Knitr et LyX

15 septembre 2016

Présentation.

Knitr est un acronyme provenant de l'anglais 'to knit' qui veut dire tricoter et 'r' pour le langage R[R]. Il s'agit d'un module qui s'intègre dans le logiciel LyX[lyx] qui permet l'écriture, l'exécution et l'affichage de résultats de code écrit en R [1] au sein d'un document textuel. Réalisé par yihui xie[knitr], Knitr peu être utilisé directement dans R puisqu'il s'agit d'une library de R, mais alors une bonne connaissance de LaTeX est nécessaire. LyX est d'un abord plus aisé et représente un changement moins important que LaTeX pour des personnes utilisant Word ou Libre office habituellement .

Syntaxe de base.

Premiers pas.

Une fois R et LyX installer, ouvrir LyX et créer un nouveau document puis fixer quelques paramètres pour ce faire aller dans **l'onglet** ▷ **Document** ▷ **Paramètres,** dans la partie **classe de document** choisissez **Articles**(classe standard) , dans la partie **Modules** choisissez **Rnw**(knitr) et dans la partie **langue**s choisissez **Françai**s (et Encodage Unicode utf8), appuyer sur **OK** et vous êtes prés à commencer un nouveaux document.

Le code R doit être écrit dans un environnement particulier, ceux-ci sont accessible dans le menu **Insertion**, **Inserts personnalisables** puis **Bloc**. La première ligne de chaque Bloc doit commencer par une syntaxe particulière ≪ ≫= et les lignes suivantes sont le code R à exécuter. L'exemple de la figure 1 ci-dessous ou la première ligne après ≪ ≫= produit 100 valeur d'une variable normal de moyenne 1, puis la deuxième ligne renvois la moyenne de l'échantillon et la troisième affiche l'histogramme correspondant.

Bloc 2

```
<<>>=
var<-rnorm(100,1)
mean(var)
hist(var)
```

Fig. 1

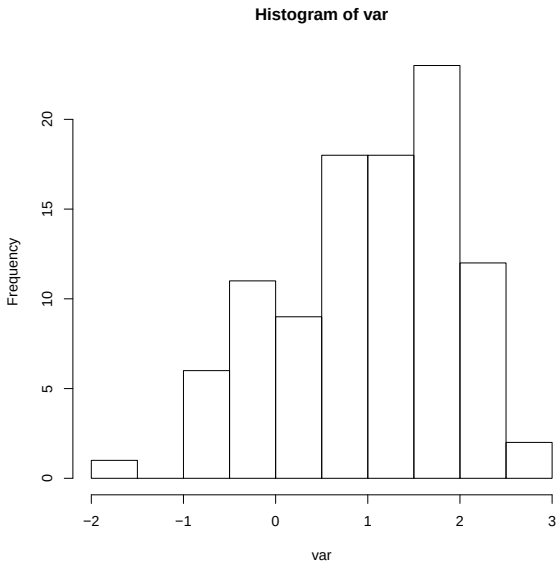
Dans le document final cela produit :

```
var<-rnorm(100,1)
mean(var)

## [1] 1

hist(var)
```

[1] Il est à noter que d'autres langage que R peuvent être utiliser avec knitr tel que le python, le bash, le C...



Bien évidement il existe de nombreuses options permettant de formater, de redimensionner les graphiques, d'afficher ou non le code, de présenter les résultats sous forme de tableau etc..., tous ses paramètres sont à inclure sur la première ligne entre les $\ll$ $\gg=$.

Il est à noter qu'un clique sur la partie grisée du bloc permet de replier ou de déplier le code écrit dedans ce qui permet selon les circonstance de ce concentrer soit sur le code soit sur le texte du document. Les flottants de figures permettant d'afficher des graphiques obéissent au même mécanisme.

Expression en ligne.

Il est possible d'évaluer des expressions R en ligne c'est a dire d'insérer directement dans le texte d'une phrase le résultat d'une variable, soit via l'onglet **Insertion**, **Inserts personnalisables, S/R expression** et on écrit directement le code R et le résultat sera affiché dans le document final, soit de faire un insère TEX et écrire l'expression `\Sexpr{code R a évaluer}`. ainsi la moyenne de l'échantillon vaut `\Sexpr{mean(var)}`, donnera : ainsi la moyenne de l'échantillon vaut 1.03.

Les variables déclarées dans un bloc sont accessibles dans tout le document et dans tous les blocs suivants et expressions en lignes.

Les options.

Dans la suite du document on indiquera les valeurs de chaque options entre parenthèses, la première valeur est la valeur par défaut, les valeurs possible sont également indiquées booléennes, numériques, ou littérales. En pratique on ne doit pas écrire les options entre parenthèse mais comme suit :

```
<<nomLabel, option1=TRUE, option2='text2', option3=2, Options=TRUE>>=
```

Chaque bloc (portion de code) peu recevoir un nom (label) qui doit être unique. Si aucun label n'est écrit knitr en génère un comme suit "unnamed-chunk-i" i pour le i^{ieme} chunk ou bloc.

Il est à noter que l'on peut écrire les options d'une autre manière en se positionnant dans un bloc puis un click droit fait apparaître un onglet dans lequel l'une des valeurs est **Options** qui a sont tour fait apparaître un encart dans lequel on peut écrire les options séparer par des virgules, dans ce cas il n'est pas nécessaire d'encadrer les valeurs par $\ll \gg=$.

