1 Automatiser la production de documents

Fabrice Dessaint
Inra - UMR Agroécologie, Dijon
10 mai 2014

L'intégration de résultats numériques et/ou graphiques issus de logiciels statistiques, dans les mémoires ou rapports est souvent compliquée. Outre les problèmes de mise en forme ou de format, toute modification dans l'analyse se traduit par une nouvelle opération d'importation. Les risques d'erreurs liés à ce mode de fonctionnement peuvent être importants dès lors que l'on a de nombreux résultats. La fonction `Sweave()` de R, couplée au logiciel LaTeX, permet de résoudre une partie de ces problèmes en automatisant cette tâche.
Cette note présente l'utilisation de la fonction `Sweave()`.

Sommaire

1.1 Introduction

La production de mémoires ou de rapports nécessite généralement l'intégration de résultats numériques et/ou graphiques dans le document. Or ces résultats sont souvent produits par un (des) logiciel(s) différent(s) de celui utilisé pour la rédaction du document. La manière la plus largement répandue de procéder passe par la copie du résultat fourni par le logiciel dans le presse papier (« copier ») puis du presse papier vers le document (« coller »). Lorsque ces opérations sont nombreuses ou lorsque l'on est amené à les répéter fréquemment, il peut être

intéressant de les automatiser en partie.

Pour ceux qui utilisent LaTeX et *R*, il existe une solution : la fonction `Sweave()`. Cette solution est basée sur une méthode appelée la programmation littéraire (*literate programming*). Dans cette approche, le code et le texte le documentant sont contenus dans le même document. L'intérêt de cette approche réside dans le fait que l'ensemble des éléments nécessaires à la construction du mémoire/rapport est contenu dans un seul fichier. On peut donc le reproduire à l'identique dès que l'on en a besoin. Associé aux fichiers de données [1], il peut faire l'objet d'une démarche de type *reproducible research*.

1.2 Premiers pas

La création d'un document à partir d'un fichier `Sweave` [2] nécessite un minimum de connaissance du fonctionnement de LaTeX et de *R*. En effet, contrairement aux traitements de texte et/ou aux logiciels de statistique classiques, ces deux programmes ne sont pas WYSIWYG. Dans les deux cas, il faut saisir des commandes de mise en forme ou des instructions de traitement, directement au clavier. C'est justement ce mode de fonctionnement qui permet l'utilisation de la fonction `Sweave()` de *R*.

On ne détaillera pas le fonctionnement de LaTeX ni celui de *R*. Le lecteur néophyte trouvera énormément de documents de présentation/utilisation de ces deux logiciels sur internet.

1.2.1 Procédé de fabrication

La création d'un document à partir d'un fichier `Sweave` comprend 4 étapes principales (Figure 1.1) :

Édition — Lors de cette première étape, on va créer le fichier *source*. Ce fichier a généralement l'extension `.Rnw` [3]. On peut aussi utiliser les extensions `.rnw`, `.Snw` ou `.snw`. Dans ce fichier, on saisit à la fois le texte d'explication, les commandes de mise en forme de LaTeX et les instructions *R*. Cette première étape ne nécessite qu'un logiciel de saisie de texte et tout éditeur de texte, même rudimentaire, peut convenir. Néanmoins, l'utilisation d'éditeurs de texte orientés LaTeX et/ou `Sweave` est recommandée. En général, ils facilitent la saisie en proposant, par exemple, une coloration syntaxique qui permet de différentier les différentes parties.

Traitement — Le fichier source (`.Rnw`) est ensuite traité sous *R* avec la fonction `Sweave()`. Si aucune erreur n'a été rencontrée par *R*, le résultat de ce traitement est un nouveau fichier ayant le même nom que le fichier source mais une extension `.tex`. Dans ce fichier le code *R* a été interprété et les résultats intégrés.

Compilation — Le nouveau fichier est un document au format `.tex`. Il doit être compilé [4] pour donner le document final. Selon le compilateur utilisé (LaTeX, pdfLaTeX, etc), ce document pourra être un fichier `.dvi`, `.eps`, `.pdf`, etc.

Visualisation — Lorsque la compilation est terminée, le résultat mis en forme peut être visualisé sur écran, vidéo-projeté, imprimé, etc.

1.2.2 Mise en œuvre

On va maintenant décrire les différentes étapes sur un petit fichier.

1. Les petits jeux de données pourraient même être inclus dans le fichier
2. Dans la suite du document, on appellera fichier `Sweave`, le fichier source qui sera ensuite traité par la fonction `Sweave()` de *R*
3. La syntaxe utilisée pour ces fichiers est basée sur la syntaxe `noweb`, dont les fichiers ont comme extension `.nw`
4. Éventuellement plusieurs fois, par exemple, si le fichier `.tex` contient des références, un sommaire, etc.

