Utiliser R Markdown pour créer des documents dynamiques

par Kevin Cazelles

09 June 2020

Le package rmarkdown permet de créer des documents dynamiques qui intègrent des morceaux de code R et ce qu’ils génèrent (dont figures et tableaux). Ce document est une introduction à R Markdown qui passe en revue différents aspects techniques nécessaires à la bonne compréhension et l’utilisation du package rmarkdown. Le code source de ce document est en lui-même un exemple d’application du package. Le lecteur trouvera également un ensemble de références disponibles en ligne pour aller plus loin.

# Avant-propos

Ce document est une introduction à R Markdown conçu pour répondre aux objectifs suivants:

1. comprendre ce qu’est R Markdown;
2. comprendre les liens entre R, Markdown et Pandoc;
3. apprendre la syntaxe Pandoc Markdown;
4. apprendre à créer des documents dynamiques avec le package rmarkdown.

Ce document a lui-même été édité avec R Markdown et peut-être converti en une page internet (au format HTML, .html), au format Word (.docx) ou encore en document portable (.pdf). Bien que dans “R Markdown”, il y ait “R” et “Markdown”, ce document n’est pas une introduction à R (<http://www.r-project.org>). Le contenu est accessible aux utilisateurs débutants et pour les lecteurs intéressés qui ne connaissent pas encore R, une courte introduction à R est nécessaire. Ils pourront consulter l’une des différentes introductions disponibles sur le site du CRAN: <https://cran.r-project.org/other-docs.html>.

Ce document est déployé automatiquement par GitHub Actions (voir <https://github.com/KevCaz/Rmarkdowndocfr/actions>), lors de la dernière compilation, la version 2.2 du package rmarkdown a été utilisée. De plus amples informations relatives à l’environnement dans lequel le document a été compilé sont indiquées ci-dessous.

sessionInfo()

## R version 4.0.0 (2020-04-24)  
## Platform: x86\_64-pc-linux-gnu (64-bit)  
## Running under: Ubuntu 20.04 LTS  
##   
## Matrix products: default  
## BLAS/LAPACK: /usr/lib/x86\_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.8.so  
##   
## locale:  
## [1] LC\_CTYPE=en\_US.UTF-8 LC\_NUMERIC=C   
## [3] LC\_TIME=en\_US.UTF-8 LC\_COLLATE=en\_US.UTF-8   
## [5] LC\_MONETARY=en\_US.UTF-8 LC\_MESSAGES=C   
## [7] LC\_PAPER=en\_US.UTF-8 LC\_NAME=C   
## [9] LC\_ADDRESS=C LC\_TELEPHONE=C   
## [11] LC\_MEASUREMENT=en\_US.UTF-8 LC\_IDENTIFICATION=C   
##   
## attached base packages:  
## [1] stats graphics grDevices utils datasets methods base   
##   
## loaded via a namespace (and not attached):  
## [1] compiler\_4.0.0 magrittr\_1.5 tools\_4.0.0 htmltools\_0.4.0  
## [5] yaml\_2.2.1 Rcpp\_1.0.4.6 stringi\_1.4.6 rmarkdown\_2.2   
## [9] knitr\_1.28 stringr\_1.4.0 xfun\_0.14 digest\_0.6.25   
## [13] rlang\_0.4.6 evaluate\_0.14

rmarkdown::pandoc\_version()

## [1] '2.3.1'

# Introduction

Dans de nombreux milieux professionnels, à des fins de communication diverses, sont produits régulièrement des documents intégrant des analyses de données (tableaux, figures, tests statistiques, etc.). Pour créer de tels documents, il faut être en mesure de manipuler des données les analyser et créer des figures pour les intégrer dans le document final. [R](http://www.r-project.org) est un langage de programmation qui répond à ces besoins avec un grand nombre de package qui permettent de manipuler et traiter un spectre très large de données et de les visualiser efficacement. Ce langage offre également la possibilité d’intégrer code et les produits du code (résultats de tests, tableaux, figures, etc.) directement dans un document qui est alors qualifié de dynamique. Le package le plus utilisé pour créer des documents dynamiques est la package rmarkdown l’intégration de R dans un document écrit avec Markdown et qui peut être converti en de nombreux formats de document (dont pdf, word, html). C’est ce package qui est l’objet de ce document, bien qu’il nous faudra en mentionner d’autre ainsi que d’autres langages et outils pour comprendre ce package, en commençant par Markdown.

Écrire un document en Markdown, quésaco? Markdown est un langage de balisage léger, c’est-à-dire un langage dans lequel on peut utiliser des ensembles de caractères spécifiques (des balises) pour délimiter une zone de texte pour laquelle un formatage associé (e.g. **text en gras**) est appliqué. Markdown est aujourd’hui très répandu sur Internet, à tel point que vous pourriez une des différentes syntaxes existantes sans me le savoir. Sans être exhaustif, il est utile de donner un peu plus de détails sur ce point pour bien comprendre ce quel Markdown on utilise avec R Markdown. La syntaxe originale de Markdown est le fruit du travail de John Gruber[[1]](#footnote-25), programmeur, bloggeur et baladodiffuseur de Philadelphie en collaboration avec Aaron Swartz (lui même connu pour avoir participer a la création de Creative Commons et son tragique destin qui fut l’objet d’un film). Sur le site de John Gruber, “daringfireball”, Markdown est décrit depuis décembre 2004 et on peut même y télécharger la version 1.0.1 (voir <http://daringfireball.net/projects/markdown>). L’idée de départ est simple et élégante : produire un langage léger qui simplifie les balises HTML (utilisé par tout les sites Internet). L’idée n’est pas tant de remplacer le HTML mais plutôt d’en augmenter l’efficacité d’écriture et de fait, il est beaucoup plus rapide d’écrire en Markdown qui couvre les opérations de formatage les plus courantes (listes, hyperliens, etc.). Notons qu’il existe d’autres langages qui répondent aux me objectif, par example ReStructuredText[[2]](#footnote-28).

Après la publication de Markdown, John Gruber a cessé de travailler sur Markdown[[3]](#footnote-30) et d’autres développeurs, sans doute séduits par le langage, ont proposé différentes additions syntaxiques. Il s’agissait surtout de lever certaines limitations tout en préservant l’esprit d’origine. Ci-dessous, en voici une liste non exhaustive de différentes variantes Markdown:

* [GitHub Flavored Markdown (GFM)](https://help.github.com/articles/github-flavored-markdown/)
* [Kramdown](http://kramdown.gettalong.org/syntax.HTML#math-blocks)
* [Markdown Extra](https://michelf.ca/projects/php-markdown/extra/)
* [Multi Markdow](https://github.com/fletcher/MultiMarkdown/wiki/MultiMarkdown-Syntax-Guide#math-support)
* [Pandoc Markdown](http://pandoc.org/README.HTML#pandocs-markdown)

Fort heureusement, depuis 2014, CommonMark (<https://commonmark.org/>) propose une spécification (norme technique) pour Markdown de plus en plus utilisée[[4]](#footnote-38), ce qui signifie qu’en allant d’un outil à l’autre qui utilise cette spécification, il n’y a pas de questions à se poser quant à savoir ce qui marche ou non en terme de syntaxe (un problème parfois frustrant quand on utilise plusieurs outils qui utilisent différentes syntaxes Markdown), il suffit de se reporter à la spécification!