Evaluation du code.

- **eval** (TRUE/FALSE ou numérique) évalue le code ou non. La valeur peu être également un vecteur numérique qui sélectionne alors les expression de R à évaluer (ligne de code), par ex. eval=c(1,5) : évalue la première et la cinquième ligne et c(-3) évalue toutes les lignes sauf la troisième, enfin c(2 :5) évalue de la deuxième ligne à la cinquième ligne.

Résultats literaux.

- **echo** (TRUE/FALSE ou numérique) affiche ou pas le code source (affiche le résultat dans tous les cas), la valeur peu être numérique fonctionne comme pour 'eval'.
- **results** ('markup' ou autre).
  - 'markup' :écrit le résultat au format LaTeX.
  - 'asis' : écrit les résultats bruts sans formatage (pour xtable p. ex).
  - 'hold' : retient tous les résultats et les affiches à la fin du chunk.
  - 'hide' : cache les résultats (mais pas le code source).
- **warning** (TRUE/FALSE) affiche ou pas les messages d'avertissements produit par la fonction R warning().
- **error** (TRUE/FALSE) affiche ou non les messages d'erreurs. (en général en cas d'erreur l'évaluation est stoppé et la compilation du document est également stoppé)
- **message** (TRUE/FALSE) affiche ou non les messages produit par la fonction message() de R.
- **split** (FALSE/TRUE) sépare ou non la sortie de R en des fichiers séparés et les inclus après dans le document avec la fonction **input**.
- **include** (TRUE/FALSE) inclus ou non la sortie de R dans le document final. si **include**=FALSE rien ne sera écrit dans le document mais le code sera quand même évalué et les graphiques générés, ceux-ci pourront être insérés manuellement après par l'utilisateur.Option à utiliser lorsque que le code et le résultat n'a aucun intérêt à être vu pour le lecteur.

Mise en forme du code.

- **tidy** (FALSE/TRUE) permet d'avoir un code bien structuré et propre, grâce à la fonction tidy.source() du package formatR. Remplace les " $=$ " par l'affectation classique " $<-$", indente le code dans les boucles 'for' , etc... Voir également les options de tidy.source().
- **prompt** (FALSE/TRUE) affiche ou non le caractère du prompt " $>$ " .
- **comment** ('\#\#' ; caractère) caractère affiché en début de ligne de chaque résultats.
- **highlight** (TRUE ; FALSE) permet de colorer le code.
- **size** ('normalsize' ; caractère) fixe la taille de la police pour la sortie LaTeX. (voir les options du package highlight pour les valeurs possibles : size = c("normalsize", "tiny", "scriptsize", "footnotesize", "small", "large", "Large", "LARGE", "huge", "Huge")
- **background** ('\#F7F7F7', caractère ou numerique) permet de fixer la couleur du fond. La couleur est donnée en rgb (red, green, blue) soit en hexdecimale, valeur entre 0 et 1 pour chaque composante ou couleur nommée, qui sont celles disponible dans R (red, springgreen3, ...) voir la fonction colors() de R pour une liste complète.

Cache.

- **cache** (FALSE/TRUE) met en réserve ou non le code chunk. Lorsque $<< cache = TRUE >>=$ le chunk n'est pas évaluer mais le résultat est chargé dans le document et provient d'une précédente évaluation. Cette évaluation se fait lors de la première évaluation ou s'il y a eu une modification du code chunk, l'ensemble des objets crées sont enregistrés au format R. Option très utile lorsque le code charge des données volumineuses ou que le traitement des données est long on évite ainsi de ré évaluer le code à chaque nouvelle compilation du document final.
- **cache.path** ('cache/' ; caractère) préfixe utilisé pour les fichiers cache. par défaut les fichier sont sauver dans un dossier 'cache' placé dans le dossier courant de travail.
- **cache.vars** (NULL) un vecteur de caractère contenant le nom des variables destinées à être sauvé dans le 'cache'. Par défaut toutes les variables sont sauvées.
- **dependson** (NULL ; caractère ou numerique) est un vecteur de nom de chunk pour spécifier de quel autre chunk il dépend. Cette option ne s'applique qu'aux chunk avec un cache=TRUE. Ainsi lorsqu'un chunk sera mis à jour les chunk dépendant (chunk fils) le seront également.
- **autodep** (FALSE ; TRUE) analyse le code au niveau des variable global pour connaître les dépendance entre les chunks. L'option dependson est alors inutile dans ce cas.

Graphiques.