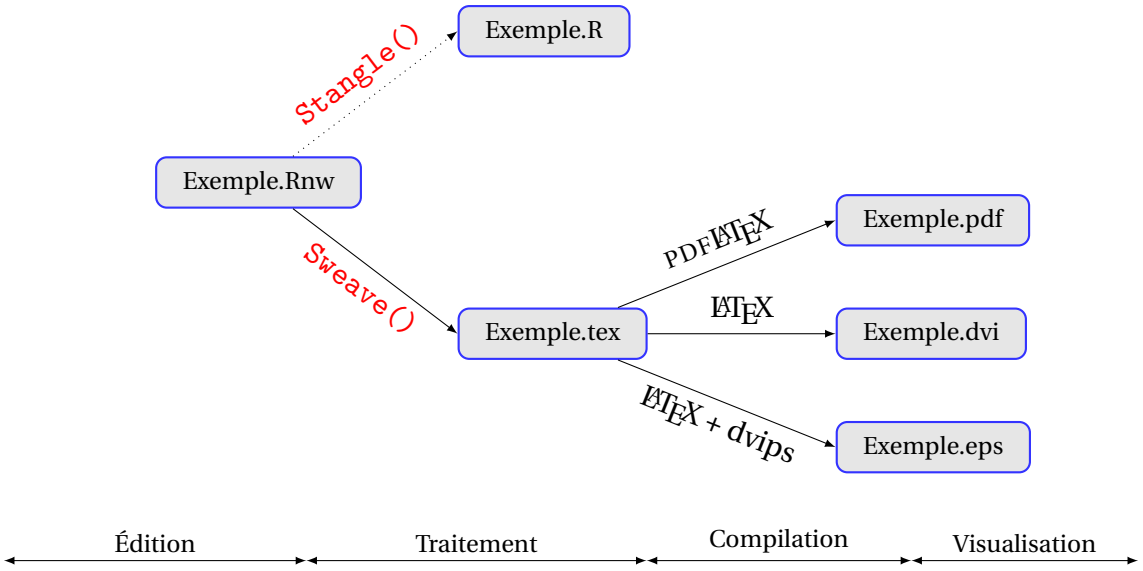


FIGURE 1.1 – Les différentes étapes du traitement d'un fichier Sweave

Édition

On a saisi dans le fichier *Exemple.Rnw* un certain nombre de lignes de texte. On a un mélange de texte d'explication, de commandes de mise en forme LaTeX, de code *R* et d'indications/commandes pour la fonction `Sweave()`.

Exemple.Rnw

```
1  \documentclass{article}
2  \usepackage{Sweave}
3  \begin{document}
4  Sous R, on peut faire des additions et des multiplications, comme celles-ci:
5  <<>>= ---------------------------------------- fin du texte et debut du code R
6  somme <- 3 + 2; somme
7  produit <- 3 * 2
8  @ ---------------------------------------------- fin du code R et suite du texte
9  et calculer la valeur du logarithme, en base 2, de 9: $\log_2(9) = \Sexpr{log(9,base=2)}$.
10
11 On peut aussi produire des graphiques comme celui-ci:
12 <<label=histogramme,fig=true>>= ---------------- fin du texte et debut du code R
13 set.seed(123)
14 x <- rnorm(1E3)
15 hist(x)
16 @ ---------------------------------------------- fin du code R et suite du texte
17 \end{document}
```

Dans ce fichier, on trouve :

- en lignes 1, 2, 3 et 17, des commandes LaTeX. Ces commandes permettent la mise en forme du document.
- en lignes 4, 9 et 11 du texte « *utilisateur* ». C'est un texte d'explication. En ligne 9, on note aussi la présence d'une commande `Sweave`, `\Sexpr{...}`, incluse dans le texte.
- en lignes 5, 8, 12 et 16 des balises reconnues par la fonction `Sweave()`. Ces balises signalent la présence de deux types de bloc de texte (*chunks*) : des blocs de code et des blocs de documentation [5].
- en lignes 6–7 et 13–15, des instructions *R*.

La partie `Sweave` se résume aux lignes 5, 8, 12 et 16 et à la commande `\Sexpr{...}`.

5. À l'origine, cette approche était utilisée pour documenter du code informatique

Les lignes 5 et 12 **commencent** par la séquence `<<...>>=`. Cette séquence indique que les lignes suivantes, constituent un bloc de code (*code chunk*) qui sera traité par *R*. Le reste de la ligne est considéré comme étant du commentaire qui doit être ignoré.

La séquence admet des **options**, sous la forme `option=valeur`. Toutes les options ont une valeur par défaut et seules les options dont on souhaite modifier la valeur sont présentes. Dans l'exemple, la séquence présente en ligne 5 n'a pas d'option alors que celle présente en ligne 12 en a 2 : la première, `label=histogramme`, affecte une étiquette particulière (`histogramme`) au bloc de code ; la seconde, `fig=true`, indique que l'on souhaite créer un fichier contenant le graphique produit par *R* (ici un histogramme).

Les lignes 8 et 16 **commencent** par la balise `@`. Cette balise indique que les lignes qui suivent constituent un bloc de documentation (*documentation chunk*) qui doit être recopié à l'identique, sans être interprété par *R*, dans le fichier `.tex`. Cette balise **doit** être suivie d'un blanc ou d'un caractère de fin de ligne. Si le reste de la ligne n'est pas vide, il est ignoré.

Sauf indication contraire, le fichier est supposé commencer par un bloc de documentation et dans ce cas, la balise `@` peut être omise.

La commande `\Sexpr{...}` permet d'exécuter une ou plusieurs instructions *R* dont le résultat est un **scalaire**. Cet élément peut alors être inclus dans un paragraphe, dans un tableau, une formule, etc.