Il est important de souligner que R Markdown (voir <http://rmarkdown.rstudio.com>)[[5]](#footnote-41) utilise [la variante Markdown de Pandoc](https://rmarkdown.rstudio.com/lesson-8.html)[[6]](#footnote-43). Une précision relative à Pandoc (<http://www.pandoc.org>) s’impose: il s’agit, comme l’indique le site internet, d’un “convertisseur de document universel”, en une ligne de commande Pandoc convertit un document dans un format donné en un document d’un autre format. Par exemple, Pandoc permet de passer d’un fichier .tex (LaTeX) à un fichier .docx (Word)! La variante Markdown de Pandoc a été pensé pour rester fidèle à l’esprit originel de Markdown tout en incluant davantage d’éléments de document communs différent format de documents[[7]](#footnote-45). Pour rendre le document dynamique, R Markdown ajoute notamment les fonctionnalités de l’excellent package [knitr](http://cran.r-project.org/web/packages/knitr/index.html) qui intègre du code R et ce qu’il produit. Le document dynamique en question pourra être converti dans un grand nombre de format grâce à l’utilisation de [Pandoc](http://www.pandoc.org/installing.html). Les intérêts de R Markdown sont nombreux :

1. utiliser une syntaxe épurée pour distinguer la mise en page, le texte, le formatage du texte et le code R,
2. utiliser les fonctionnalités du package knitr pour créer morceaux de codes, tableaux et figures dynamiquement,
3. utiliser le format de sérialisation YAML pour personnaliser la mise en page des documents produits grâce à la fonction render(), elle-même appelée en quelques clics avec [RStudio](http://www.rstudio.com),
4. obtenir ainsi un document avec un contenu R dynamique en différents formats dont HTML (fichier .html), PDF (.pdf) et Word (.docx).

Il existe une documentation abondante relative à R Markdown, malheureusement presque exclusivement écrite en anglais. Pour les lecteurs capables de lire la la langue de Shakespeare, je recommande [le site officiel de R Markdown](http://rmarkdown.rstudio.com) sur lequel vous trouverez, entre autres:

1. un condensé d’utilisation sous forme de *Cheat Sheet*, voir <http://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf>;
2. un guide de référence disponible sur le site de RStudio <http://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf>.

Il y a également sur le site <https://bookdown.org> :

1. un livre en anglais très complet sur le sujet, par l’un des grands architectes de R Markdown, voir <https://bookdown.org/yihui/rmarkdown/>;
2. [un livre d’astuces (*cook book*)](https://bookdown.org/yihui/rmarkdown-cookbook/) en cours d’écriture.

Une section complète du site de l’auteur principal de knitr, <http://yihui.name/knitr/>, permet de bien comprendre l’intégration des morceaux de code et des résultats de ce code, contenu qui fait l’objet d’un livre de ce même auteur (Xie, 2017). Je tiens également à signaler la [présentation de Mansun Kuo sur Rpubs](http://rpubs.com/mansun_kuo/24330). Pour apprendre la syntaxe Pandoc Markdown, vous pouvez vous reportez à la très complète page internet écrite en français par Jean-Daniel Bonjour, <http://enacit1.epfl.ch/markdown-pandoc>.

Pour conclure cette introduction, je signale que la syntaxe [Pandoc Markdown](http://rmarkdown.rstudio.com/authoring_pandoc_markdown.html) est, à mon sens, très complète, mais elle ne permet pas une personnalisation totale de la mise en page de votre document. Il existe des moyens simples pour aller plus loin avec R Markdown, détaillé dans le livre d’astuce mentionné ci-dessus. Si toutefois vous souhaitez aller encore plus loin dans la mise en forme de votre document, vous pouvez utiliser une des trois approches suggérez ci-dessous. Néanmoins, avant d’envisager l’un de ces recours, il est très important de se demander si cette personnalisation est nécessaire.

1. vous pouvez travailler sur le document dans le format qui vous intéresse après l’avoir générer, c’est une solution qui a une limite évident: à chaque fois que le document sera compilé, les modifications post-compilation seront perdues,
2. vous pouvez utiliser un *template* personnel[[8]](#footnote-59), un fichier qui gère partie la mise en forme de votre document) associé à votre document,
3. vous pouvez toujours composé en un langage donné, p. ex. [HTML](http://www.w3.org/html/) ou [Latex](http://www.latex-project.org), mais les commandes non valides pour un format donné, seront ignorées.

# Utiliser un fichier R Markdown

## Organisation générale

Un fichier R Markdown (dont l’extension est .rmd ou .Rmd), c’est :

1. un document de texte plein qui utilise la variante syntaxique Pandoc de Markdown;
2. des blocs de code particuliers qui, en plus de pouvoir être utilisé pour présenter du code, le code va pouvoir être exécuter et le résultat du code va pouvoir être intégré dans le document. Ces blocs commencent et finissent trois accents graves (*backtick* ou *backquote* en anglais): ` et les trois accents graves ouvrant le bloc sont suivis d’une accolade qui commence par r ou R, par exemple

* ```{R name, option1, option2}  
  # code R à exécuter  
  ```
* ou encore
* ```{r option1}  
  # code R à exécuter  
  ```

1. enfin, un *Front Matter*[[9]](#footnote-65) YAML optionnel (au début du fichier, dans un bloc de trois tirets : ---) qui permet de donner des indications sur le ou les documents à générer.

Au début, il peut-être un peu difficile de se retrouver un tel fichier car on y croise différents langages, notamment

1. un langage de programmation pour les analyses, R;
2. un langage de balisage pour l’écriture du document, Markdown;
3. un langage de sérialisation de données pour spécifier les sorties du documents, YAML.

On peut ajouter qu’avec la variante syntaxique Pandoc de Markdown les symboles mathématiques TeX[[10]](#footnote-66) sont utilisés pour rendre facile l’écriture, entre autres, des équations,[[11]](#footnote-68) en un sens c’est un quatrième langage à connaître!

## Créer un fichier R Markdown

Créer un fichier R Markdown, c’est simplement créer un fichier dont l’extension est .Rmd ou .rmd, ce qui peut être vais avec n’importe quelle éditeur de texte, ou de code, ou même en ligne de commande. Par example, étant moi-même utilisateur du terminal et travaillant dans un environment Linux, j’entre dans ce dernier la commande:

$ echo "---\nauthor: Kevin Cazelles\n---" > mondoc.Rmd

et le fichier est créé! Pour les utilisateurs de R Studio, cela peut se faire en 2 clics, comme illustré sur les figures qui suivent. L’intérêt d’utiliser R Studio pour cette opération est que le fichier ainsi créé contient des indications relatives à l’utilisation du fichier en question.



Première étape, utilisez l’icône de création de nouveaux fichiers.



Seconde étape, choisissez le format de sortie désiré



Un document contenant différentes instructions et examples est généré.

## Spécifier les documents à obtenir avec YAML

Le site officiel <https://yaml.org/> indique:

[*Traduction*]

YAML est un standard de sérialisation de données pour tous langages de programmation adopté à l’humain.

Autrement dit, dans un fichier au format YAML (extension .yaml ou .yml), on entre des données qui pourront être utilisées dans par de nombreux langages de programmation et ce fichier est facile à lire par l’humain. Pour que le contenu du fichier soit effectivement utilisables dans un langage de programmation donné, il faut que le code ad hoc soit disponible. Pour les utilisateurs de R, c’est le package yaml[[12]](#footnote-75) qui permet de lire de tels fichiers et de les importer sous forme de listes.

