Utiliser R Markdown pour créer des documents dynamiques

par Kevin Cazelles

23 December 2021

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 (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>). Bien que le contenu est accessible aux utilisateurs débutants, 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 2.11 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 4.1.2 (2021-11-01)  
## Platform: x86\_64-pc-linux-gnu (64-bit)  
## Running under: Ubuntu 20.04.3 LTS  
##   
## Matrix products: default  
## BLAS: /usr/lib/x86\_64-linux-gnu/blas/libblas.so.3.9.0  
## LAPACK: /usr/lib/x86\_64-linux-gnu/lapack/liblapack.so.3.9.0  
##   
## locale:  
## [1] LC\_CTYPE=C.UTF-8 LC\_NUMERIC=C   
## [3] LC\_TIME=C.UTF-8 LC\_COLLATE=C.UTF-8   
## [5] LC\_MONETARY=C.UTF-8 LC\_MESSAGES=C.UTF-8   
## [7] LC\_PAPER=C.UTF-8 LC\_NAME=C.UTF-8   
## [9] LC\_ADDRESS=C.UTF-8 LC\_TELEPHONE=C.UTF-8   
## [11] LC\_MEASUREMENT=C.UTF-8 LC\_IDENTIFICATION=C.UTF-8  
##   
## attached base packages:  
## [1] stats graphics grDevices utils datasets methods base   
##   
## other attached packages:  
## [1] knitr\_1.37 yaml\_2.2.1 JuliaCall\_0.17.4  
##   
## loaded via a namespace (and not attached):  
## [1] Rcpp\_1.0.7 here\_1.0.1 lattice\_0.20-45 png\_0.1-7   
## [5] rprojroot\_2.0.2 digest\_0.6.29 rappdirs\_0.3.3 grid\_4.1.2   
## [9] jsonlite\_1.7.2 magrittr\_2.0.1 evaluate\_0.14 highr\_0.9   
## [13] rlang\_0.4.12 stringi\_1.7.6 jquerylib\_0.1.4 Matrix\_1.3-4   
## [17] reticulate\_1.22 rmarkdown\_2.11 tools\_4.1.2 stringr\_1.4.0   
## [21] xfun\_0.29 fastmap\_1.1.0 compiler\_4.1.2 htmltools\_0.5.2

rmarkdown::pandoc\_version()

## [1] '2.14.2'

# 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’autres ainsi que d’autres langages et outils pour comprendre ce package, en commençant par Markdown.

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-24), programmeur, blogueur 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 mêmes objectifs, par exemple ReStructuredText[[2]](#footnote-27).

Après la publication de Markdown, John Gruber a cessé de travailler sur Markdown[[3]](#footnote-29) 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. 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-37), 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-40) utilise [la variante Markdown de Pandoc](https://rmarkdown.rstudio.com/lesson-8.html)[[6]](#footnote-42). 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 pensée pour rester fidèle à l’esprit originel de Markdown tout en incluant davantage d’éléments de document communs différent format de documents[[7]](#footnote-44). Pour rendre le document dynamique, R Markdown utilise les fonctionnalités de [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 pour ceux qui ne pratiquent pas la langue de Shakespeare, cette documentation est presque exclusivement écrite en anglais. Pour les lecteurs capables de lire l’anglais, je recommande [le site officiel de R Markdown](https://rmarkdown.rstudio.com) sur lequel vous trouverez, entre autres, un condensé d’utilisation sous forme de *Cheat Sheet*, voir <http://www.rstudio.com/wp-content/uploads/2015/02/rmarkdown-cheatsheet.pdf>. Aussi, sur le site <https://bookdown.org> vous trouverez un livre en anglais très complet sur le sujet, par l’un des architectes de R Markdown, Yihui Xie, <https://bookdown.org/yihui/rmarkdown/> ainsi qu’un [livre d’astuces (*cook book*)](https://bookdown.org/yihui/rmarkdown-cookbook/).

Une section complète du site de l’auteur principal de knitr, Yihui Xie, lui encore, <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>.

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’astuces 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-58), 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.