Traitement

Sous *R*, on va maintenant exécuter la fonction `Sweave()`. Cette fonction est disponible dans la version de base de *R* : il n'y a donc rien à installer [6]. La fonction possède plusieurs arguments [7] mais le seul qui soit obligatoire, c'est le premier, `file`. Il indique le nom du fichier source à traiter, ici le fichier *Exemple.Rnw* .

```
> Sweave(file="Exemple.Rnw", encoding="UTF-8")
```

La fonction extrait les blocs de code qu'elle soumet ensuite à *R*. Chaque bloc de code est exécuté dans l'ordre d'apparition dans le fichier. La sortie *R* résume les opérations effectuées :

```
Writing to file Exemple.tex
Processing code chunks with options ...
 1 : echo keep.source term verbatim (Exemple.Rnw:5)
 2 : echo keep.source term verbatim pdf  (label = histogramme, Exemple.Rnw:12)

You can now run (pdf)latex on 'Exemple.tex'
```

Deux blocs de code ont été traités avec les options précisées derrière le numéro du bloc. Les sorties produites sont copiées dans le nouveau fichier, *Exemple.tex* , ainsi que les blocs de documentation.

Le contenu du fichier *Exemple.tex* [8] est le suivant :

Exemple.tex

```
1  \documentclass{article}
2  \usepackage{Sweave}
3  \begin{document}
4  Sous R, on peut faire des additions et des multiplications, comme celles-ci:
5  \begin{Schunk}
6  \begin{Sinput}
7  > somme <- 3 + 2; somme
8  \end{Sinput}
9  \begin{Soutput}
10 [1] 5
11 \end{Soutput}
12 \begin{Sinput}
```

6. Elle est présente dans le package *utils*
7. Faire `help(Sweave)` pour avoir la liste
8. Ce fichier est reconstruit à chaque exécution de la fonction `Sweave()`. On peut le considérer comme un fichier annexe au même titre que les fichiers `.log`, `.aux`, etc, créés par les compilateurs LaTeX

```
13 > produit <- 3 * 2
14 \end{Sinput}
15 \end{Schunk}
16 et calculer la valeur du logarithme, en base 2, de 9: $\log_2(9) = 3.16992500144231$.
17
18 On peut aussi produire des graphiques comme celui-ci:
19 \begin{Schunk}
20 \begin{Sinput}
21 > set.seed(123)
22 > x <- rnorm(1E3)
23 > hist(x)
24 \end{Sinput}
25 \end{Schunk}
26 \includegraphics{Exemple-histogramme}
27 \end{document}
```

Dans ce fichier, les indications/commandes Sweave ont disparu. On retrouve le contenu des blocs de documentation ; ils ont simplement été copiés. Par contre, chaque bloc de code a été remplacé par un ensemble de lignes contenues dans un environnement LaTeX particulier : l'environnement `Schunk`. Le premier bloc de code a produit les lignes 5 à 15 et le second, les lignes 19 à 26.

À l'intérieur de l'environnement `Schunk`, on trouve 2 autres environnements : le premier, `Sinput`, va contenir les lignes de code qui ont été traitées par *R* et le second, `Soutput`, les sorties produites par le traitement.

La fonction `Sweave()` essaie de « mimer » le déroulement d'une session *R*. C'est à dire que chaque instruction est précédée du caractère d'invite (*prompt*) de *R*. Si elle produit un résultat affiché, il est copié dans le fichier puis on passe à l'instruction suivante. Chaque sortie affichée apparait après l'instruction qui l'a produite.

Pour le second bloc de code la fonction `Sweave()` ajoute en ligne 26 une commande LaTeX spécifiant l'inclusion d'un fichier graphique, *Exemple-histogramme* . Le nom de ce fichier est construit en combinant le nom du fichier source (sans extension) et l'étiquette du bloc de code.

La commande `\Sexpr{...}` a aussi été exécutée. La valeur $\log_2(9)$ a été calculée par *R*

```
16 et calculer la valeur du logarithme, en base 2, de 9: $\log_2(9) = 3.16992500144231$.
```

et le résultat intégré dans le fichier.

Compilation

Cette étape est classique dans le traitement des fichiers `.tex`. Il existe plusieurs compilateurs LaTeX permettant de transformer un fichier `.tex` en document mis en forme et pouvant être visualiser ou imprimer. Ici on utilise PdfLaTeX. Ce compilateur produit des fichiers au format `.pdf`. Selon le compilateur utilisé, certaines options de la commande Sweave ne pourront pas être utilisées ; ce sont des options qui gèrent le format des graphiques produits par *R*.

Sur notre exemple, après compilation, on obtient le fichier *Exemple.pdf* que l'on peut ensuite visualiser, imprimer, etc.

Visualisation

La Figure 1.2 présente le document mis en forme. Dans ce document, on retrouve les lignes de code, en italique, des sorties « texte », avec une police non proportionnelle et une sortie graphique (ici une figure). La mise en forme du code et des sorties est due au fichier de style *Sweave.sty* . Ce fichier contient, entre autre chose, la définition des environnements `Sinput` et `Souput`. On verra plus loin que l'on peut modifier ces environnements [9].