- **fig.path**('figure/' ; caractère) préfixe utilisé pour le nom des fichiers graphiques. (fig.path et le nom du chunk sont assemblé pour former le nom du fichier)
- **fig.keep** ('high' ; caractère) la manière dont les graphiques seront produit.
  - 'high' : garde uniquement les graphiques de haut niveau (fusionne les changements de bas niveau)
  - 'none' : n'affiche aucun graphique.
  - 'all' : garde toutes les figures (l'ajout d'un paramètre graphique de bas niveau produira un nouveau graphique)
  - 'first' : garde uniquement le premier graphique.
  - 'last' : garde uniquement le dernier graphique.
- **fig.show** ('asis' ; caractère) affichage et disposition des graphiques.
  - 'asis' : affiche les figures à l'endroit du code ou elles sont générées.
  - 'hold' : retient les figures et les affichent à la fin du chunk.
  - 'animate' : rassemble toutes les figure pour créer une animation, si le chunk génère plusieurs figures.
  - 'hide' : les figures sont générées et enregistrés sur le disque dur mais ne sont pas affiché. Elles peuvent être ensuite inclus dans le document manuellement dans d'autres endroits ou dans des environnement particulier (ex. enrobage de figure).
- **dev** ('pdf' pour une sortie LaTeX et 'png' pour une page HTML/ markdown ; caractère) indique le nom de la fonction qui sera utilisé pour la sortie graphique. Une vingtaine d'autres fonctions sont disponible pour autant que les packages soient installer dans R (bmp, svg, jpg, ...)
- **dpi** (72, numérique) pour les sorties bitmap indique le nombre de pixels par pouce.
- **fig.width**, **fig.height** (7, numérique) hauteur et largeur des graphiques en pouce.
- **out.width**, **out.height** (NULL, caractère) hauteur et largeur dans le document final, qui peut être différent de ceux indiqués dans fig.width et fig.height du fait d'une mise à l'échelle dans le document final.

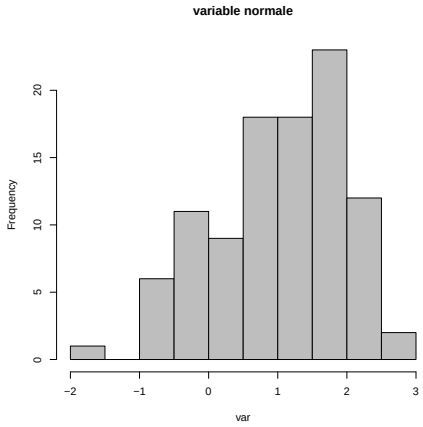
  - quelques exemple latex : '.8\\linewidth', '3in' ou '8cm'.
  - en HTML : '300px'.

- **out.extra** (NULL, caractère) option supplémentaire, ex out.extra=' angle=90 ', provoque une rotation de 90° de la figure.
- **fig.align** ('default', caractère) permet l'alignement des figures, valeurs possibles : 'left', 'center', 'right' .
- **fig.cap** (NULL, caractère) légende utilisé pour la figure.
- **fig.scap** (NULL, caratère) légende courte, tous les mots placé avant un '.', ';' ou ':' seront utilisé comme légende.
- **fig.lp** ('fig', caractère) mot utilisé comme préfixe pour les figures, par défaut concatène le préfixe et le nom (label) du chunk : 'fig :label1' .
- **fig.pos** (' ', caractère) indique la position à utiliser dans le document, les valeurs sont celles utilisées en LaTeX dans \begin{figure}[fig.pos].
  - 'h' Place le flottant ici, c'est-à-dire à l'endroit auquel il apparaît dans le texte source.
  - 't' Position en haut de la page.
  - 'b' Position en bas de la page.
  - 'p' Place sur une page particulière réservée aux flottants.
  - '!' Passe outre les paramètres internes que Latex utilise pour déterminer une position optimale des flottants.

Quelques exemples de graphiques.

Un graphique simple de 6cm de large, centré et positionner ici.

```
<<echo=T, out.width='6cm', fig.align='center', fig.pos='!', Options=T>>=
hist(var, main='variable normale', col="grey")
```



Pour le titre il vaut mieux éviter d'utiliser le paramètre **main**='titreDeLaFigure' ou la fonction **title**('titreDeLaFigure') de R mais plutôt utiliser le paramètre **fig.cap**='titreDeLaFigure' du bloc, cela permet en plus d'écrire le titre, de référencer la figure dans le document final lorsque que l'on écrit en LaTeX.

Dans LyX la meilleur méthode est de placer le code R dans un flottant de figure comme ci-dessous (menu **Insertion** ▷ **Flottant** ▷ **Figure**) cela permet de référencer la figure dans LyX (dans la liste des figures) , dans le document final et de pouvoir insérer une référence croisée au sujet de la figure suivante c'est à dire la fig.2 au sein du texte.

flottant : Figure

Bloc 6

```
<<out.width='6cm', fig.align='center', echo=T, Options=T>>=
hist(var, col="grey", main="")
```

Figure 2: histogramme d'une variable normale fig:histogramme-d'une-variable


```
<<out.width='6cm', fig.align='center', echo=T, Options=T>>=
hist(var, col="grey", main="")
```

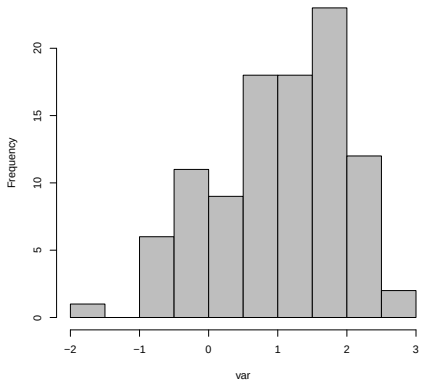



Fig. 2: histogramme d'une variable normale

Graphiques multiples.