Pour comprendre l’essentiel du fonctionnement du YAML, je vous recommande le tutoriel à l’URL suivante : <https://sweetohm.net/article/introduction-yaml.html> en vous concentrant sur la syntaxe et nous sur comment utiliser Python pour le faire. De manière générale:

nomduchamp: données

et on saute une ligne pour ajouter un autre champ:

nomduchamp: données  
nomduchamp2: données2

Un champ données peut contenir une chaine de charactère, une date, des chiffres ou encore une liste, pa

nomduchamp: [élément1, élément2]

ou encore

nomduchamp:  
 - élément1  
 - élément2

Noté l’indentation utilisée, c’est capital dans dans YAML car c’est Dans YAML, l’indentation est très importante.

et les sous-champ peuvent eux aussi être nommées

nomduchamp:  
 souchamp1: élément1  
 souchamp2:  
 souchamp2a: élément2a  
 souchamp2b: élément2b

Aussi, voici un exemple qui, bien que non-exhaustif, montre comment un fichier est structuré et comment l’important dans R (le fichier utilisé est disponible dans le répertoire GitHub <https://github.com/KevCaz/Rmarkdowndocfr/blob/master/aux/data_01.yaml>):

author: Kevin Cazelles  
date: 2020-06-06  
description: "Il s'agit d'un exemple, utilisez les guillemets pour les longs textes avec ponctuation!"  
# Un commentaire  
var1: [C, C++, Dart, Julia, Python, R, Rust]  
var2: y  
var3:  
 var3a:  
 x: 1  
 "y": 2  
 var3b:  
 - a  
 - b  
 var3c: |  
 utiliser plusieurs  
 lignes et garder les sauts de lignes

Pour l’importer avec le package yaml, j’utilise les commandes suivantes:

library(yaml)  
yaml.load\_file("aux/data\_01.yaml")

## $author  
## [1] "Kevin Cazelles"  
##   
## $date  
## [1] "2020-06-06"  
##   
## $description  
## [1] "Il s'agit d'un exemple, utilisez les guillemets pour les longs textes avec ponctuation!"  
##   
## $var1  
## [1] "C" "C++" "Dart" "Julia" "Python" "R" "Rust"   
##   
## $var2  
## [1] TRUE  
##   
## $var3  
## $var3$var3a  
## $var3$var3a$x  
## [1] 1  
##   
## $var3$var3a$y  
## [1] 2  
##   
##   
## $var3$var3b  
## [1] "a" "b"  
##   
## $var3$var3c  
## [1] "utiliser plusieurs\nlignes et garder les sauts de lignes"

Quelques remarques importantes “y, yes, true, TRUE” et “n, no, false, FALSE” seront interprétées comme variables booléennes et donc TRUE, c’est une fonctionnalité bien pratique mais a double tranchant: si j’utilise y, le champ sera lui-même indiqué comme TRUE et donc il faut ajouter des guillemets.

**WIP** **WIP** **WIP**

Les remarques précédentes sont générales et vous pouvez facilement utiliser YAML pour autre chose que R Markdown. Revenons au fichier R Markdown dans lequel on utilise YAML d’une manière plus spécifique. Premièrement, les indications sont en début de document. Deuxièmement les champs concernent le format de sortie du document dynamique édité avec R Markdown. Troisièmement, seuls les champs qui sont définis dans des templates sont valides. Arrêtons-nous sur ce dernier point. On ne peut pas utiliser n’importe quelle champ.

Pour un aperçu assez complet des options YAML utilisables dans fichier R Markdown, rendez-vous à la dernière page du [guide de référence](http://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf).

C’est un ensemble de métadonnées relatives aux documents qui vont être générés. Ces données sont alors soit intégrées dans la ligne de commande Pandoc, soit dans les *templates*.

À titre d’exemple, ce document, utilise un grand nombre d’options en voici une reproduction quasi-identique (sans le résumé complet et sans la majorité des commentaires que vous pouvez trouver dans le fichier source de ce document) :

---  
title: "Utiliser R Markdown pour créer des documents dynamiques"  
date: "09 June 2020"  
author: par Kevin Cazelles  
lang: fr  
abstract: "Le package [...] pour aller plus loin."  
fontfamily: fourier  
linestretch: 1  
fontsize: 10pt  
lof: yes  
output:  
 html\_document:  
 toc: yes  
 toc\_float: yes  
 highlight: default  
 theme: flatly  
 include:  
 after\_body: aux/footer.html  
 pdf\_document:  
 highlight: tango  
 toc: yes  
 toc\_depth: 3  
 fig\_caption: yes  
 keep\_tex: yes  
 latex\_engine: pdflatex #xelatex  
 number\_section: yes  
 includes:  
 before\_body: aux/license.tex  
 word\_document:  
 #toc: yes  
 fig\_caption: yes  
 highlight: pygments  
 md\_document:  
 variant: markdown\_strict  
bibliography: aux/mybiblio.bib  
csl: aux/journal-of-theoretical-biology.csl  
header-includes:  
 - \usepackage{fancyhdr}  
 - \pagestyle{fancy}  
 - \fancyfoot[CO,CE]{Documents dynamiques avec R Markdown}  
 - \fancyfoot[R]{\thepage}  
---

L’indentation reflète les groupes de spécification. Ainsi, de “pdf\_document:” à “word\_document:”, les commandes permettent de spécifier les sorties du document au format pdf. Notez que “toc” signifie “table of content” (table des matières), “toc\_depth” permet de sélectionner le niveau maximal de sous-titre affiché dans la table des matières. De même “tof” signifie “table of figure” (table des figures). L’option “number\_section:true” permet d’obtenir une numérotation des différentes parties. Les commentaires sont introduits par un “#”.

## Éditer le fichier Markdown

On écrit simplement du texte plein avec la syntaxe R markdown et on utilise les bloc de code R qui permettent d’éxécute du code

## Obtenir le document final (ou les documents finaux)

Quand le fichier est complet, et que le package rmarkdown est installé[^4], on utilise la fonction render() du package en lui indiquant le chemin de du fichier .Rmd. Si on est un utilisateur de RStudio, on peut cliquer sur Knit dans la barre contextuelle associée au fichier ( voir la documentation associée, ?render). Dans la suite du document, la function render() est utilisée:

render("ex\_Rmardown.rmd", "all")

L’argument “all” permet d’obtenir tous les fichiers pour lesquels il existe une spécification dans le fichier YAML. Les formats utilisés ici sont :

* [PDF](https://en.wikipedia.org/wiki/Portable_Document_Format)
* [HTML](http://rmarkdown.rstudio.com/html_document_format.html)
* [Word](http://rmarkdown.rstudio.com/word_document_format.html)

Les documents au format pdf requièrent [l’installation de Latex](https://www.tug.org/texlive/quickinstall.html). En effet, pour produire un pdf à partir d’un fichier “.rmd”, un premier fichier Latex est produit. Pour le lecteur qui désire conserver le fichier Latex utilisé pour produire le pdf, il suffit d’ajouter l’option “keep\_tex: true” dans les options relatives au format pdf dans le [Le fichier YAML][]. Pour le Markdown, un fichier “.md” est généré suivant la variante de Markdown précisée dans le YAML par “variant: markdown” (voir la section suivante). Le format “Tufte handout” est une mise en page particulière, avec marges larges où sont insérées les illustrations (dont les figures), que nous devons au chercheur Edward Tufte.

# Le variante Pandoc de Markdown

Dans cette partie, je détaille les éléments de formatage du texte proposés par Pandoc Markdown. L’ensemble est très bien présenté sur le [site de référence](http://rmarkdown.rstudio.com/authoring_pandoc_markdown.html)[[13]](#footnote-86) et très bien résumé à la première page du [guide de référence](http://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf). Pour une source en français, j’ai trouvé un excellent tour d’horizon nommé : “Élaboration et conversion de documents avec Markdown et Pandoc” (<http://enacit1.epfl.ch/markdown-pandoc/>) écrit par Jean-Daniel Bonjour. Notez que certains symboles sont réservés au formatage du texte. Cependant, quand leur affichage est requis, on les fait précéder du caractère d’échappement qui est, pour Markdown, l’antislash : “\”. Par exemple, j’entre :

\\ \& \# \$ \[

Pour obtenir : \ & # $ [. J’ajoute que certains sauts de ligne sont obligatoires pour obtenir la mise en page désirée (pour les notes de bas de page par exemple).