9. C'est ce qui a été fait pour le présent document

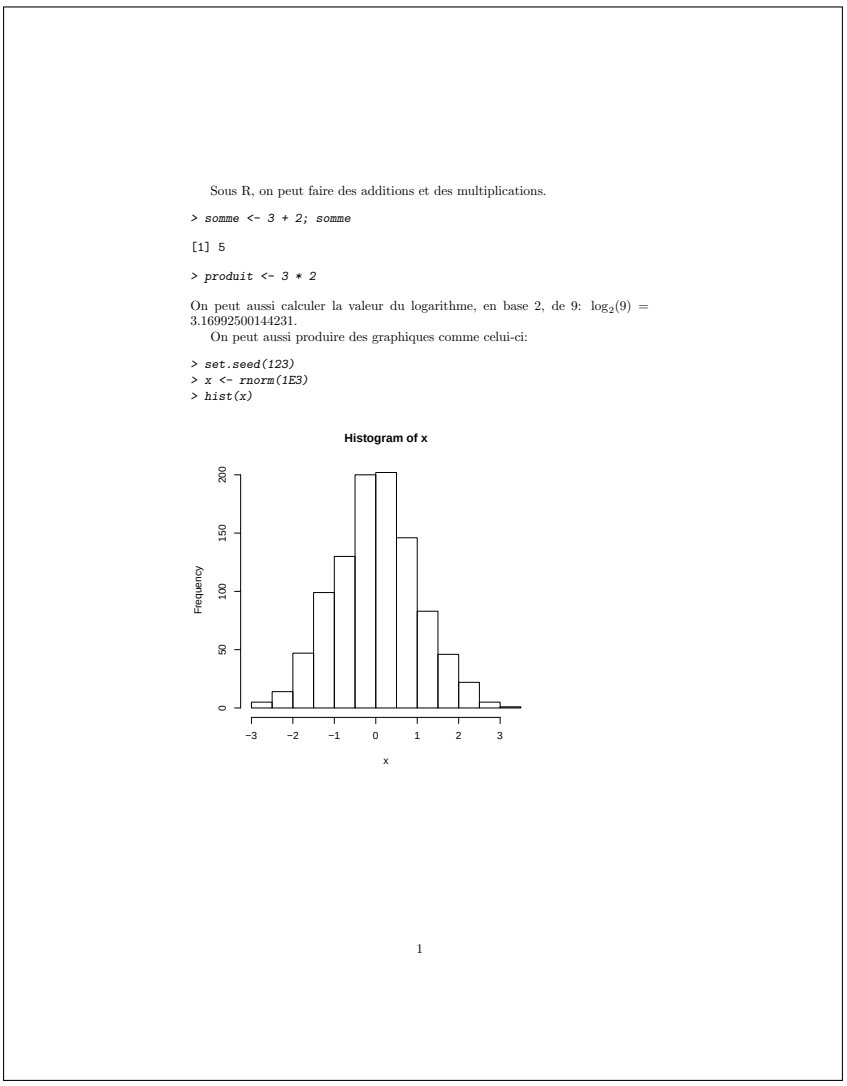


FIGURE 1.2 – Document produit à partir de PDFLaTeX et de la fonction `Sweave()` de *R*

1.3 Le fichier `.Rnw` : la syntaxe *noweb*

La syntaxe [10] utilisée pour les fichiers `.Rnw` reconnait deux types de blocs de texte (*chunks*) : des blocs de documentation (*documentation chunk*) et des blocs de code (*code chunk*).

Les blocs de documentation **commencent** par la balise `@`. Les lignes qui suivent, constituent un bloc de texte qui doit être recopié à l'identique, sans être interprété par *R*, dans le fichier `.tex`. Cette balise doit être suivie d'un blanc ou d'un caractère de fin de ligne. Si le reste de la ligne n'est pas vide, il est ignoré.

Les blocs de code **commencent** par la séquence `<<...>>=`. Le reste de la ligne est considéré comme étant du commentaire qui doit être ignoré. Les lignes qui suivent cette séquence, constituent un bloc de texte qui doit être traité par *R*.

1.3.1 La syntaxe *noweb*

L'insertion du code *R* et/ou des sorties (texte ou graphique) se fait au travers d'options (Tableau 1.1). Ces options sont de la forme `option=valeur`. La valeur peut être un nombre, une chaine de caractères ou une valeur

10. Elle est basée sur la syntaxe `noweb`

Argument	Défaut	Description
label		Étiquette du bloc de code
eval	true	Indique si le bloc de code doit être exécuté par *R*
echo	true	Indique si on veut voir apparaitre le code *R* dans le document
keep.source	true	Indique si on souhaite conserver l'ensemble du code *R*, y compris les commentaires
results	verbatim	Indique la présence et le format des sorties dans le document
print	false	Affiche tous les objets construits
strip.white	true	Supprime les lignes blanches
term	true	Mime le fonctionnement
fig	false	Indique que l'on souhaite exporter le graphique dans un fichier
include	true	Indique que l'on souhaite inclure un fichier graphique dans le document final
pdf	true	Le format d'exportation du fichier pdf
eps	false	Le format d'exportation du fichier eps
png	false	Le format d'exportation du fichier png
jpg	false	Le format d'exportation du fichier jpg

TABLE 1.1 – Liste des options de la commande Sweave

logique.

Les options peuvent être placées soit

— dans la séquence << ... >>=, sous la forme <<opt=valeur>>=. Dans ce cas, leur portée est locale ; elle ne concerne que le bloc de code qui suit.

— dans la commande \SweaveOpts{}, sous la forme \SweaveOpts{opt=valeur}. Dans ce cas, leur portée est globale et concerne tous les blocs de code qui suivent la commande. Cette commande ne doit être utilisée que dans un bloc de documentation. Elle est généralement utilisée dans le préambule LaTeX pour fixer la valeur de certaines options. Mais elle peut apparaitre plusieurs fois dans le document ; les effets des options sont alors cumulatifs.