Avant de rentrer dans le vif du sujet, je souhaiterais insister sur le fait que cette introduction est axée sur la création de document en format Word, PDF et HTML pour de nombreux usages (rapport, article, etc.), mais que les possibilités offertes par rmarkdown sont plus vastes (voir le site dédié <http://rmarkdown.rstudio.com>), entre autres

* 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)
  + [ioslide](http://rmarkdown.rstudio.com/ioslides_presentation_format.html)
  + [Slidy](http://rmarkdown.rstudio.com/slidy_presentation_format.html)

De plus, il existe des packages qui se basent sur rmarkdown et en étendent les applications. Par exemple, blogdown vous permet d’écrire des postes de blogue avec R Markdown dans un site permet généré par Hugo (<https://gohugo.io/>). De même, avec bookdown (<https://bookdown.org/yihui/bookdown/>), vous pouvez créer de long documents HTML structurés comme un [Gitbook](https://www.gitbook.com). Ainsi, les fonctionnalités décrites dans ce document permettent d’utiliser une très grande variété de médias pour partager les fruits de votre labeur.

# 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 utilisés 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-74) 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 dans un tel fichier car différents langages y sont utilisés, 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.

Ajoutons à cela qu’avec la variante syntaxique Pandoc de Markdown, les symboles mathématiques TeX[[10]](#footnote-75) (<https://pandoc.org/MANUAL.html#math>) sont utilisés pour rendre facile l’écriture, entre autres, des équations[[11]](#footnote-78), 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 fait avec n’importe quelle éditeur de texte, ou de code, ou même en ligne de commande. Par exemple, 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 exemples 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-85) 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 non 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 chaîne de caractère, une date, des chiffres ou encore une liste:

nomduchamp: [élément1, élément2]

ou encore

nomduchamp:  
 - élément1  
 - élément2

Noté que l’indentation est utilisée pour signifier la hiérarchie entre les différents éléments et que les commentaires sont introduits par un “#”.

nomduchamp:  
 souchamp1: élément1  
# ce qui suit sont les élément dans le 2eme souschamp  
 souchamp2:  
 souchamp2a: élément2a  
 souchamp2b:  
 - élément2ba  
 élément2ba

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/assets/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!"  
# Les différents variables  
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("assets/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"

Les chaînes de caractères y, yes, true, TRUE ainsi que n, no, false, FALSE seront toutes interprétées comme variables booléennes (respectivement TRUE ou FALSE dans R). C’est une fonctionnalité pratique mais à double tranchant: par exemple si y est utilisé comme nom de champ, le nom du champ sera interprété comme TRUE et pour palier ce comportement, il faut ajouter des guillemets (voir l’exemple ci-dessus).

Les remarques précédentes sont générales et vous pourriez être amené à utiliser YAML dans un autre contexte que l’édition de document avec R Markdown, par exemple pour stocker des données. Revenons à l’utilisation de YAML dans un document R Markdown. Ce qu’il faut bien garder à l’esprit, c’est que les options utilisées servent à spécifier différents propriétés des documents à générer à partir du fichier .Rmd (p-ex le titre, la date, les polices de caractères utilisées, ajout d’une table des matières, etc.) grâce à des *templates* utilisés par Pandoc. Ainsi, les options disponibles dépendent des templates utilisés qui sont spécifiques à un format donné, les options par défaut varient ainsi d’un format à l’autre. 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). Notez qu’il est possible de créer ses propres templates et donc d’ajouter autant d’options que désiré.

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 trouverez dans le fichier source de ce document) :

---  
title: "Utiliser R Markdown pour créer des documents dynamiques"  
date: "23 December 2021"  
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: assets/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: assets/license.tex  
 word\_document:  
 #toc: yes  
 fig\_caption: yes  
 highlight: pygments  
 md\_document:  
 variant: markdown\_strict  
bibliography: assets/mybiblio.bib  
csl: assets/journal-of-theoretical-biology.csl  
header-includes:  
 - \usepackage{fancyhdr}  
 - \pagestyle{fancy}  
 - \fancyfoot[CO,CE]{Documents dynamiques avec R Markdown}  
 - \fancyfoot[R]{\thepage}  
---

Comme expliqué précédemment, l’indentation est utilisée pour regrouper les options par format. Ainsi, les lignes qui se trouvent entre “pdf\_document:” et “word\_document:”, spécifient différents aspects relatifs à la génération au format PDF. Par exemple, pour ajouter une table des matières, on ajoute toc: yes, “toc” signifiant *table of content* (table des matières) et pour en contrôler la profondeur (le niveau maximal des sous-titres affichés dans la table), on utilise “toc\_depth”. De même, “tof” (*table of figure*) permet d’ajouter une table des figures. Aussi, l’option “number\_section:true” permet d’obtenir une numérotation des différentes parties.