## Décoration du texte

* Pour écrire du *texte en italique*, vous avez deux possibilités :

\*le texte à mettre en italique\*  
\_le texte à mettre en italique\_

* Pour écrire du **texte en gras**, encore deux possibilités :

\*\*le texte à mettre en gras\*\*  
\_\_le texte à mettre en gras\_\_

* Pour écrire du ***texte en gras et italique***, utilisez :

md \*\*le \_texte en italique et en gras\_\*\*md

* Pour obtenir un ~~texte rayé~~, entrez :

~~texte rayé~~

* Pour écrire un élément en exposant, utilisez :

^texte en exposant^

* Pour écrire un élément en indice, tapez :

~texte en indice~

Attention, exposants et indices ne fonctionneront pas si l’élément à formater n’est pas exempt de tout espace. Notez qu’il n’y a pas de balises pour le soulignement du texte. Si vous souhaitez l’obtenir pour le HTML, vous pouvez utilisez la balise suivante:

<u>texte souligné</u>

Pour le pdf vous pouvez utiliser:

\underline{texte souligné}

Si vous utilisez l’une ou l’autre de ces balises, votre formatage sera moins général.

## Les titres

Le plus simple est d’utiliser un nombre croissant de # (ATX heading) pour descendre dans l’arborescence des titres:

# Un titre d'ordre 1  
## Un titre d'ordre 2  
### Un titre d'ordre 3

Il est aussi possible d’utiliser une série de “=” en dessous des titre de premier niveau et une ligne de “-” en dessous des titres de niveau 2. Cette option a la qualité de permettre de repérer facilement les titres dans le code source.

Un titre d'ordre 1  
==================  
Un titre d'ordre 2  
------------------  
### Un titre d'ordre 3

## Les listes

Les listes sont très intuitives en Markdown, alors qu’elles requièrent des balises un peu lourdes que ce soit en Latex ou en HTML. Dans les exemples donnés, il faut toujours séparer le texte principal de la liste par des sauts de ligne.

### Listes non numérotées

Pour obtenir une liste non numérotée j’entre :

\* objet 1,  
\* objet 2,  
\* objet 3.

ou bien :

+ objet 1,  
+ objet 2,  
+ objet 3.

ou encore :

- objet 1,  
- objet 2,  
- objet 3.

et même :

+ objet 1,  
\* objet 2,  
- objet 3.

Dans tous les cas, cela donne

* objet 1,
* objet 2,
* objet 3.

En utilisant une indentation de 4 espaces (ou une tabulation), on peut obtenir des listes hiérarchisées, ainsi:

- objet 1,  
 + machin 1  
 - chose 1  
 - chose 2  
 + machin 2  
- objet 2,  
- objet 3.

donne:

* objet 1,
  + machin 1
    - chose 1
    - chose 2
  + machin 2
* objet 2,
* objet 3.

En ajoutant au moins deux espaces à la fin des éléments d’une liste, chaque élément de la liste est formaté comme un paragraphe  :

* objet 1,
* objet 2,
* objet 3.

Pour alterner des listes avec du texte ou du code, il faut utiliser des sauts de lignes avec l’indentation adéquate. Ainsi, avec les lignes suivantes :

- élément 1&nbsp;:  
  
 Un petit texte qui pourrait expliciter ce qu'est l'élément 1.  
  
- machin 2:  
  
 for (i in 1:2) print(i)

j’obtiens :

* élément 1 :
* Un petit texte qui pourrait expliciter ce qu’est l’élément 1.
* machin 2:
* for (i in 1:2) print(i)

### Les listes non numérotées

Pour une liste numérotée, c’est aussi très simple, en entrant par exemple si je rentre :

1. machin 1,  
2. machin 2,  
3. machin 3.

J’obtiens:

1. machin 1,
2. machin 2,
3. machin 3.

Si les nombres ne sont pas écrits manière ordonnée, cela ne changera pas le résultat. Néanmoins, le premier nombre détermine le point de la liste. En écrivant :

3. machin 1,  
3. machin 2,  
3. machin 3,  
5. machin 4.

j’obtiens :

1. machin 1,
2. machin 2,
3. machin 3,
4. machin 4.

Pour ne pas se soucier des numéros, il existe un style par défaut :

#. machin 1,  
#. machin 2,  
#. machin 3.

on retrouve bien la première liste numérotée :

1. machin 1,
2. machin 2,
3. machin 3.

Plusieurs styles de numérotation sont disponibles, p. ex. :

#) élément 1  
 #) élément 2  
 #) élément 3  
 (1) truc 1  
 (2) truc 2  
 (5) truc 3  
 i. machin 1  
 i. machin 2  
 i. machin 3

nous donne :

1. élément 1
2. élément 2
3. élément 3
4. truc 1
5. truc 2
6. truc 3
   1. machin 1
   2. machin 2
   3. machin 3

Nous avons aussi la possibilité de mélanger les niveaux numérotés et les niveaux non-numérotés :

1. machin 1,  
 1. machin 1.1,  
 2. machin 1.2,  
2. machin 2,  
 - machin 2.1,  
 - machin 2.2,  
 - machin 3,  
3. machin 4,  
4. machin 5.

ce qui donne :

1. machin 1,
   1. machin 1.1,
   2. machin 1.2,
2. machin 2,
   * machin 2.1,
   * machin 2.2,

* machin 3,

1. machin 4,
2. machin 5.

Enfin, il possible de mettre manuellement fin à une liste en introduisant un commentaire entre les listes à séparer :

(1) truc 1  
 (2) truc 2  
 (3) truc 2b  
  
 <!-- end -->  
  
 (1) truc 3  
 (2) truc 4

ces lignes sont rendues ainsi :

1. truc 1
2. truc 2
3. truc 2b
4. truc 3
5. truc 4

## Blocs de citation

Pour utiliser un bloc de citation (la balise “blockquote” en HTML), il suffit d’utiliser “>” avant la citation. Ainsi les lignes suivantes :

> la citation est ajoutée comme ceci, elle nous donne une indentation adéquate  
pour une mise en page agréable dont le style peut être facilement travailler  
en html grâce au CSS.

deviennent :

La citation est ajoutée comme ceci, elle nous donne une indentation adéquate pour une mise en page agréable dont le style peut être facilement travailler en html grâce au CSS.

Pour ajouter une hiérarchie dans les citations, on entre :

> La citation de départ  
>  
>> une hiérarchie dans la citation

ce qui donne :

La citation de départ

une hiérarchie dans la citation

## Blocs de code

Pour l’ensemble des exemples que je présente sous forme de lignes de code, j’utilise un environnement simple qui utilise une police d’écriture à chasse fixe et affiche tous les caractères tels qu’ils sont entrés (les balises ne sont pas interprétées).

Il est possible d’ajouter des morceaux de code colorés selon la nature des éléments de la syntaxe d’un langage de programmation donné. Les morceaux de codes peuvent être écrits entre série de tildes (~) et une accolade est nécessaire pour préciser le langage[[14]](#footnote-96). Pour présenter un morceau de code C, par exemple :

~~~~~~~~~~~~~~~{.c}  
// Commentaire  
int c,d,g ;  
c=10 ;  
d=4 ;  
int func(int a,int b) { return a\*b; }  
g = func(c,d)  
printf("%i",g)  
~~~~~~~~~~~~~~~

devient :

// Commentaire  
int c,d,g;  
c=10;  
d=4;  
int func(int a,int b) { return a\*b; }  
g = func(c,d);  
printf("%i",g);

Plus simplement, on peut utiliser des bloc de trois apostrophes inversés (*backquote* en anglais) et enlever l’accolade et le point pour préciser le langage et ainsi, avec le code ci-dessous

```c  
// Commentaire  
int c,d,g ;  
c=10 ;  
d=4 ;  
int func(int a,int b) { return a\*b; }  
g = func(c,d)  
printf("%i",g)  
```

on obtient bien la même bloc de code que précédemment.

// Commentaire  
int c,d,g ;  
c=10 ;  
d=4 ;  
int func(int a,int b) { return a\*b; }  
g = func(c,d)  
printf("%i",g)

Dans le document, j’utilise justement ces blocs de code pour montrer comment utiliser Markdown. Une autre façon simple pour faire un bloc de code st d’ajouter simplement 4 espaces au début de chaque ligne. C’est d’ailleurs ce que j’utilise quand je montre l’utilisation des apostrophes inversés ci-dessus!

