

Fiche TD avec le logiciel R : tdr81
 

---

## Comment rédiger un rapport avec la commande `odfWeave()` de R ?

 J.R. Lobry & A.B. Dufour
 

---

Cette fiche donne quelques indications pour rédiger un rapport intégrant des analyses statistiques et des graphiques produits par R. La connaissance de la fiche `tdr78.pdf` facilitera la compréhension de cette dernière.

### Table des matières

|  |    |
|--|----|
| 1. Introduction.....                                     | 2  |
| 2. Pré-requis.....                                       | 2  |
| 3. Utilisation d' <code>odfWeave</code> .....            | 2  |
| 3.1 Exemple 1 : une simple addition.....                 | 2  |
| 3.2 Exemple 2 : un graphique simple .....                | 2  |
| 3.3 Quelques astuces.....                                | 3  |
| 3.4 Formatage des entrées et sorties de code .....       | 3  |
| 4. Quelques clés d'utilisation.....                      | 3  |
| 4.1 Inclure un résultat dans le corps du texte.....      | 3  |
| 4.2 Générer une table à partir des données.....          | 4  |
| 4.3 Insérer une figure réalisée sous R.....              | 5  |
| 4.3.1 Sauvegarde des valeurs par défaut.....             | 5  |
| 4.3.2 Contrôler la taille d'une figure.....              | 6  |
| 4.3.3 Insertion de multiples figures.....                | 7  |
| 4.3.4 Remarque : une erreur classique.....               | 7  |
| 5. Les paramètres de contrôle des fragments de code..... | 8  |
| 5.1 Le paramètre <code>echo</code> (TRUE).....           | 8  |
| 5.2 Le paramètre <code>eval</code> (TRUE).....           | 8  |
| 5.3 Le paramètre <code>results</code> (verbatim).....    | 8  |
| 5.4 Le paramètre <code>print</code> (FALSE).....         | 9  |
| 5.5 Le paramètre <code>term</code> (TRUE).....           | 9  |
| 5.6 Le paramètre <code>fig</code> (FALSE).....           | 9  |
| 6. Annexe.....   | 10 |
| 7. Restaurations des paramètres de la console.....       | 11 |

## 1. Introduction

Comme déjà souligné dans la fiche tdr78 consacrée à la rédaction d'un rapport avec la commande Sweave(), il existe deux grandes approches pour rédiger un rapport faisant appel à des analyses et à des graphiques produits sous  $\mathbb{R}$ . La première approche est artisanale et consiste à utiliser le copier/coller entre  $\mathbb{R}$  et l'éditeur de texte. La deuxième approche consiste à utiliser la commande Sweave() permettant d'automatiser la production de documents en insérant le code source de  $\mathbb{R}$  dans le document. Sweave() s'inscrit dans la création de documents LaTeX. Ces derniers nécessitent un apprentissage que l'utilisateur ne désire pas forcément réaliser. En effet, ce dernier est souvent plus habitué aux logiciels de traitements de texte tels que Word ou Openoffice. Afin de rester dans la philosophie des logiciels libres, Max Kuhn propose un paquet `odfWeave` permettant de lier du code source de  $\mathbb{R}$  avec la rédaction de rapports qui peuvent ainsi être automatisés.

## 2. Pré-requis

Pour bien s'appropriier `odfWeave`, l'utilisateur doit posséder quelques connaissances en Sweave, ce que nous supposons ici. `odfWeave` nécessite des fichiers en format Open Document, version 1.0 ou plus. Ils peuvent être générés par exemple par OpenOffice version 2.0 (ou plus). Le paquet requiert également un utilitaire pour compresser ou décompresser des documents (zip et unzip). Par défaut, `odfWeave` sauve les images en format png.

## 3. Utilisation d'odfWeave

L'intérêt de lier code source de  $\mathbb{R}$  et rédaction peut être perçu dans cette première approche élémentaire.

1. Récupérez le fichier OpenOffice `tdr81in.odt` correspondant à ce document à l'URL <http://pbil.univ-lyon1.fr/R/donnees/> pour le sauvegarder dans votre dossier de travail.
2. Ouvrez ce document avec OpenOffice.
3. Lancez  $\mathbb{R}$ .
4. Allez dans votre dossier de travail.
5. Charger la bibliothèque `odfWeave` avec la commande :

```
library("odfWeave")
```

6. Compilez le document avec la commande :

```
odfWeave("tdr81in.odt", "tdr81out.odt").
```

Vous devez maintenant avoir dans votre dossier de travail un document `tdr81out.odt` que vous pouvez ouvrir avec OpenOffice. Ainsi, en comparant les deux documents, vous visualiserez à la fois le code et le résultat obtenu après compilation. Le document généré contient ce que vous aviez écrit, les commandes sont en rouge et les résultats en bleu, comme dans la fenêtre de commande  $\mathbb{R}$ .