Les graphiques multiples ne posent pas problèmes particulier on utilise le paramètre **mfrow()** de la fonction **par()** par exemple pour diviser la sortie graphique en autant de ligne et de colonnes souhaitées.

```
<<out.width='7cm', fig.align='center', Options=T>>=
par(mfrow=c(2,2)); # divise le graphique en 2 lignes et 2 colonne
x<-rnorm(100,1,1)
hist(x, col="grey"); #histogramme
boxplot(x, horizontal=T, col="grey"); #boite a moustache
d<-density(x)
hist(x,freq=F, col="blue") #histogramme
lines(d, col="red")      #courbe de densité associé a l'histogramme
qqplot(x,rnorm(100,0,1))         #distribution quantile/quantile
```

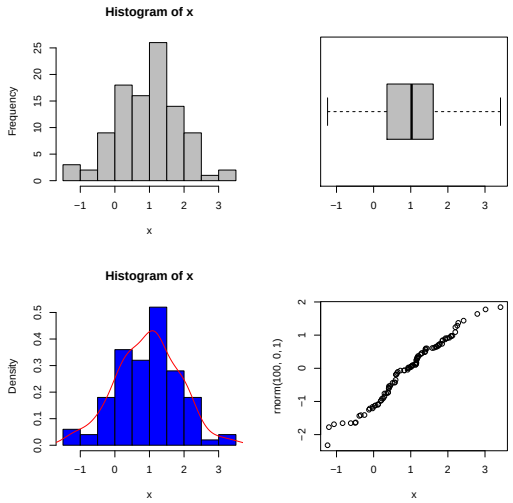


Fig. 3: figures multiples

Les figures double ou triples peuvent être réalisées dans LyX en incluant plusieurs flottants de figures dans un flottant de figure, on obtient alors des sous-flottants de figures numérotés par des lettres. On peu écrire directement le code R dans le flottant de figure comme ci dessous ce qui produit la figure 4.

flottant : Figure

sous-flottant : Figure

Bloc 8

```
<<out.width='7cm', fig.align='center', echo=FALSE>>=
hist(x, col="grey")
```

Sous-Figure a: histogramme

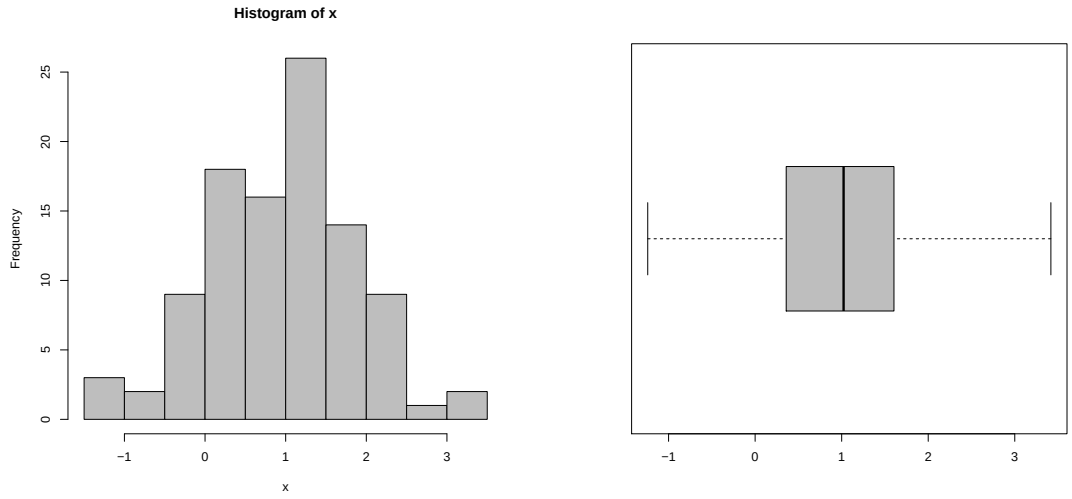
sous-flottant : Figure

Bloc 9

```
<<out.width='7cm', fig.align='center', echo=FALSE>>=
boxplot(x, horizontal=T, col="grey");
```

Sous-Figure b: boite à moustache

Figure 4: Variable normale



(a) histogramme

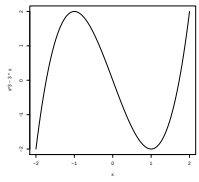
(b) boite à moustache

Fig. 4: Variable normale

Enrobé de figure.

Une autre manière de faire consiste à générer le graphique à un endroit quelconque du document et à mettre l'option **fig.show**='hide' ce qui ne produit aucune figure dans le document mais celle-ci est quand même écrite sur le disque dur il ne reste plus qu'a intégrer la figure de manière classique comme pour cet enrobé de figure.

```
<<graph, fig.show='hide', Options=T>>=
curve(x^3-3*x, -2, 2)
```



En effet knitr utilise un système de cache lorsqu'une figure est produite, le code R qui produit la figure est évalué à la première compilation du document, la figure est alors intégré au document et enregistrée sur le disque dur dans le dossier **figure**. Lors d'une prochaine compilation si le code de la figure n'a pas changer knitr intégrera directement la figure déjà produite sans de nouveau évaluer le code ce qui permet d'accélérer la compilation. On peut tirer partie de ce mécanisme lorsque l'on veut intégrer une figure dans un environnement qui ne peut accepter du code R, par exemple dans un enrobé de figure avec window{} du package picinpar comme la figure ci-contre. Il faut alors compiler deux fois le document une fois pour créer l'image et l'enregistrer sur le disque et une autre fois pour intégrer l'image dans le document.