Toutes les options ont une valeur par défaut et seules les options dont on souhaite modifier la valeur sont précisées. La séquence minimale signalant le début d'un bloc de code est : <<>>=.

Plusieurs options peuvent être utilisées simultanément. Elles sont alors séparées par une virgule ; par exemple, <<opt1=valeur,opt2=valeur,..., opt5=valeur>>=.

Les options et les valeurs peuvent être écrites en caractères majuscules, minuscules et même en mélangeant les 2 types de caractères ; par exemple,<<OPT1=VALEUR,opt2=valeur,..., optN=vALEUr>>=. Dans la suite du document on utilisera la forme en caractères minuscules pour les options et les valeurs.

Les espaces présents dans la séquence <<>>= sont ignorées ; <<opt1 = valeur>>=, <<opt1= valeur>>= et <<opt1=valeur>>= sont des formes équivalentes.

L'ordre d'apparition des options n'a pas d'importance. Il est équivalent d'écrire <<opt1=valeur,opt2=valeur>>= ou <<opt2=valeur,opt1=valeur>>=.

Remarques : Dans la suite du document, l'effet des différentes options sera décrit en les positionnant dans la séquence << ... >>=. On aurait le même résultat, en utilisant la commande \SweaveOpts{}. La valeur par défaut d'une option, sera mise en *italique*.

1.3.2 L'option label=

Chaque bloc de code *R* possède une étiquette. Cette étiquette est construite automatiquement par la fonction Sweave() mais elle peut aussi être spécifiée avec l'option label=. Si cette option est placée au début de la liste des options, on peut n'indiquer que la valeur. Il est équivalent d'écrire

<<label=étiquette,opt=valeur>>= ou <<étiquette,opt=valeur>>=.

Dans tous les autres cas, on doit utiliser la forme complète. On obtient une erreur si on écrit

`<<opt=valeur,étiquette>>=` au lieu de `<<opt=valeur,label=étiquette>>=`.

Cette étiquette permet l'identification du bloc de code

- lors du traitement du fichier `.Rnw` par *R*. Elle apparait dans l'affichage des résultats de la fonction `Sweave()` dans la console de *R* (voir la section 1.2.2) ;
- dans le fichier `.R` produit par la fonction `Stangle()` que l'on verra plus loin ;
- lors de la création de fichiers, contenant, par exemple, des graphiques.

Elle est aussi utile lorsque l'on souhaite réutiliser un bloc de code que l'on a déjà écrit. La syntaxe à utiliser est alors `<<étiquette>>`. Cette possibilité est intéressante en particulier lorsque l'on ne fait que des ajouts sur un code commun.

Dans l'exemple suivant, on a 2 blocs de code. Le premier sert à créer 2 objets (`a` et `b`). Il a pour étiquette la valeur `exemple`.

.Rnw

```
...
<<exemple>>=
a <- 3 + 2
b <- 5 + 7
@
...
<<>>=
rm(a,b) # on efface les objets
<<exemple>>
a * b
@
```

Dans le second bloc, on efface les objets `a` et `b` et on utilise cette étiquette pour les recréer.

```
> a <- 3 + 2
> b <- 5 + 7

> rm(a,b)  # on efface les objets
> a <- 3 + 2
> b <- 5 + 7
> a * b

[1] 60
```

Remarque : L'instruction `<<exemple>>`, dans le bloc 2, a été remplacée par le code contenu dans le bloc 1 puis a été exécuté par *R*.

1.4 Traitement du code *R*

Plusieurs options contrôlent l'inclusion du code *R* dans le fichier `.tex` et son exécution.

1.4.1 L'option echo=

La copie d'un bloc de code *R* dans le fichier `.tex` est contrôlée par l'option `echo=`. Cette option peut prendre l'une des 2 valeurs logiques suivantes :

— La valeur *true*, permet l'inclusion du code dans le fichier `.tex`. Le code est contenu dans un environnement LaTeX particulier : l'environnement `Sinput`. Cet environnement isole le code qui n'est pas évalué par LaTeX.
La copie se fait ligne par ligne. Elle s'arrête dès que le code génère une sortie (un affichage par exemple). La sortie est alors traitée dans un autre environnement LaTeX. Ensuite, la copie reprend dans un nouvel environnement `Sinput`. Un même bloc de code peut donc conduire à la production de plusieurs répétitions de l'environnement `Sinput`. Une illustration de ce comportement est donnée dans l'exemple de la page 5.

— Avec la valeur `false`, le code n'est pas inclus dans le fichier `.tex`. Ce comportement est valable pour l'ensemble du bloc de code.

Dans l'exemple suivant, le code *R* est composé de 3 instructions : 2 affectations et un affichage. On ne modifie que l'option `echo=` ; les autres options sont laissées à leur valeur par défaut.

Dans la partie gauche, on utilise la valeur par défaut alors que dans la partie droite on utilise l'option avec la valeur `false`.

L'option `echo=`*true* : `<<>>=`

```
> a <- 3 + 2
> b <- 5 + 7
> a
```

```
[1] 5
```

L'option `echo=false` : `<<echo=false>>=`

```
[1] 5
```

Attention : Cette option ne contrôle pas l'exécution du code. Dans l'exemple, le code *R* a été exécuté dans les cas (`true` ou `false`).