### 3.1 Exemple 1 : une simple addition

```
<<fac1>>=  
2+2  
@
```


### 3.2 Exemple 2 : un graphique simple

```
<<fac2, fig=T>>=  
vec1 <- c(3,2,4,8)  
mean(vec1)
```

```
sd(vec1)
boxplot(vec1)
@
```



### 3.3 Quelques astuces

OpenOffice est un traitement de texte, pas un éditeur de texte orienté pour la programmation (par exemple il n'y a pas de colorisation syntaxique contextuelle comme dans Emacs). Nous avons noté ici quelques astuces destinées à vous faciliter la vie.

- Le code source est généralement plus lisible avec une police de caractères non proportionnelle, nous avons défini dans ce document un style appelé `Code` destiné aux fragments de code .
- Pour neutraliser les doubles guillemets typographiques par défaut aller dans Outils, AutoCorrection, Guillemets topographiques, et désactiver le remplacer.
- Pour éviter d'avoir les caractères "<" remplacés automatiquement par "←" aller dans Outils, AutoCorrection, et supprimer les activations gênantes.
- Pour désactiver le sur-lignage des mots inconnus dans le dictionnaire aller dans Outils, Orthographe et grammaire, Options et décocher les cases.
- Pour mettre à jour la table des matières après compilation faire un clic-droit sur la table des matières du document. FIXME : peut-on forcer la mise à jour automatique des index à l'ouverture du document ?

Ce document est loin d'être complet, les points non résolus sont notés avec une balise FIXME.

### 3.4 Formatage des entrées et sorties de code

Pour que les entrées et sorties de code  dans votre document final soient plus faciles à copier/coller directement dans une console , vous pouvez évaluer le fragment de code suivant :

```
<<optionsCode>>=
oldOptions <- options()
options(prompt = " ", continue = " ", width = 85)
@
```

Le principal effet est de supprimer le caractère d'invite de commande > dans le document final :

```
<<exnoinvit>>=
2 + 2
@
```

Le dernier fragment de code tout à la fin de ce document permet de restaurer les options sauvegardées ici. Vous pouvez utiliser l'option `echo=FALSE` pour masquer ces fragments de code dans le document final.

## 4. Quelques clefs d'utilisation

### 4.1 Inclure un résultat dans le corps du texte

Le volume  $V$  d'une sphère de rayon  $r$  est donné par :  $V(r) = 4/3 \pi r^3$

```
<<fac3, fig=F>>=
volume <- fonction(r) (4/3)*pi*r^3
volume(1)
@
```

Le volume d'une sphère de rayon unité est donc  $\backslash\text{Sexpr}\{\text{volume}(1)\}$ . Si on ne veut afficher que quatre chiffres après la virgule, il faut utiliser la fonction `round()`. On obtient alors que le volume d'une sphère unité est  $\backslash\text{Sexpr}\{\text{round}(\text{volume}(1), 4)\}$ . Notez au passage que les styles de caractères sont conservés dans le document final (ici des caractères en bleu).

## 4.2 Générer une table à partir des données

Affichons les 6 premières lignes du data frame des iris d'Anderson.

```
<<fac4a, echo = TRUE, results=xml>>=
data(iris)
echa6 <- head(iris)
odfTable(echa6)
@
```

Nous remarquons que la table construite par défaut est totalement inesthétique. Il faut donc définir le style souhaité. Pour ce faire, nous avons créé au préalable **dans OpenOffice** :

- un "styleTableCell" courier new 10pt centré pour le contenu des cellules d'une table,
- un "styleTableHead" courier new gras 10pt centré pour les entêtes des colonnes d'une table.

```
<<fac4b, echo = TRUE, results=xml>>=
data(iris)
echa6 <- head(iris)
#
# modification des styles de la table
#
iristabStyles <- tableStyles(echa6, useRowNames=F, header=names(iris))
iristabStyles$text[,] <- "StyleTableCell"
iristabStyles$header <- "StyleTableHead"
odfTable(echa6, useRowNames=F, styles=iristabStyles)
@
```