Graphique avec ggplot2.

```
<<graphggplot, out.width="8cm", Options=T>>=
library(ggplot2)

                        ## Loading required package: methods

mtcars$gear <- factor(mtcars$gear,levels=c(3,4,5),
labels=c("3 vitesses","4 vitesses","5 vitesses"))
mtcars$am <- factor(mtcars$am,levels=c(0,1),
labels=c("Automatique","Manuelle"))
mtcars$cyl <- factor(mtcars$cyl,levels=c(4,6,8),
labels=c("4cyl","6cyl","8cyl"))
qplot(mpg, data=mtcars, geom="density", fill=gear, alpha=I(.5),
main="Distribution de la consommation", xlab="Miles par Gallon",    ylab="Densité")
```

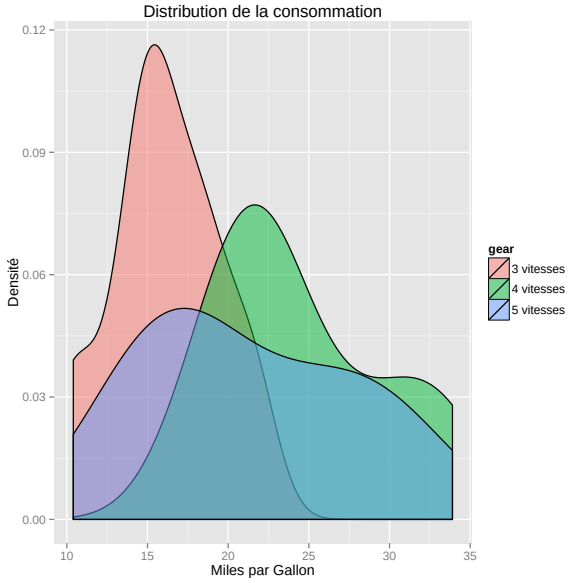


Fig. 5: Graphique ggplot2

Graphique en perspective.

```
<<out.width='12cm', Options=T>>=
require(grDevices)
x<-seq(-10,10,length=30) ;
y<-x ;
f<-function(x,y){r<-sqrt(x^2+y^2) ;10*sin(r)/r}
z<-outer(x,y,f) ;
z[is.na(z)]<-1;
op<-par(bg="white") ;
persp(x,y,z,theta=30,phi=30,expand=0.5,col="lightblue") ;
```

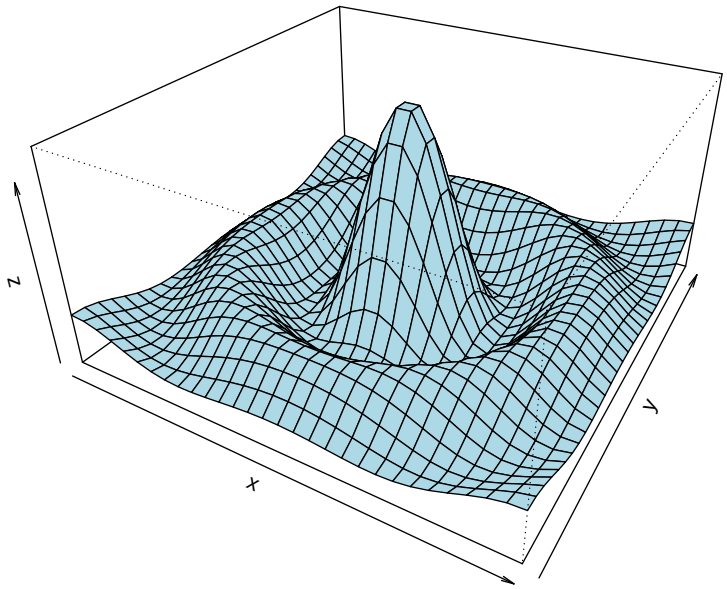


Fig. 6: perspective

Cartographie.

La représentation de cartes se fait de la même manière que les graphiques, seules quelques bibliothèques R supplémentaires sont nécessaires, notamment **sp** et **rgdal**[sp][rgdal].

R peut ouvrir de nombreux types de fichiers vectoriels et rasters via la bibliothèque **rgdal**. Le type le plus courant est le format shape (.shp).

readOGR() permet de charger un fichier cartographique dans R (shp, mif, dxf ...), pour le type shapes layer=nom du fichier sans l'extension et dsn=nom du dossier contenant le fichier. (pour les autres type voir la documentation[rgdal])

on accède au dataframe de l'objet spatial avec le slot **@data**.

```
<<out.width="7cm",message=F ,Options=T>>=
library(rgdal)
ecosse<-readOGR(layer='Scottish', dsn='data', verbose=F)
plot(ecosse)
box()
```

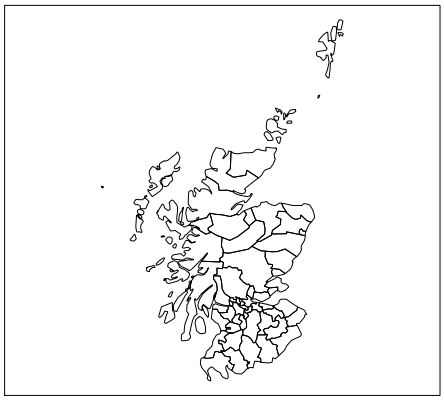


Fig. 7: Écosse

D'autres bibliothèques peuvent améliorer l'apparence (sp, rgrs, ggmap, cartography, tmap...) liste non exhaustive, voir également Analyse spatiales sous R de Nicolas Casajus [gisonr].