Un des grands intérêts de R Markdown est d’avoir des blocs de codes R qui peuvent être exécutés par R! Je détaille cette fonctionnalité plus bas.

## Les barres horizontales

Une ligne barre horizontale peut être ajoutée comme suit :

-------------

ou encore :

\* \* \* \* \*

## Les tables

La création facilitée de tableaux est l’une des extensions bien utile de Pandoc Markdown. Il existe plusieurs extensions pour faire des tableaux. Pour avoir des détails sur le sujet, je vous recommande la [page écrite par Jean-Daniel Bonjour](http://enacit1.epfl.ch/markdown-pandoc/#tableaux). Je présente ci-dessous un exemple de tableau de style “pipe table”:

| Aligné à gauche | Aligné au centre | Par défaut | Aligné à droite  
 | :------- | :-------: | ------ | -------:  
 | truc 1.1 | truc 2.1 | \*\*\_truc 3.1\_\*\* | truc 4.1  
 | truc 1.2 | truc 2.2 | ~~truc 3.2~~ | truc 4.2  
 | truc 1.3 | truc 2.3 | \*truc 3.3\* | truc 4.1  
  
 : La légende associé au tableau.

La légende associé au tableau.

|  |  |  |  |
| --- | --- | --- | --- |
| Aligné à gauche | Aligné au centre | Par défaut | Aligné à droite |
| truc 1.1 | truc 2.1 | ***truc 3.1*** | truc 4.1 |
| truc 1.2 | truc 2.2 | ~~truc 3.2~~ | truc 4.2 |
| truc 1.3 | truc 2.3 | *truc 3.3* | truc 4.1 |

Si vous préférez les obtenir grâce à une interface de type “WYSIWYG” (*What you see is what you get*), vous pouvez utiliser [ce générateur de table](http://www.tablesgenerator.com/markdown_tables).

## Mathématiques

### Symboles mathématiques

Pour utiliser les symboles mathématiques dans le texte, les commandes associées doivent être placées entre deux “$”. Bien sur, il faut connaître les combinaisons de caractère associées aux différents symboles. Ce sont les même que celles proposées par [Latex](http://latex-project.org/intro.html) et qui seront utilisées par [MathJax](http://www.mathjax.org) pour générées les expressions mathématiques dans le fichier HTML. Pour quelques exemples, [regarder ce site](http://www.suluclac.com/Wiki+MathJax+Syntax), pour quelque choses de plus complet, jetez un œil [ici](https://en.wikibooks.org/wiki/LaTeX/Mathematics). Voici tout de même quelques exemples :

* quelques lettres grecques :
* $\alpha$, $\beta$, $\delta$, $\lambda$, $\pi$, $\phi$, $\omega$, $\varpi$, $\vartheta$
* pour obtenir : , , , , , , , ,
* quelques symboles mathématiques :
* $\sum$, $\prod$, $\int$, $\infty$, $\lim$
* pour obtenir : , , , ,
* quelques combinaisons :
* $\mu \in\mathbb{R}$, $\lim\_{x \rightarrow 3} f(x)$
* pour obtenir : ,

### Équations

Pour faire des équations, il faut placer l’équation entre double $$. Créons une première équation :

$$\frac{vache}{oiseau} = \frac{2\pi}{l}$$

Pour utiliser des équations numérotées, il faut ajouter “(@label)” avant l’équation, je ré-utilise l’équation précédente :

(@eq1) $$\frac{vache}{oiseau} = \frac{2\pi}{l}$$

Je rajoute une seconde équation qui utilise les balises Latex pour créer un système :

(@eq2) $$\begin{array}{ccc}  
x^2+y^2 &=& z^2 \\  
xy &=& z  
\end{array}$$

La référence aux équations se fait en utilisant “(@label)” dans le texte. Ainsi, en écrivant “(@eq1)”, j’appelle l’équation (1). De même je peux faire référence à l’équation (2) en utilisant “(@eq2)”[[15]](#footnote-109).

## Les images

Pour insérer une image, deux solutions nous sont offertes :

1. la combinaison dite “inline” : !+[nom de l’image (inclus dans la légende)]+(adresse)+{#label}. Par exemple,

* ![Le logo de R](./images/Rlogo.png)



Le logo de R

1. la combinaison dite “reference” : !+[nom de l’image (inclus dans la légende)]+[id] et ailleurs dans le document le détail [id]+:+adresse. Par exemple

* [img2]: ./images/Markdown.png  
    
   ![Le logo sur la page du Markdown original][img2]{#fig:logo}

Le logo sur la page du Markdown original{#fig:logo}

Il existe deux possibilités pour faire des références à une figure. Malheureusement, aucune n’est idéale:

1- On peut injecter une balise \\label{lelabel} et utiliser \\ref{lelabel}, mais ce n’est valable que pour le pdf (pour un exemple voir la section “[Les figures produites avec R](#les-figures-produites-avec-r)”) 2- On peut appeler un filtre particulier de pandoc : [pandoc-fignos](https://github.com/tomduck/pandoc-fignos), cela demande une installation à la main (mais cela peut vite changer),

## Références

### Liens hypertextes

Les liens hypertextes sont utilisés sous la forme [groupe de mots sur lequel cliquer]+(adresse du lien). Pour créer un lien vers la page Markdown de Wikipedia, j’utilise :

[Markdown](https://fr.wikipedia.org/wiki/Markdown)

et voila le lien vers la page [Markdown](https://fr.wikipedia.org/wiki/Markdown) de Wikipedia.

### Notes de bas de page

On peut produire une note de bas de page en plaçant la balise “[^id]” là où la note doit être insérée. Pour préciser le texte qui y est associé, on utilise: “[^id]:texte associé” où l’on souhaite dans le document, il faut simplement que les notes de bas pages (rassemblées ou non) soit séparé du reste du texte par un saut de ligne. Personnellement, je préfère rassembler les notes d’une section à la fin de la section en question. Par exemple :

Un bout de texte avec une note[^note1] et une autre [^note2].  
  
[^note1]: à la fin d'une section par exemple.  
[^note2]: ou encore, à la fin du document.

Un bout de texte avec une note[[16]](#footnote-119) et une autre[[17]](#footnote-120).

Notez qu’il faut séparer le texte des notes de bas de pages du texte principal d’un saut de ligne.

### Références à une section

La référence à une section se fait à l’aide de deux crochets. Dans le premier crochet, on trouve le texte associé au lien et dans le second, le nom de la partie à laquelle on fait référence :

Référence à la [section sur les liens hypertextes][Liens hypertextes]

Référence à la [section sur les liens hypertextes](#liens-hypertextes)

On peut également utiliser le nom de la section dans le premier crochet et rien dans le second :

Rappelez vous la section [Liens hypertextes][]

Rappelez vous la section [Liens hypertextes](#liens-hypertextes)

### Références bibliographiques

Un des points forts de Pandoc est la possibilité de gérer de manière très efficace votre bibliographie. Un grand nombre de fichiers de bibliographie sont bien gérés[[18]](#footnote-123), dont les fichiers bibtex. Pour plus de renseignements, visitez la [page du site de R Markdown consacrée](http://rmarkdown.rstudio.com/authoring_bibliographies_and_citations.html).