Nous pouvons également mettre en évidence une cellule d'une table comme par exemple le minimum de largeur du sépale en créant un style "texteRouge".

```
<<fac4c, echo = TRUE, results=xml>>=
iristabStyles <- tableStyles(echa6, useRowNames=F, header=names(iris))
iristabStyles$text[,] <- "StyleTableCell"
iristabStyles$text[3,2] <- "texteRouge"
iristabStyles$header <- "StyleTableHead"
odfTable(echa6, useRowNames=F, styles=iristabStyles)
@
```

OdfWeave permet un contrôle très fin des tables générées comme le montre l'exemple un peu plus sophistiqué ci-après.

```
Style pour les valeurs nulles (monstyle1)
Style pour les valeurs non nulles (monstyle2)
Style pour les noms des lignes et colonnes (monstyle3)
```

```
<<essaicrimtab, results=xml, echo = FALSE>>=
#
# Création du jeu de données :
#
data(crimtab)
colnames(crimtab) <- as.numeric(colnames(crimtab))/2.54
rownames(crimtab) <- as.numeric(rownames(crimtab))*10
crimtab <- crimtab[nrow(crimtab):1, ]
#
# Modification des styles de la table :
#
crimtabStyles <- tableStyles(crimtab, useRowNames = TRUE,
```

```

header = names(crimtab))

leszeros <- which(crimtab == 0, arr = TRUE)
leszeros[,2] <- leszeros[,2] + 1
crimtabStyles$text[leszeros] <- "monstyle1"

lesnonzeros <- which(crimtab != 0, arr = TRUE)
lesnonzeros[,2] <- lesnonzeros[,2] + 1
crimtabStyles$text[lesnonzeros] <- "monstyle2"

crimtabStyles$header <- "monstyle3"
crimtabStyles$text[, 1] <- "monstyle3"

#
# Mise à zéro des marges des cellules :
#

currentDefs <- getStyleDefs()
currentDefs$myNoBorder <- currentDefs$noBorder
currentDefs$myNoBorder[[3]] <- "0.0in"
currentDefs$myNoBorder[[5]] <- "0.0in"
setStyleDefs(currentDefs)
crimtabStyles$cell[ , ] <- "myNoBorder"

#
# On force à commencer sur une nouvelle page :
#

odfPageBreak()

#
# Création de la table :
#

odfTable(crimtab, useRowNames = TRUE, styles = crimtabStyles)

#
# Création de la légende :
#

odfTableCaption("Les données utilisées par William Sealy Gosset, plus connu sous le nom de plume de Student et du test statistique éponyme, en 1908 (Biometrika 6:1-25). La collecte des données a été faite par MacDonell en 1902 (Biometrika 1:177-227). Il s'agit d'une table de contingence obtenue en discrétisant la taille (exprimée ici en pouces, soit 2.54 cm, et portée en colonne) et la longueur du majeur de la main gauche (exprimée ici en mm et portée en ligne) de 3000 criminels adultes de sexe mâle écroués au pays de Galle et en Angleterre à la fin du XXIe siècle. Voir ?crimtab pour plus d'informations.")
@

```

FIXME : comment faire référence facilement dans le corps du texte à une table générée automatiquement ?

FIXME : comment ne pas perdre certains attributs typographiques dans la légende d'une table?

## 4.3 Insérer une figure réalisée sous R

### 4.3.1 Sauvegarde des valeurs par défaut

Le style des images est défini en utilisant les fonction `setImageDefs` et `getImageDefs`. Dans la recherche d'un graphique satisfaisant, il faut commencer par sauvegarder le mode graphique de base pour pouvoir le restaurer si besoin est.

```

<<plot1, fig=FALSE>>=
oldPlotDefs <- getImageDefs()
oldPlotDefs
@

```

Nous voyons ici que par défaut les graphiques générés seront au format `\Sexpr{oldPlotDefs$type}`, avec un périphérique graphique qui s'appelle `\Sexpr{oldPlotDefs$type}`. Un graphique est caractérisé par sa hauteur (`height`) et sa largeur (`width`). Il faut distinguer la taille du graphique au moment de sa création avec le périphérique graphique de sa taille dans le document final. Ici au moment de sa création le

graphique fait  $\backslash\text{Sexpr}\{\text{oldPlotDefs}\$plotHeight\}$  unités de haut par  $\backslash\text{Sexpr}\{\text{oldPlotDefs}\$plotWidth\}$  unités de large. Les unités utilisés sont variables d'un périphérique graphique à l'autre, ici ?png nous dit qu'il s'agit de pixels par défaut (1 pouce = 72 pixels). La taille du graphique dans le document final est exprimée en pouces (1 inch = 2.54 cm) soit ici  $\backslash\text{Sexpr}\{\text{oldPlotDefs}\$dispHeight\}$  pouces de haut par  $\backslash\text{Sexpr}\{\text{oldPlotDefs}\$dispWidth\}$  pouces de large.