spplot de la librairie sp, le paramètre zcol spécifie la colonne à cartographier.

```
<<out.width="7cm", Options=T>>=
head(ecosse@data)

##           NAME ID pcaff Observed Expected
## 0   Sutherland 12    16        5      1.8
## 1        Nairn 13    10        3      1.1
## 2    Inverness 19     7        9      5.5
## 3 Banff-Buchan  2    16       39      8.7
## 4     Bedenoch 17    10        2      1.1
## 5   Kincardine 16    16        9      4.6

spplot(ecosse, zcol=c('pcaff'))
```

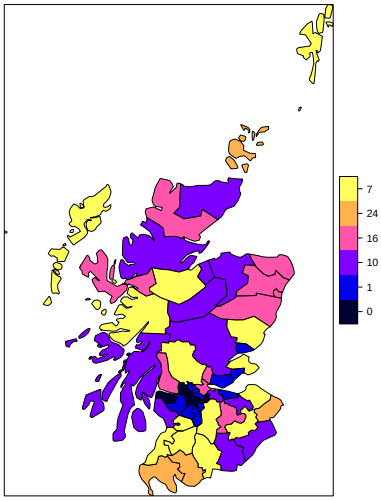


Fig. 8: Écosse

Autre exemple avec le package cartography [cartography].

```
<<size="scriptsize",out.width="8cm", Options=T>>=
library("cartography")
opar <- par(mar = c(0,0,1.2,0)) #fixe les marges

cols <- carto.pal(pal1 = "green.pal", n1 = 2, pal2 = "red.pal", n2 = 4) #creer la palette de couleur

plot(ecosse, border = NA, col = NA, bg = "#A6CAEO") #couleur du fond (mer)

ecosse@data$Sir<-ecosse@data$Observed/ecosse@data$Expected #vecteur a cartographier

choroLayer(spdf = ecosse,
        df = ecosse@data,
        var = "Sir",
        breaks = c(0,0.5,1,2,4,6),
        col = cols, # colors
        border = "grey40",
        lwd = 0.5,
        legend.pos = "right",
        legend.title.txt = "SIR \ncancer de la lèvre",
        legend.values.rnd = 2,
        add = TRUE)

layoutLayer(title = "Cancer de la lèvre en Ecosse",
        author = "jp LELEU",
        sources = "source",
        frame = FALSE,
        col = "grey40",
        scale = 0,
        coltitle = "black",
        north = TRUE)
```

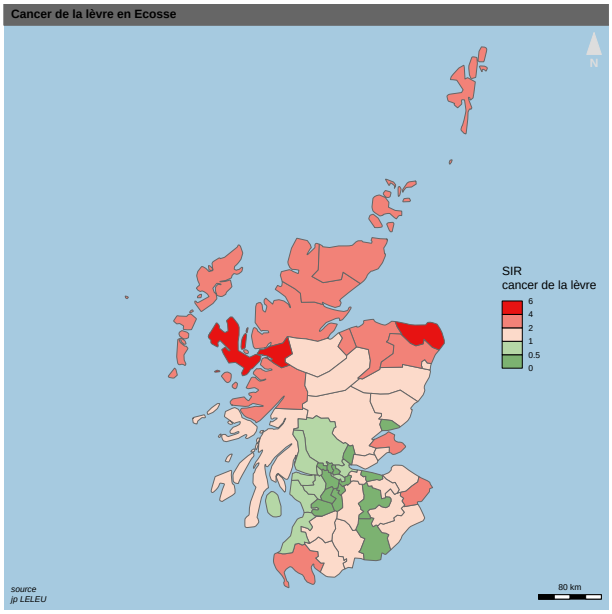


Fig. 9: Incidence du cancer de la lèvre en Ecosse

Et enfin avec le package tmap[tmap] (thematic map) sûrement le plus abouti des packages pour afficher des cartes dont appréciera la concision du code ou une seule ligne de code est nécessaire pour afficher la figure 11, il est à noter que tmap suit à peu près la même «grammaire» graphique que ggplot2.

```
<<tmap1, out.width="8cm", Options=TRUE>>=
library(tmap)
data(Europe)
qtm(Europe)
```

Fig. 10: Europe

```
<<tmap2,out.width="8cm", Options=TRUE>>=
qtm(Europe, fill="life_exp")+tm_scale_bar()
```

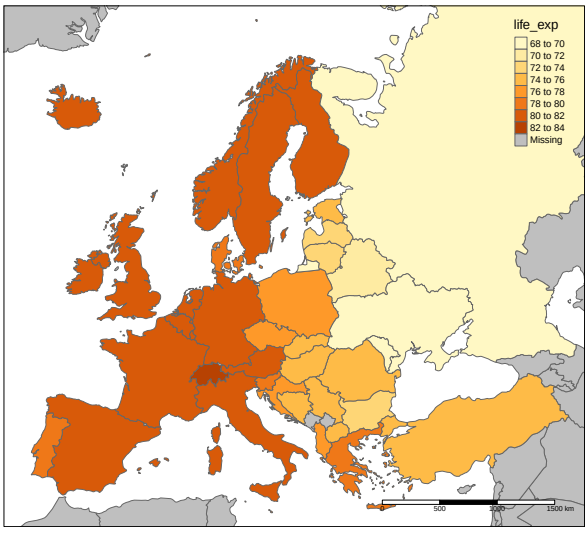


Fig. 11: Espérance de vie en Europe

child Document.