L’exemple qui suit est basé sur le fichier bibtex (mybiblio.bib) disponible dans le dossier “aux” qui est donnée avec ce document et que je donne ci-dessous :

@article{Lande1979,  
 author = {Lande, R},  
 journal = {Evolution (N. Y).},  
 number = {1},  
 pages = {402--416},  
 title = {Quantitative Genetic Analysis of Multivariate Evolution,  
 Applied to Brain : Body Size Allometry Russell Lande},  
 volume = {33},  
 year = {1979}  
}  
@article{Oreskes1994,  
 author = {Oreskes, Naomi and Shrader-Frechette, Kristin and Belitz, Kenneth},  
 journal = {Science (80-. ).},  
 number = {5147},  
 pages = {641--646},  
 title = {Verification, Validation, and Confirmation of Numerical  
 Models in the Earth Sciences},  
 volume = {263},  
 year = {1994}  
}  
@article{Knauff2014,  
 author = {Knauff, Markus and Nejasmic, Jelica},  
 issn = {1932-6203},  
 journal = {PLoS One},  
 number = {12},  
 pages = {e115069},  
 title = {{An Efficiency Comparison of Document Preparation Systems  
 Used in Academic Research and Development}},  
 volume = {9},  
 year = {2014}  
}

Ce fichier est spécifié dans le YAML : “bibliography: mybiblio.bib” (voir la section “[Le fichier YAML][]”). Le style de citation peut être spécifié, entre autres, à l’aide d’un fichier CSL, lui aussi spécifié dans le YAML et fourni dans le dossier “aux”  : “csl: ./aux/journal-of-theoretical-biology.csl”. Dans le texte, on peut appeler une référence en utilisant la forme sans parenthèse ou avec parenthèse. Pour la forme sans parenthèse, il suffit d’utiliser “@id” où “id” est l’identifiant donné par le fichier “mybiblio.bib”. Pour une référence sans parenthèse :

Selon @Oreskes1994, la modélisation [...].

1. Selon Oreskes et al. (1994), la modélisation […].

Pour les citations entre parenthèses, on procède ainsi :

Dans la littérature, [...] [@Oreskes1994; @Lande1979; @Knauff2014].

1. Dans la littérature, […] (Knauff and Nejasmic, 2014; Lande, 1979; Oreskes et al., 1994).

Il est important de savoir que la liste des références est mise à la fin du document, nous ajoutons donc #Références (ou # bibliographie) à la fin du document pour lui donner un titre. Une autre remarque pour vous dire qu’il existe de nombreux fichiers “csl” (acronyme pour [*Citation Style Langage*](https://en.wikipedia.org/wiki/Citation_Style_Language)) sur le [site de Zotero](https://www.zotero.org/styles).

# Intégration de R dans le document

L’intérêt du package rmarkdown est d’étendre la syntaxe Pandoc Markdown avec les fonctionnalités du [package knitr](http://cran.r-project.org/web/packages/knitr/index.html) pour insérer du code R ainsi que les sorties associées (sorties console et figures). Nous obtenons ainsi un document dynamique en ce sens que si les données associées et/ou le code R changent, le document évolue aussi.

## Les morceaux de code (*code chunks*)

### Deux injections de code possible

Il y a deux types de morceaux de code. En premier lieu, ceux qui sont insérés directement dans le texte (“inline”). Pour cela, il faut inclure le texte sous la forme : `r expression`. Je peux, par exemple, demander l’heure et la date à R en utilisant la function *Sys.time()* : 2020-06-09 14:03:35.

Ensuite, il y a le code que nous souhaitons séparer du reste du texte et dont on souhaite obtenir le résulat. Les morceaux de code sont alors écrits :

```{r + spécifications}  
ligne de code 1  
ligne de code 2  
...  
ligne de code n  
```

Les commentaires sont introduits, comme dans R, sous la forme de lignes de codes commençant par un #. Débutons avec un exemple simple qui inclut un commentaire et une addition :

```{r addition}  
# une addition avec R.  
2+3  
```

Notez que “addition” qui suit “r” dans l’accolade est l’identifiant du morceau de code. Les lignes précédentes me donnent :

# une addition avec R.  
2+3

## [1] 5

J’obtiens ainsi le code de R dans un environnement adéquate (voir la coloration du code) avec la sortie console associée, en l’occurrence, le résultat de l’addition. Je présente dans la suite un certain nombre d’options qui donne une large flexibilité dans la création des morceaux de code. Pour avoir accès à plus de précisions, reportez-vous au [guide de référence](http://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf) ou au [site de knitr](http://yihui.name/knitr/options/). L’apparence des morceaux de code peut être facilement modifiée dans le YAML, avec le paramètre “highlight” (voir [plus haut][Le fichier YAML]) qui peut être spécifié pour les différents formats.

### Le paramètre “results”

Il est aussi possible de changer l’apparence des sorties console grâce au paramètre “results” dans l’accolade.

* result=‘asis’ pour que les résultats soient affichées avec la police du document texte principal :
* ``{r addition2, results='asis'}  
   # une addition avec R.  
   2+3  
   ```

# une addition avec R.  
2+3

[1] 5

* result=‘hide’ :
* ``{r addition3, results='hide'}  
   # une addition avec R.  
   2+3  
   ```

# Commentaire : une addition avec R.  
2+3

* Il existe aussi les options “results=‘hide’” et “results=‘hold’”. La première n’affiche pas les sorties console de R (sauf les messages spéciaux comme les messages d’erreur). La seconde permet d’afficher l’ensemble des sorties après le morceau de code. Par exemple, sans cette option, en utilisant le code suivant’ :
* ``{r} a <- 2 print(a) b <- 3 print(b) 2+3 ``` j’obtiens :

a <- 2  
print(a)

## [1] 2

b <- 3  
print(b)

## [1] 3

2+3

## [1] 5

alors qu’en ajoutant “results=‘hold’” :

a <- 2  
print(a)  
b <- b  
print(b)  
2+3

## [1] 2  
## [1] 3  
## [1] 5

### Le paramètre “comment”

Les symboles devant les résultats de R peuvent être choisis en utilisant le paramètre “comment”:

``{r addition4, comment=">>", background='#FFFF00'}  
# Commentaire : une addition avec R.  
2+3  
```

# une addition avec R.  
2+3

>> [1] 5

### Les paramètres “eval” et “echo”

Nous pouvons être amené à présenter du code qu’il ne faut pas exécuter, il existe pour cela le paramètre “eval”. Ainsi, avec

``{r, eval=FALSE}  
install.packages(`rmarkdown`)  
```

l’installation du package rmarkdown n’est pas exécutée mais bien affichée :

install.packages(`rmarkdown`)

Inversement, il est possible, grâce au paramètre “echo”, de ne pas afficher un code, mais ce qu’il génère, les lignes suivantes :