FIXME : peut-on utiliser d'autres unités ?

FIXME : peut-on insérer des figures dans un format vectoriel ?

### 4.3.2 Contrôler la taille d'une figure

Voici ce qui est obtenu avec les réglages par défaut :

```
<<uniplotbase, fig=TRUE>>=
resnorm <- rnorm(1000)
hist(resnorm, col="lightblue")
@
```

On remarque que le graphe obtenu par défaut occupe beaucoup de place dans le document final. L'objectif est de combiner les paramètres afin d'obtenir un graphe conforme à son attente. On peut réduire la taille de son graphique en choisissant de le représenter dans des dimensions plus petites.

```
<<plot2, fig=TRUE>>=
currentPlotDef <- getImageDefs()
currentPlotDef$dispHeight <- 2.5
currentPlotDef$dispWidth <- 2.5
setImageDefs(currentPlotDef)
hist(resnorm, col="lightblue")
@
```

Si maintenant on augmente la taille du graphique au moment de sa production, ceci va augmenter sa résolution dans le document final, par exemple :

```
<<plot2b, fig=TRUE>>=
currentPlotDef$plotHeight <- 2*384
currentPlotDef$plotWidth <- 2*384
setImageDefs(currentPlotDef)
hist(resnorm, col="lightblue")
@
```

Ne pas oublier une fois le graphe réalisé de redéfinir la version originale du style d'image, sans quoi toute la suite du document sera contaminée.

```
<<reset, fig=FALSE>>=
setImageDefs(oldPlotDefs)
@
```

Voici maintenant un exemple un peu plus sophistiqué dans lequel on cherche à placer deux figures juxtaposées dans le document final :

```
<<plottwofigs, fig=TRUE>>=
currentPlotDef <- getImageDefs()
currentPlotDef$dispHeight <- 2.5
currentPlotDef$dispWidth <- 5
currentPlotDef$plotHeight <- 2*2.5*72
currentPlotDef$plotWidth <- 2*5*72
setImageDefs(currentPlotDef)
par(mfrow = c(1, 2))
hist(resnorm, col="lightblue")
qqnorm(resnorm)
qqline(resnorm)
@
```

```
<<echo=F>>=
<<reset>>
@
```

On peut obtenir le même résultat de façon plus facile en utilisant la fonction utilitaire `adjustImageSize`. `scale` est un coefficient multiplicateur des dimensions en ordonnée et en abscisse.

```
<<plot4a, fig=TRUE>>=
adjustImageSize(5, 2.5, scale = 1)
par(mfrow = c(1, 2))
hist(resrnorm, col="lightblue")
qqnorm(resrnorm)
qqline(resrnorm)
@
```

```
<<plot4b, fig=TRUE>>=
adjustImageSize(5, 2.5, scale = 2)
par(mfrow = c(1, 2))
hist(resrnorm, col="lightblue")
qqnorm(resrnorm)
qqline(resrnorm)
@
```

Enfin, comme pour la réalisation des tables, on peut rajouter une légende au graphique.

```
<<plot5, echo=FALSE, results= xml, fig=TRUE>>=
adjustImageSize(3, 3, scale = 1.75)
hist(resrnorm, col="lightblue")
odfFigureCaption("Histogramme de 1000 valeurs de tirage dans une loi normale centrée -
réduite")
@
```

On peut noter également que l'ajustement d'un graphique peut se réaliser dans openOffice de manière classique en double-cliquant sur l'image et en modifiant les paramètres.

FIXME : peut-on contrôler ceci au moment de la génération des figures ?