1.4.2 L'option keep.source=

La quantité de code copié, dépend de l'option `keep.source=`. Cette option peut prendre l'une des 2 valeurs logiques suivantes :

— La valeur `true` permet la copie du toutes les lignes contenues dans le bloc de code, y compris les commentaires (`#`), les lignes vides, etc.

— En utilisant la valeur `false`, seul le code exécuté par *R* est copié. Les lignes de commentaires, les lignes vides, etc, ne sont pas copiées.

Dans l'exemple suivant, on a inclus 2 lignes de commentaires (elles commencent par `#`) : ces lignes ne sont pas évaluées par *R*.

L'option `keep.source=`*true* : `<<>>=`

```
> # Fixe une valeur pour l'objet a
> a <- 3
> # Fixe une valeur pour l'objet b
> b <- 2
```

L'option `keep.source=false` : `<<keep.source=false>>=`

```
> a <- 3
> b <- 2
```

Avec la valeur `false`, seules les instructions exécutées par *R* sont copiées.

1.4.3 L'option eval=

L'option eval= contrôle l'exécution du code *R* et donc la production des sorties texte et/ou graphique. Cette option peut prendre l'une des 2 valeurs logiques suivantes :

— La valeur *true* entraine le traitement du bloc de code par *R* et l'exécution des instructions.

— La valeur false indique à *R* d'ignorer les instructions qui suivent. Il n'y a pas de production, pas de création d'objet, d'affichage, etc.

Dans l'exemple suivant, on crée deux blocs de code. Dans le premier, on crée 2 vecteurs et dans le second, on demande l'affichage des objets créés lors de la session. Dans la partie gauche, l'option eval= a sa valeur par défaut. Dans la partie droite, l'option eval= a comme valeur false dans le premier bloc (les objets a et b ne sont pas créés) et true dans le second. Comme il n'y a pas d'objets, on obtient une liste vide.

L'option eval=*true* : <<>>=

```
> a <- 3
> b <- 2
```

L'option eval=*true* : <<>>=

```
> ls()
```

```
[1] "a" "b"
```

L'option eval=false : <<eval=false>>=

```
> a <- 3
> b <- 2
```

L'option eval=*true* : <<>>=

```
> ls()
```

```
character(0)
```

Avec l'option eval=false, le code *R* n'est pas exécuté . . . mais il est affiché.

Attention : L'exécution n'est pas liée à l'affichage du code. On peut faire exécuter des instructions *R* qui ne seront pas copiées dans le fichier .tex.

1.4.4 L'option expand=

L'option expand= permet le développement ou non d'un code réutilisé [11]. Cette option peut prendre deux valeurs logique :

— La valeur *true*. Le code réutilisé est développé.

— La valeur false. Seule l'étiquette du bloc de code est affichée.

L'option expand=*true* : <<>>=

```
> a <- 3
> b <- 2
```

L'option expand=false : <<expand=false>>=

```
> a <- 3
> b <- 2
```

1.4.5 La fonction Stangle()

La fonction Stangle() permet d'extraire du document source (ici le *Exemple.Rnw*), l'ensemble des blocs de code.

```
> Stangle("Exemple.Rnw", encoding="UTF-8")
```

```
Writing to file Exemple.R
```

Elle crée, en sortie, un fichier ayant comme extension .R (ici le fichier *Exemple.R*) qui ne contient que les instructions *R*. Ce fichier peut ensuite être exécuté sous *R* avec la fonction source().

11. Cette option ne donne pas les résultats attendues !! . . . à vérifier

Exemple.R

```
1  ### R code from vignette source 'Exemple.Rnw'
2
3  ###################################################
4  ### code chunk number 1: Exemple.Rnw:5-7
5  ###################################################
6  somme <- 3 + 2; somme
7  produit <- 3 * 2
8
9
10 ###################################################
11 ### code chunk number 2: histogramme
12 ###################################################
13 set.seed(123)
14 x <- rnorm(1E3)
15 hist(x)
16
17
```

Chaque bloc de code est copié et identifié dans une zone de commentaire. Le premier bloc apparait en lignes 4 à 8. Son identifiant est en ligne 5. Dans cet identifiant, on retrouve le nom du fichier source, suivi du numéro des lignes contenant les instructions dans le fichier source.

Le second bloc de code apparait en lignes 11 à 16. Ce bloc avait une étiquette que l'on retrouve comme identifiant en ligne 12.

Attention : Les instructions contenues dans la commande `\Sexpr{...}`, sont ignorées.

1.5 Traitement des sorties texte

La plupart des sorties produites par *R* sont destinées à être incluses dans le document final [12]. On peut les inclure dans le texte courant ou sous la forme de blocs de résultats dans un environnement particulier de LaTeX (l'environnement `Soutput`).

1.5.1 La commande `\Sexpr{...}`

On peut vouloir intégrer dans un paragraphe, le résultat d'une instruction *R* ou le contenu d'un objet. Pour cela on va utiliser la commande `\Sexpr{...}`. Dans l'exemple présenté au début de ce document, on a utilisé cette possibilité dans la ligne suivante :

```
9 et calculer la valeur du logarithme, en base 2, de 9: $\log_2(9) = \Sexpr{log(9,base=2)}$.
```

R exécute l'instruction et insère le résultat de son calcul à la place de la commande. Cette commande peut aussi être utilisée dans une formule (hors texte), la légende d'une figure ou d'un tableau, dans les cellules d'un tableau, etc.