``{r, echo=FALSE}  
cat("Mais quel code a été utilisé?")  
```

deviennent :

## Mais quel code a été utilisé?

Encore une fois, je n’essaye pas d’être exhausti. Pour plus de précision, il y a le [guide de référence](http://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf) ou le [site de knitr](http://yihui.name/knitr/options/). Vous y apprendrez, entre autres, qu’il est possible de créer des dépendances entre morceaux de code, ce que je n’aborde pas dans le document.

## Les tableaux depuis R

QAvec le package knitr, il est possible d’intégrer dans le texte une table créée sous R. Pour cela, il suffit de construire un objet R, le plus cohérent est de créer un data.frame avec R (éventuellement à partir d’un fichier importé), par exemple :

var1 <- 20\*runif(15)  
table1 <- data.frame(  
 experience = paste0("exp",rep(1:3, each = 5)),  
 replicat = rep(letters[1:5], 3),  
 var1 = var1,  
 var2 = var1 + rnorm(15)  
 )

La fonction *kable()* du package “knitr” nous permet d’obtenir notre object *table1* en différents formats (dont “latex”, “html”, “markdown”).

knitr::kable(table1, caption="Table créée à partir du data.frame \*table1\*")

Table créée à partir du data.frame *table1*

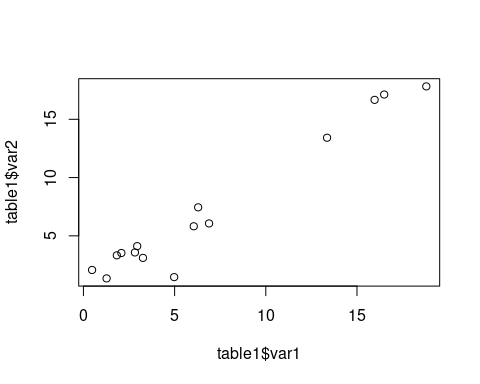
|  |  |  |  |
| --- | --- | --- | --- |
| experience | replicat | var1 | var2 |
| exp1 | a | 18.8016875 | 17.820280 |
| exp1 | b | 15.9658362 | 16.665329 |
| exp1 | c | 1.2706577 | 1.346614 |
| exp1 | d | 6.2911804 | 7.445146 |
| exp1 | e | 0.4748829 | 2.069349 |
| exp2 | a | 2.9483551 | 4.114919 |
| exp2 | b | 1.8315272 | 3.322477 |
| exp2 | c | 6.8889373 | 6.062798 |
| exp2 | d | 16.4920669 | 17.123742 |
| exp2 | e | 4.9753849 | 1.456529 |
| exp3 | a | 2.0806911 | 3.524951 |
| exp3 | b | 2.8248638 | 3.573359 |
| exp3 | c | 13.3593449 | 13.415606 |
| exp3 | d | 6.0489705 | 5.818821 |
| exp3 | e | 3.2647097 | 3.106776 |

Pour en savoir plus, reportez-vous à la documentation de cette fonction. Le package [kableExtra](https://cran.r-project.org/web/packages/kableExtra/vignettes/awesome_table_in_html.html) vaut aussi le coupe d’oeil!

## Les figures produites avec R

Avec le package rmarkdown, il est très facile d’insérer les figures produites avec R dans un document, le plus simple est de regrader ce que nous génère l’appel à un plot simple.

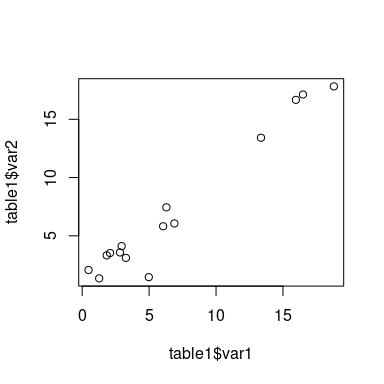
plot(table1$var1,table1$var2)



J’obtiens la figure demandée qui est d’assez grande dimension et numérotée “Figure 3”. Pour formater une figure obtenue avec R, il existe des options pour contrôler, entre autres, sa taille, son alignement et sa légende (taille et alignement ne sont pas supportés pour la sortie Word). Pour visualiser l’ensemble des options disponibles, je vous invite à regarder la page 3 du [guide de référence](http://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf). Je vais changer la taille des figures, l’alignement et ajouter une légende

``{r, fig.cap="une figure avec une légende",  
fig.height=4, fig.width=4, fig.align='left'}  
plot(table1$var1,table1$var2)  
```

plot(table1$var1,table1$var2)



une figure avec une légende

On peut alors profiter des facilités graphiques de R pour créer des figures élégantes. Je peux, par exemple, personnaliser la figure précédente avec quelques lignes supplémentaires.

par(bty="l", font=2)  
plot(table1$var1,table1$var2, pch=15:19, col=rep(c(8,4,6),each=5), xlab="Mon axe des abscisses",  
 ylab="Mon axe des ordonnées")  
 legend("bottomright", letters[1:5], pch=15:19, bty="n")

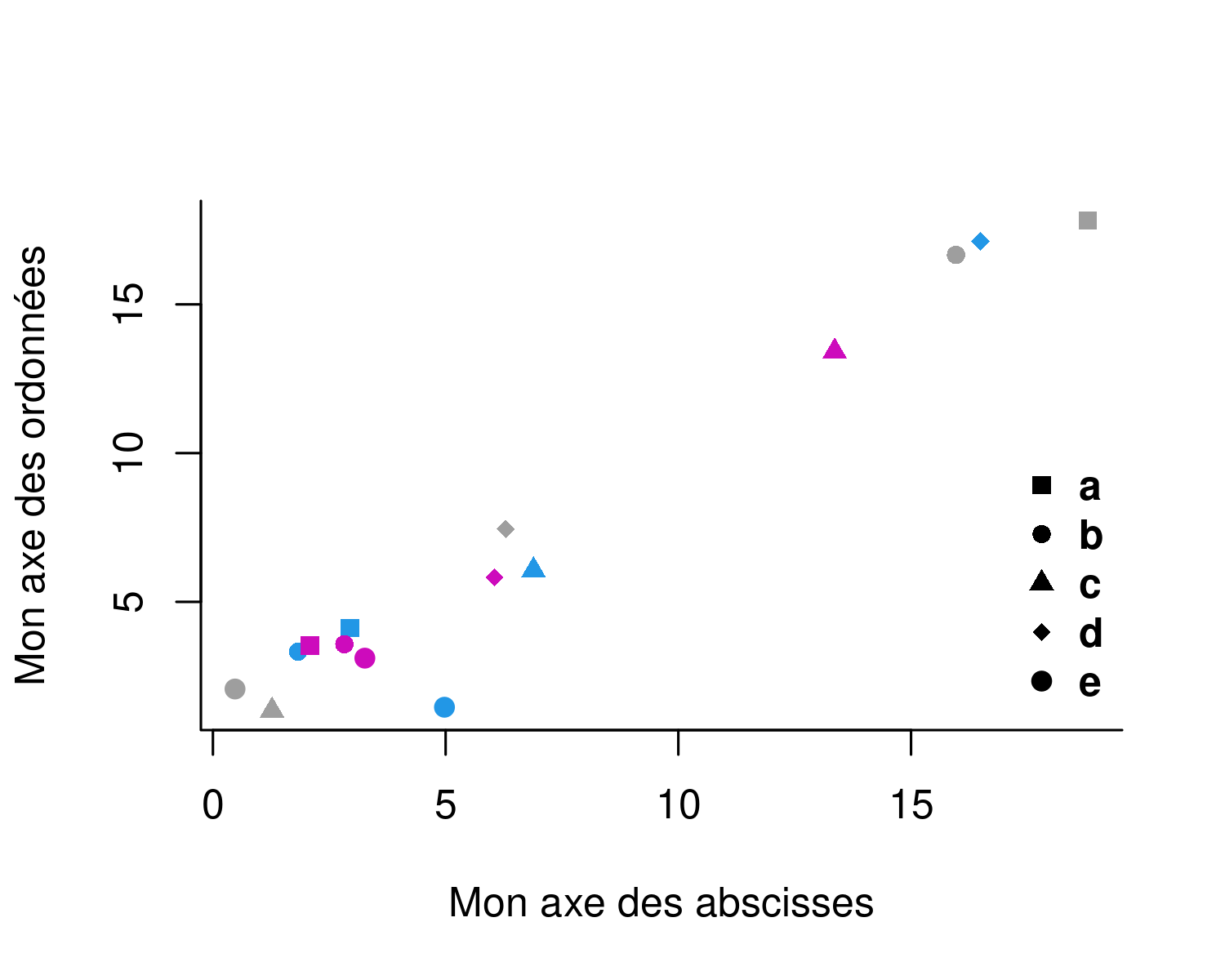
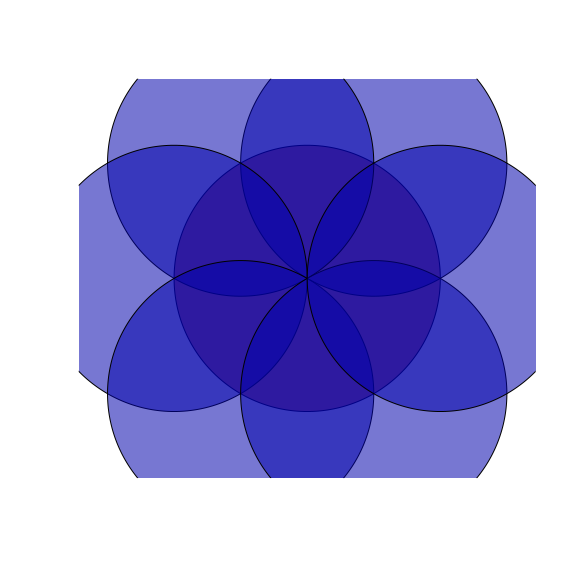


Figure associé à la table 1 […]. Le gris pour l’expérience 1, bleu pour la seconde et violet pour la troisième.