- **child** (NULL, caractère) un vecteur de noms de fichiers à exécuter et à intégrer au document principal.

Options globales.

Il est possible si on utilise fréquemment une option de la changer de manière globale pour l'ensemble du document avec la fonction opts_chunk$set(option1=val1, option2=val2), évidement à mettre au début du document. Il faut noter aussi que opts_chunk$set n'est pas à mettre dans les options mais à écrire comme du code R.

ex : n'affiche pas le code R exécuté ni les avis d'alertes et redimensionne les graphiques à 8cm de large pour l'ensemble du document.

```
<<optionGlobales, eval=F, Options=T>>=
opts_chunk$set(echo=FALSE, warning=FALSE, out.width='8cm')
```

Externalisation du code.

knitr permet de sourcer du code R c'est à dire importer un fichier texte contenant du code R et exécuter ce code grâce à la fonction read_chunk(). Il ne s'agit pas d'une option, la fonction doit être intégré dans un chunk comme une fonction R classique.

```
<<eval=F, Options=T>>=
read_chunk('nomDeFichier.R')
```

Ou 'nomDeFichier.R' est fichier texte contenant le code R, celui-ci doit contenir des noms de chunk pour pouvoir les réutiliser par la suite ceux-ci peuvent être écris de 2 manières :

- - - - nom1 - - - -
code R a exécuter
- - - - nom2 - - - -
autre code R
@ knitr nom3
autre code R

nom1, nom2 et nom3 peuvent ensuite être utilisés dans le document comme un chunk normal, il suffit de l'appeler par son nom :

<< nom1 >>=

Dans le but d'externaliser du code R on peut également utiliser les fonctions R classiques à savoir **load**(«nomFichier.RData ») pour charger un ou plusieurs objets R ou **source**(«nomFichier.R ») pour charger un script écrit en R.

L'idée pouvant être étendu au partage d'objet R entre plusieurs documents. Imaginons que l'on ait à écrire un article et sa présentation en style «power point » (par exemple à l'aide de la classe Beamer), l'article comprend le corps du texte ainsi que le traitement des données sous R en enregistrant les résultats principaux graphiques, tableaux, etc... qui devront être présent dans la présentation à l'aide de la fonction **save**(objet1, objet2,..., file= »nomFichier.RData »). Ces objets peuvent être récupérés dans la présentation à l'aide de la fonction **load**(«nomFichier.RData »). Ainsi la compilation de l'article va produire les différents objets et les écrire sur le disque dur, et la compilation de la présentation va charger les objets produit par l'article, il y a donc une mise a jour automatique des résultats entre les deux documents, ce qui évite les fastidieux copier-coller de graphiques de tableaux plus ou moins bien mise en page.

Utilisation de xtable.

Xtable[xtable] est une librairie de R qui permet d'exporter un objet qui peut être mis sous forme d'un tableau, cette mise en forme est faites avec des balise LaTeX, pour afficher un tableau on doit utiliser l'option **results**='asis'.

Exemple tableau sans mise en forme.

```
data(iris)
print(head(iris))

##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1          5.1         3.5          1.4         0.2
## 2          4.9         3.0          1.4         0.2
## 3          4.7         3.2          1.3         0.2
## 4          4.6         3.1          1.5         0.2
## 5          5.0         3.6          1.4         0.2
## 6          5.4         3.9          1.7         0.4
##   Species
## 1  setosa
## 2  setosa
## 3  setosa
## 4  setosa
## 5  setosa
## 6  setosa
```

Le même tableau avec xtable, ne pas oublier **results**='asis'.

```
<<results='asis', Options=T>>=
library(xtable)
data(iris)
print(xtable(head(iris),caption='Iris') )
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.10	3.50	1.40	0.20	setosa
2	4.90	3.00	1.40	0.20	setosa
3	4.70	3.20	1.30	0.20	setosa
4	4.60	3.10	1.50	0.20	setosa
5	5.00	3.60	1.40	0.20	setosa
6	5.40	3.90	1.70	0.40	setosa

Tab. 1: Iris

Des options sont disponible pour la fonction xtable comme le placement (floating=TRUE par défaut), ou via à la fonction print comme caption ou scale pour modifier l'échelle comme pour le tableau 2.

```
<<results='asis', echo=T, Options=T>>=
print(xtable(head(iris),floating=TRUE,caption='Iris\\label{tabscale}'),label="tabscale",include.rownames=F
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.10	3.50	1.40	0.20	setosa
4.90	3.00	1.40	0.20	setosa
4.70	3.20	1.30	0.20	setosa
4.60	3.10	1.50	0.20	setosa
5.00	3.60	1.40	0.20	setosa
5.40	3.90	1.70	0.40	setosa

Tab. 2: Iris

options de xtable les plus courantes :

- **caption**='texte' : titre du tableau.
- **label**='texte' : label utilisé pour le référencement.