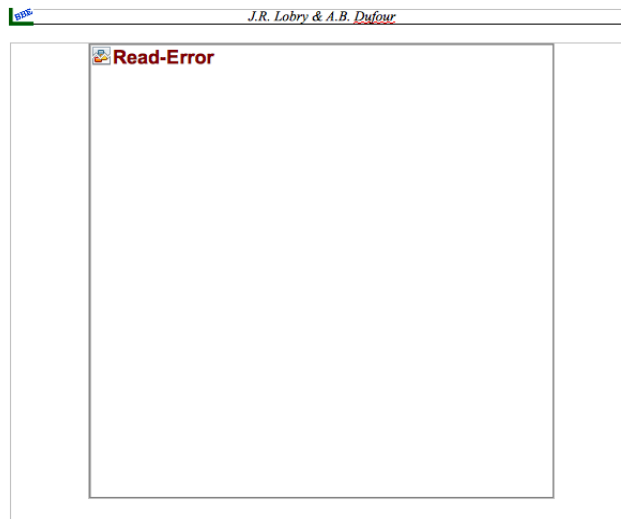
### 4.3.3 Insertion de multiples figures

L'option `fig=TRUE` ne permet d'insérer qu'une seule figure par fragment de code. On peut contourner cette limitation en générant directement du XML avec un code du type :

```
<<multiplot, eval = TRUE, fig=FALSE, echo = FALSE, results=xml>>=
nfig <- 3
for(i in 1:nfig){
  fname <- paste("multiplot", i, ".png", sep = "")
  png(fname)
  hist(rnorm(1000), col = "lightblue", main = fname)
  dev.off()
  cat(odfInsertPlot(fname, 2, 2, externalFile = TRUE))
}
@
```

### 4.3.4 Remarque : une erreur classique

Si vous utilisez l'option `fig=TRUE` mais que votre fragment de code ne produit aucun graphique vous aurez une erreur de lecture de la figure dans votre document final :



## 5. Les paramètres de contrôle des fragments de code

Ces paramètres sont documentés dans `?RweaveOdf`.

FIXME : peut-on les contrôler globalement ?

### 5.1 Le paramètre `echo` (*TRUE*)

Ce paramètre contrôle si le code R doit être repris dans le fichier de sortie. Comparez ces deux exemples avec et sans `'echo = TRUE'` :

```
<<echoT, echo = TRUE>>=  
2+2  
@
```

```
<<echoF, echo = FALSE>>=  
2+2  
@
```

### 5.2 Le paramètre `eval` (*TRUE*)

Ce paramètre contrôle si le code R doit être évalué ou non :

```
<<evalT, eval=TRUE>>=  
2+2  
@
```

```
<<evalF, eval=FALSE>>=  
2+2  
@
```

### 5.3 Le paramètre `results` (*verbatim*)

Avec la valeur par défaut, les résultats des commandes R sont intégrées automatiquement dans le document compilé avec le style de type `verbatim` pré-défini `ttBlue`.

```
<<resultsV, results=verbatim>>=  
2+2  
@
```

Avec la valeur `hide`, les résultats sont complètement omis dans le document final, mais le code R est bien exécuté.

```
<<resultsH, results=hide>>=  
res <- 2+2  
print(res)  
@
```

On n'a aucune sortie ici, mais la variable `res` vaut bien `\Sexpr{res}`.

Enfin, avec `results=xml` il est possible de générer directement du code XML qui sera directement intégré dans le document final. C'est l'équivalent de `results=tex` avec `Sweave()`. C'est typiquement ce que l'on fait quand on utilise la fonction `odfTable()`. Voici un exemple dans lequel on génère un texte différent dans le document final en fonction du résultat d'un test statistique:

```
<<resultsXML, echo=T, results=xml>>=  
alpha <- 0.05  
x <- rnorm(10)  
y <- rnorm(10)  
restt <- t.test(x, y, conf.level = 1 - alpha)  
message <- paste("Nous obtenons une p.valeur de", round(restt$p.value, 4), ". ")  
if(restt$p.value <= alpha){  
  message <- paste(message, "Avec un risque de première espèce de", round(100*alpha), "%  
les données expérimentales sont donc en contradiction avec l'hypothèse nulle.")  
} else {  
  message <- paste(message, "Avec un risque de première espèce de", round(100*alpha), "%  
les données expérimentales ne permettent donc pas de rejeter l'hypothèse nulle.")  
}  
odfCat(message)  
@
```

FIXME : comment contrôler le style du paragraphe généré par `odfCat` ?