J’ai ajouté à la fin de la légende “\label{figcol}”, je peux donc l’appeler en utilisant “\ref{figcol}” mai ce n’est valable que pour la sortie pdf. De mon point de vue, on peut considérée qu’on a une extension graphique en ligne de commande un peu moins verbeuse que [PGF/TikZ](http://sourceforge.net/projects/pgf/).

kcircle <- function(rad=1,centre=c(0,0),from=0,to=2\*pi,dt=0.001,...){  
 pt <- seq(from, to, dt)  
 polygon(centre[1]+rad\*cos(pt),centre[2]+rad\*sin(pt),...)  
}  
plot(c(-1.5,1.5), c(-1.5,1.5), asp=1, type="n", ann=FALSE,  
 axes=FALSE, xaxs="i", yaxs="i")  
kcircle(col="#AA000088")  
for (i in 1:6) kcircle(1,c(cos(i\*pi/3),sin(i\*pi/3)), col="#0000AA88")



Ma rosace

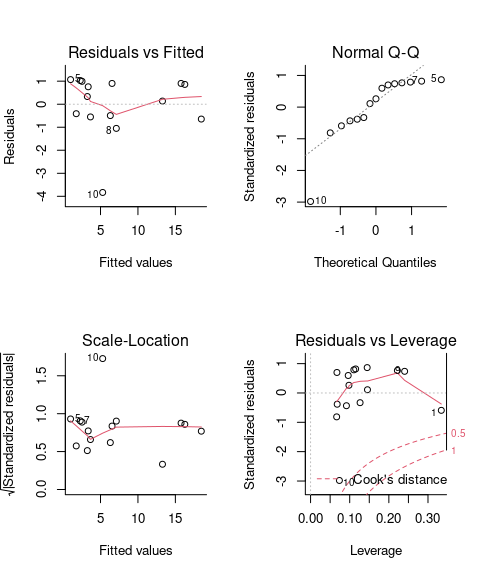
## Une application, modèle linéaire dynamique

Je souhaite faire un modèle linéaire avec la variable explicative *var1* et la variable à expliquer *var2* de notre *table1*. Nous créons alors un petit document sachant que les données évolueront. Je vais faire une inclusion donc utiliser du code R dans mon document.

mod1<-lm(var2~var1, data=table1)  
summary(mod1)

##   
## Call:  
## lm(formula = var2 ~ var1, data = table1)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -3.8326 -0.5228 0.3335 0.9024 1.0689   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 0.54787 0.52848 1.037 0.319   
## var1 0.95295 0.05803 16.422 4.49e-10 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.336 on 13 degrees of freedom  
## Multiple R-squared: 0.954, Adjusted R-squared: 0.9505   
## F-statistic: 269.7 on 1 and 13 DF, p-value: 4.494e-10

par(bty="l", mfrow=c(2,2))  
plot(mod1)



Mon modèle linéaire

# De nombreuses utilisations de R Markdown

WIP !! WIP !! WIP !!

Concluons ce document par une ouverture sur davantage de fonctionnalité du package R Markdown.

* pour les documents :
  + [PDF](https://en.wikipedia.org/wiki/Portable_Document_Format)
  + [HTML](http://rmarkdown.rstudio.com/html_document_format.html)
  + [Word](http://rmarkdown.rstudio.com/word_document_format.html)
  + [Tufte handout (Pdf)](http://rmarkdown.rstudio.com/tufte_handout_format.html)
  + [Package Vignette (Html)](http://rmarkdown.rstudio.com/package_vignette_format.html)
* pour les présentations :
  + [Beamer](http://rmarkdown.rstudio.com/beamer_presentation_format.html), un lien utile: <https://bitbucket.org/rivanvx/beamer/wiki/Home>
  + [ioslide](http://rmarkdown.rstudio.com/ioslides_presentation_format.html), un lien utile: <https://code.google.com/p/io-2012-slides/>
  + [Slidy](http://rmarkdown.rstudio.com/slidy_presentation_format.html), un lien utile <http://www.w3.org/Talks/Tools/Slidy2/#(1)>

# Quelques conseils et astuces

WIP !! WIP !! WIP !!

# Références

Knauff, M., Nejasmic, J., 2014. An Efficiency Comparison of Document Preparation Systems Used in Academic Research and Development. PLoS One 9, e115069.

Lande, R., 1979. Quantitative Genetic Analysis of Multivariate Evolution , Applied to Brain : Body Size Allometry Russell Lande. Evolution (N. Y). 33, 402–416.

Oreskes, N., Shrader-Frechette, K., Belitz, K., 1994. Verification, validation, and confirmation of numerical models in the earth sciences. Science (80-. ). 263, 641–646.

Xie, Y., 2017. Dynamic documents with R and knitr. CRC Press, Boca Raton, Florida.

1. <https://en.wikipedia.org/wiki/John_Gruber>, consulté le 1er juin 2020. [↑](#footnote-ref-25)
2. <https://fr.wikipedia.org/wiki/ReStructuredText>, consulté le 1er juin 2020. [↑](#footnote-ref-28)
3. <https://blog.codinghorror.com/responsible-open-source-code-parenting/>, consulté le 1er juin 2020. [↑](#footnote-ref-30)
4. Par example Goldmark, <https://github.com/yuin/goldmard>, un parser Markdown écrit en Go et utilisé par Hugo (un générateur de site très populaire), est compatible avec Common Mark. [↑](#footnote-ref-38)
5. La version 1 n’utilise pas Pandoc. [↑](#footnote-ref-41)
6. Pandoc est d’ailleurs capable de gérer différentes variantes de Markdown. [↑](#footnote-ref-43)
7. <https://pandoc.org/MANUAL.html#pandocs-markdown>, 1er juin 2020. [↑](#footnote-ref-45)
8. voir <https://pandoc.org/MANUAL.html#templates>, consulté le 1er juin 2020. [↑](#footnote-ref-59)
9. du contenu préliminaire. [↑](#footnote-ref-65)
10. voir ><https://fr.wikibooks.org/wiki/LaTeX/%C3%89crire_des_math%C3%A9matiques>>, consulté le 2 juin 2020. [↑](#footnote-ref-66)
11. Voir la section “Math” à l’URL suivant ><https://pandoc.org/MANUAL.html#pandocs-markdown>>, consulté le 2 juin 2020. [↑](#footnote-ref-68)
12. <https://cran.r-project.org/web/packages/yaml/index.html>, consulté le 8 juin 2020. [↑](#footnote-ref-75)
13. Le site de documentation est totalement écrit en Markdown et reprend ce que nous pouvons lire sur le [site de Pandoc](http://pandoc.org/demo/example9/pandocs-markdown.html) et que je reprend mais en français! [↑](#footnote-ref-86)
14. Voir la section *Fenced code blocks* du manuel de référence <https://pandoc.org/MANUAL.html>. [↑](#footnote-ref-96)
15. voir <https://tex.stackexchange.com/questions/111868/pandoc-how-can-i-get-numbered-latex-equations-to-show-up-in-both-pdf-and-html-o>) [↑](#footnote-ref-109)
16. la première [↑](#footnote-ref-119)
17. la seconde [↑](#footnote-ref-120)
18. Les logiciels de gestion de bibliographie génèrent ces fichiers. [↑](#footnote-ref-123)