Society*, 22B :41–88, 1960.