## 5.4 Le paramètre `print` (**FALSE**)

Ce paramètre force toutes les expressions à être encapsulées dans une fonction `print()`, est peut-être utile en phase de mise au point.

```
<<printT, print=TRUE>>=  
a <- 2+2  
a  
@
```

## 5.5 Le paramètre `term` (**TRUE**)

Ce paramètre permet de simuler le comportement d'une console R:

```
<<termT, term=TRUE>>=  
a <- 2+2  
a  
@
```

```
<<termT, term=FALSE>>=  
a <- 2+2  
a  
@
```

## 5.6 Le paramètre `fig` (**FALSE**)

Ce paramètre indique si le fragment de code R produit une sortie graphique ou non:

```
<<figF, fig=F>>=
hist(rnorm(100))
@

<<figT, fig=T>>=
adjustImageSize(4, 4)
hist(rnorm(100), col = "lightblue")
@
```

## 6. Annexe

Ce document est rédigé d'après `formatting.odt` publié par Max Kuhn le 29 septembre 2009. En voici une traduction partielle.

A l'intérieur d'un document, on peut changer les styles des éléments tels que les tables, les figures. Une série de graphiques peut avoir des dimensions variées. Les cellules d'une table peuvent être mises en valeur de différentes façons. Les deux fonctions principales sont `getStyles` et `setStyles`.

```
<<showDefs>>=
currentDefs <- getStyleDefs()
allStyles <- data.frame(type = unlist(lapply(currentDefs, function(x) x$type)))
allStyles <- allStyles[order(allStyles$type), ,drop = FALSE]
allStyles
@
```

### Les styles de paragraphes

Le style d'un paragraphe peut affecter plusieurs parties du document : du texte dans un paragraphe simple à celui qui apparaît dans les cellules d'un tableau. Si nous regardons la définition de la police de caractères associée à un paragraphe, nous trouvons `ArialNormal`. Le code chunk associé est :

```
<<basicTextDef>>=
styleDetails <- function(x)
{
  out <- as.data.frame(unlist(getStyleDefs()[x]))
  rownames(out) <- unlist(
    lapply(strsplit(rownames(out), "\\."), function(x) x[2]))
  colnames(out) <- "Element Value"
  out
}
styleDetails("ArialNormal")
@
```

Les arguments liés au style du paragraphe sont :

- `type` : for this style to correspond to paragraph formatting, this should be the text string "paragraph".
- `ParentStyleName` définit une chaîne de caractères optionnelle pour un style « parent » - hiérarchie supérieur au paragraphe – qu'il est possible de modifier.
- `TextAlign` correspond à la façon de justifier le texte dans un paragraphe. Cinq choix sont possibles : "start", "end", "left", "right", or "center".
- `fontName`: une chaîne de texte qui spécifie la police de caractère utilisée telle que « Arial », « Courier », etc.

- `fontSize` est une chaîne de texte définissant la taille de la police de caractère. Elle doit être définie en terme de taille de point comme par exemple “12pt”.
- `fontType` correspond à une chaîne de caractères spécifiant le style de la police utilisée : gras, italique, etc soit en anglais “bold,” “italic”, “underline”, “shadow”, “superscript” et “subscript”. Dans une même chaîne de caractères, il est possible bien sûr de choisir des critères de style multiple comme par exemple à la fois gras et italique ( “bold italic” ) ; l'ordre d'apparition est sans importance. OdfWeave a codé en dur la définition de quelques propriétés comme par exemple le texte superscript représente 58% du texte normal et ne peut être changé.
- `fontColor` est une autre chaîne de caractères permettant de préciser la couleur en format hex. La fonction `colorConverter` du package `grDevices` peut être utilisée pour préciser cette valeur. Le code chunk ci-dessous fournit une fonction permettant de se déplacer entre les noms de couleur et les représentations hex (comme le fait la fonction `color()` ou d'autres fonctions).

```
col2hex <- function(col)
{
  require(colorspace)
  rgbVals <- col2rgb(col)/255
  rgbObj <- RGB(rgbVals[1,], rgbVals[2,], rgbVals[3,])
  hex(rgbObj)
}
```

## 7. Restaurations des paramètres de la console

Le fragment de code suivant exécuté tout à la fin du document permet de restaurer les paramètres de la console R auquel l'utilisateur est habitué (par exemple le caractère > comme invite de commande).

```
<<restaurationDesOptions>>=
options(oldOptions)
@
<<verifRestaurationDesOptions>>=
2 + 2
@
```