La commande `\Sexpr{...}` ne permet d'afficher qu'un seul élément. Lorsque le résultat ou l'objet *R* est un vecteur, une matrice, etc., seul le premier élément est affiché. Si l'objet `vecteur` contient 10 éléments :

```
> vecteur <- 10:1
> vecteur

 [1] 10  9  8  7  6  5  4  3  2  1
```

le résultat de la ligne suivante :

12. On verra plus loin que l'on peut aussi cacher certaines sorties

```
Le vecteur \sortie{vecteur} contient les éléments: \Sexpr{vecteur}
```

affiche : 10. Soit le premier élément du vecteur.

Ce n'est pas réellement un problème, car on peut toujours faire afficher les résultats dans un bloc ou ne faire afficher que le résultat final après une série de calculs effectués dans un bloc de code que l'on cachera.

Comme on peut le voir dans la figure 1.2, le résultat de la commande `\Sexpr{...}` est donné quelquefois avec une très grande précision. On peut arrondir le résultat en utilisant les fonctions de *R*, comme la fonction `round()` ou alors au travers de commandes LaTeX.

Sous LaTeX, si on utilise le package *siunitx*, on peut utiliser la commande `\num` . Cette commande offre de nombreuses possibilités dont celle de pouvoir arrondir les nombres et celle de les afficher avec un séparateur décimal adapté à la langue du document.

Pour un document rédigé en français et une précision de 5 chiffres significatifs, on écrirait :

```
\sisetup{output-decimal-marker={,},round-mode=figures,round-precision=5}
... logarithme, base 2, de 9, $\log_2(9)=$ \num{\Sexpr{log(9,base=2)}}.
```

soit 3,1699.

1.5.2 L'option `results=`

L'insertion de blocs de sorties dans le fichier `.tex` se fait au travers d'options que l'on va détailler dans cette section. Ces options peuvent être utilisées pour moduler l'ampleur des sorties. Les sorties sont copiées dès qu'elles apparaissent.

L'option `results=` gère à la fois la copie et le format de sortie des résultats. Elle peut prendre 3 valeurs (chaine de caractères) :

- La valeur *verbatim* écrite sans `" "`, indique que les sorties doivent être copiées sans modifications (ajouts ou retraits) dans le fichier `.tex`. Les sorties sont copiées dans un environnement particulier : l'environnement `Souput`. Cet environnement isole le texte des sorties qui n'est pas évalué par LaTeX.
- La valeur `tex`, toujours sans `" "`, indique que les sorties doivent être incluses dans le document mais elles ne sont pas contenues dans l'environnement `Soutput`. Cette possibilité permet d'inclure des résultats mis en forme directement au format LaTeX.
- La valeur `hide`, sans `" "`, indique que les sorties ne doivent pas être copiées dans le fichier `.tex`.

Soit le bloc de code suivant :

```
<<>>=
set.seed(123)
(essai <- sample(1:20,5) )
cat("\\Huge{",essai,"}")
```

qui crée un vecteur `essai`, puis l'affiche 2 fois : la seconde fois accompagné d'une commande LaTeX.

Dans la partie gauche, on utilise la valeur `verbatim` et dans la partie droite la valeur `tex`.

L'option `results=verbatim` : `<<>>=`

```
> set.seed(123)
> (essai <- sample(1:20,5) )

[1]  6 15  8 16 17

> cat("\\Huge{",essai,"}")

\Huge{ 6 15 8 16 17 }
```

L'option `results=tex` : `<<results=tex>>=`

```
> set.seed(123)
> (essai <- sample(1:20,5) )
```

[1] 6 15 8 16 17

```
> cat("\\Huge{",essai,"}")
```

6 15 8 16 17

La valeur `verbatim` permet l'insertion des résultats dans l'environnement `Soutput`. Cet environnement LaTeX ne permet pas l'interprétation des commandes LaTeX comme la commande `\Huge`.

La valeur `tex` permet aussi l'insertion des résultats mais sans l'environnement `Soutput`. Elle permet ainsi l'interprétation de la commande LaTeX `\Huge` (augmentation de la taille de la police). Les sorties apparaissent sous la forme d'un texte dans la fonte utilisée pour le reste du document.

Enfin, la valeur `hide` n'inclut pas les résultats dans le fichier.

L'option `results=hide` : `<<results=hide>>=`

```
> set.seed(123)
> (essai <- sample(1:20,5) )
> cat("\\Huge{",essai,"}")
```

1.5.3 L'option `print=`

L'option `print=` gère le contenu/la quantité des résultats affichés. Elle peut prendre 2 valeurs logiques :

- La valeur *false* : seuls les objets dont on demande l'affichage ou les sorties produites par certaines fonctions *R*, sont inclus dans le document.
- La valeur `true` : tous les objets crées ou utilisés sont affichés. Chaque instruction du code est passé à la fonction `print()`.

L'option `print=false` : `<<>>=`

```
> exprint <- seq(10,100,10)
> summary(exprint)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   10.0    32.5    55.0    55.0    77.5   100.0
```

L'option `print=true` : `<<print=true>>=`

```
> exprint <- seq(10,100,10)

 [1]  10  20  30  40  50  60  70  80  90 100

> summary(exprint)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   10.0    32.5    55.0    55.0    77.5   100.0
```

Dans le second cas, l'objet `exprint` est affiché alors que l'on ne l'a pas demandé explicitement.