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.10	3.50	1.40	0.20	setosa
2	4.90	3.00	1.40	0.20	setosa
3	4.70	3.20	1.30	0.20	setosa
4	4.60	3.10	1.50	0.20	setosa
5	5.00	3.60	1.40	0.20	setosa
6	5.40	3.90	1.70	0.40	setosa

Tab. 3: Iris

- **floating**=(TRUE ; FALSE) placement du tableau dans le document flottant ou non.
- **align**="r|llrc" permet d'aligner les colonnes 'r' right, 'l' left ,'c' center et '|' permet de positionner une ligne verticale, chaque lettre représente une colonne, p{3cm} permet de faire une colonne de 3 cm.
- **digits**=0 ou autre spécifie le nombre de décimale à afficher dans le tableau.
  - exemple met les colonnes 2 et 4 sans decimale.
  - (res<-xtable(res) ; digits(res)[c(2,4)]<-0
- **include.colnames**=(TRUE, FALSE) affiche le nom des colonnes.
- **include.rownames**=(TRUE, FALSE) affiche le nom (numéro) des lignes.
- **rotate.rownames**=(FALSE, TRUE) rotation de nom de lignes.
- **rotate.colnames**=(FALSE, TRUE) rotation des nom de colonnes.

Options de print :

- **caption.placement**=('bottom', 'top') : placement du titre.
- **table.placement**=('','h','b','t','p',' !') : placement du tableau. voir **fig.pos page 5.**
- **latex.environment**= ('NULL', 'center') : permet de centrer ou non le tableau.
- **tabular.environment**='longtable' : permet d'afficher des tableau long (sur plusieurs pages).
- **scalebox**=0.7 : multiplie par 0.7 la taille du tableau.
- **hline.after**=c(1) : ajoute une ligne après le ou les numéros de colonnes spécifiés.

L'utilisation de l'option **label** dans LyX , qui rappelons-le permet de référencer et citer les tableaux dans le texte présente une une petite particularité.

Dans le code suivant on remarque que label est spécifier 2 fois, une première fois dans le caption de la fonction xtable précédé de \\ et dans la fonction print. L'un des label est interprété dans R et l'autre dans le compilateur TEX , bien évidement le label doit être identique dans les 2 définitions. La référence au tableau ainsi créé se fera dans un insère TEX sous forme \ref{nomLabel}.

```
<<results='asis', echo=T, Options=T>>=
print(xtable(head(iris),caption="Iris\\label{TabIris}"), label="TabIris", table.placement=" !")
```

Le tableau3 est maintenant référencé.

Dans LyX il est préférable de mettre le tableau dans un flottant de tableau, le titre et le référencement y sont plus aisé, il faut alors mettre l'option floating=FALSE

```
<<results='asis', Options=TRUE>>=
print(xtable(head(iris)), floating=FALSE)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.10	3.50	1.40	0.20	setosa
2	4.90	3.00	1.40	0.20	setosa
3	4.70	3.20	1.30	0.20	setosa
4	4.60	3.10	1.50	0.20	setosa
5	5.00	3.60	1.40	0.20	setosa
6	5.40	3.90	1.70	0.40	setosa

Tab. 4: xtable inclus dans un flottant de tableau deLyX

Astuces.

Définition d'une nouvelle options.

Dans ce document de nombreux blocs présentent l'option : «Options=TRUE » cette option n'existe pas dans knitr à l'origine, elle a pour but d'afficher dans la sortie les options des blocs à titre pédagogique. knitr possède un mécanisme qui permet de modifier ou de définir de nouvelle options grâce à la fonction knit_hooks$set(). Nous allons voir à titre d'exemple la définition de cette nouvelle option.

```
knit_hooks$set(Options=function(before, options,envir){
if(before)
{paste("\\begin{alltt}","<<",paste(options)[length(options)],">>= \n","\\end{alltt}", sep="")}
else
{paste("",sep="")}
})
```

La nouvelle options nommée «Options » est appelé deux fois, une fois avant l'évaluation du code R la variable 'before' est alors égale à TRUE et une fois après l'évaluation du code,' before' est égal à FALSE. Ce mécanisme permet d'écrire avant et après le résultat renvoyer par R, par exemple des balises LaTeX. La variable 'options' correspond à l'ensemble des options possible d'un bloc et la dernière partie de cette variable correspond aux options écris par l'utilisateur. Au final on insère les options écrite par l'utilisateur entre des << >>= dans un environnement {alltt} (similaire à verbatim) avant les résultats des blocs et rien après les résultats.

Formatage des nombres.

Par exemple pour que les nombres soient affichés avec un arrondis à deux chiffres après la virgule pour l'ensemble du document on peut fixer des options directement dans R avec digits=2.

```
x<-0.1234567
options(digits=2, width=50)
x

## [1] 0.12

#voir ?options pour plus de détails
```

Références

[lyx] https ://www.lyx.org/WebFr.Home

[R] R Development Core Team (2008). R : A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http ://www.R-project.org.

[xtable] http ://cran.r-project.org/web/packages/xtable/xtable.pdf

[knitr] http ://yihui.name/knitr/

[rgdal] Roger Bivand, Keitt Tim, Rowlingson Barry, Edzer Pebesma, Sumner Michael, Hijmans Robert. Bindings for the Geospatial Data Abstraction Library. https ://r-forge.r-project.org/projects/rgdal/

[sp] Edzer Pebesma and Roger Bivand. Classes and methods for spatial data. http ://cran.r-project.org/web/packages/sp/sp.pdf

[gisonr] http ://qcbs.ca/wiki/_media/gisonr.pdf

[cartography] https ://cran.r-project.org/web/packages/cartography/index.html

[tmap] https ://cran.r-project.org/web/packages/tmap/index.html