1.5.4 L'option `strip.white=`

L'option `strip.white=` permet de supprimer les lignes vides présentes dans les sorties. Elle peut prendre 3 valeurs (chaines de caractères).

— La valeur *true* permet de supprimer la première et la dernière lignes vides des sorties.

— La valeur false permet de conserver toutes les lignes vides des sorties.

— Enfin, la valeur all supprime toutes les lignes vides.

L'option strip.white=*true* : <<>>=

```
        Chi-squared test for given probabilities

data:  HairEyeColor
X-squared = 557.6216, df = 31, p-value <
2.2e-16
```

L'option strip.white=false : <<strip.white=false>>=

```
        Chi-squared test for given probabilities

data:  HairEyeColor
X-squared = 557.6216, df = 31, p-value <
2.2e-16
```

L'option strip.white=all : <<strip.white=all>>=

```
        Chi-squared test for given probabilities
data:  HairEyeColor
X-squared = 557.6216, df = 31, p-value <
2.2e-16
```

1.5.5 L'option term=

L'option term= permet de mimer le fonctionnement de *R*. Elle peut prendre 2 valeurs logiques :

— La valeur *true* : on affiche ce qui est produit lors d'une session *R*. Le résultat des affectations n'est pas affichés, etc.

— La valeur false : seules les demandes explicites (fonctions print(), cat(), etc.) produisent un affichage.

L'option term=true : <<>>=

```
> Rdt <- 10
> Rdt

[1] 10

> print(Rdt)

[1] 10
```

L'option term=false : <<term=false>>=

```
> Rdt <- 10
> Rdt
> print(Rdt)

[1] 10
```

Dans le second cas, l'affichage de l'objet Rdt nécessite l'utilisation de la fonction print().

1.6 Traitement des sorties graphiques

Sweave() permet aussi de construire (option fig=) et d'inclure (option include=) automatiquement des graphiques dans le document final.

1.6.1 Les options `fig=` et `include=`

La création des graphiques se fait avec l'option `fig=`. Cette option peut prendre l'une des 2 valeurs logiques suivantes :

— La valeur *`false`* : le graphique est construit, affiché dans une fenêtre graphique de *R* mais il n'est pas exporté sous forme de fichier.

— La valeur `true` : le graphique est construit, affiché dans une fenêtre graphique de *R* et exporté sous forme de fichier.

Cet argument n'autorise la création que d'un seul graphique par bloc de code.

L'inclusion dans le document final dépend de l'argument `include=`. Cet argument prend l'une des 2 valeurs logiques suivantes :

— La valeur *`true`* : elle permet l'inclusion du graphique dans le document. En fait elle génère les commandes LaTeX qui le permette.

— La valeur `false` : pas d'inclusion du graphique.

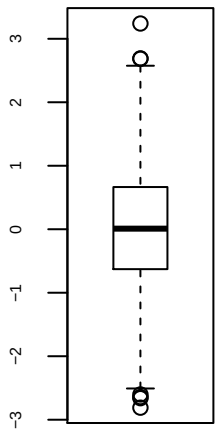
Cet argument permet de moduler l'inclusion des graphiques et les créant à un endroit mais en les incluant à un autre endroit.

`fig=`*`false`*

```
> set.seed(123)
> x <- rnorm(1E3)
> boxplot(x,xlab="",cex.axis=.5)
> title("")
```

`fig=true`, `include=`*`true`*

```
> set.seed(123)
> x <- rnorm(1E3)
> boxplot(x,xlab="",cex.axis=.5)
> title("")
```



`fig=true`, `include=false`

```
> set.seed(123)
> x <- rnorm(1E3)
> boxplot(x,xlab="",cex.axis=.5)
> title("")
```

1.6.2 Les formats : vectoriels ou images

On doit aussi indiquer le format de sortie du graphique. Quatre (?) formats sont disponibles :

— le format `.pdf`. L'option est `pdf=`, avec 2 valeurs possibles, *`true`* ou `false`. Par défaut, cette option a la valeur *`true`*.

— le format `.eps`. L'option est `eps=`, avec 2 valeurs possibles, `true` ou *`false`*. Par défaut, cette option a la valeur *`false`*.

— le format `.png`. L'option est `png=`, avec 2 valeurs possibles, `true` ou *`false`*. Par défaut, cette option a la valeur *`false`*.

— le format `.jpg`. L'option est `jpg=`, avec 2 valeurs possibles, `true` ou *`false`*. Par défaut, cette option a la valeur *`false`*. Dans le cas de ce format, le fichier produit a comme extension `.jpeg`.

On peut cumuler les options et obtenir le même graphique dans chacun des formats précédents. Dans l'exemple, on utilise le compilateur PDFLaTeX qui ne reconnait pas le format `.eps`.

Attention : Lorsqu'il existe plusieurs fichiers avec le même nom mais des extensions différentes, c'est le compilateur LaTeX qui choisit celui qui sera utilisé.

1.6.3 Les options `width` et `height`

On peut aussi définir la taille des graphiques avec les options `width` et `height`. Par défaut, la taille est fixée à 6 pouces (soit environ 15,250 cm).

1.7 Pour aller plus loin

Références

Leisch F. (2006). Sweave User Manual.

La page Noweb: a simple, extensible tool for literate programming.