Avant de finir, une astuce qui peut être bien pratique, il est possible d’accéder à la liste des options passée dans l’en-tête YAML comme suit:

rmarkdown::metadata

## $title  
## [1] "Utiliser R Markdown pour créer des documents dynamiques"  
##   
## $date  
## [1] "`r format(Sys.time(), '%d %B %Y')`"  
##   
## $author  
## [1] "par Kevin Cazelles"  
##   
## $lang  
## [1] "fr"  
##   
## $abstract  
## [1] "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."  
##   
## $fontfamily  
## [1] "fourier"  
##   
## $linestretch  
## [1] 1  
##   
## $fontsize  
## [1] "10pt"  
##   
## $lof  
## [1] TRUE  
##   
## $output  
## $output$html\_document  
## $output$html\_document$toc  
## [1] TRUE  
##   
## $output$html\_document$toc\_float  
## [1] TRUE  
##   
## $output$html\_document$highlight  
## [1] "default"  
##   
## $output$html\_document$theme  
## [1] "flatly"  
##   
## $output$html\_document$include  
## $output$html\_document$include$after\_body  
## [1] "assets/footer.html"  
##   
##   
##   
## $output$pdf\_document  
## $output$pdf\_document$highlight  
## [1] "tango"  
##   
## $output$pdf\_document$toc  
## [1] TRUE  
##   
## $output$pdf\_document$toc\_depth  
## [1] 3  
##   
## $output$pdf\_document$fig\_caption  
## [1] TRUE  
##   
## $output$pdf\_document$keep\_tex  
## [1] TRUE  
##   
## $output$pdf\_document$latex\_engine  
## [1] "pdflatex"  
##   
## $output$pdf\_document$number\_section  
## [1] TRUE  
##   
## $output$pdf\_document$includes  
## $output$pdf\_document$includes$before\_body  
## [1] "assets/license.tex"  
##   
##   
##   
## $output$word\_document  
## $output$word\_document$fig\_caption  
## [1] TRUE  
##   
## $output$word\_document$highlight  
## [1] "pygments"  
##   
##   
## $output$md\_document  
## $output$md\_document$variant  
## [1] "markdown\_strict"  
##   
##   
##   
## $bibliography  
## [1] "assets/mybiblio.bib"  
##   
## $csl  
## [1] "assets/journal-of-theoretical-biology.csl"  
##   
## $`header-includes`  
## [1] "\\usepackage{fancyhdr}"   
## [2] "\\pagestyle{fancy}"   
## [3] "\\fancyfoot[CO,CE]{Documents dynamiques avec R Markdown}"  
## [4] "\\fancyfoot[R]{\\thepage}"

## Éditer le contenu du fichier R Markdown

Pour éditer le contenu du fichier R Markdown, on utilise simplement du texte plein avec la syntaxe Pandoc Markdown (détaillée dans la section suivante), la différence avec un document Markdown est qu’il est possible d’exécuter du code R et d’utiliser les résultat du code, ce qui rend le document dynamique (voir la section dédiée plus bas).

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

Une fois que le fichier est prêt (ou lorsqu’on souhaite voir un aperçu de notre document) 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 de menu contextuelle associée au fichier (voir la documentation associée, ?render). Dans la suite du document, la fonction render() est utilisée:

render("ex\_Rmardown.rmd", output\_format = "all")

Passer “all” au paramètre output\_format permet d’obtenir tous les fichiers pour lesquels il existe une spécification dans le fichier YAML. Les formats utilisés ici sont :

* [PDF](https://bookdown.org/yihui/rmarkdown/pdf-document.html)
* [HTML](https://bookdown.org/yihui/rmarkdown/html-document.html)
* [Word](https://bookdown.org/yihui/rmarkdown/word-document.htmll)
* [Markdown](https://bookdown.org/yihui/rmarkdown/markdown-document.html)

Les documents au format PDF requièrent [l’installation de Latex](https://www.tug.org/texlive/quickinstall.html) étant donné que 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). Au passage, le format “Tufte handout” (qui n’est pas utilisé ici) est une mise en page qui inclut des marges larges où sont insérées les illustrations (dont les figures) et que nous devons au chercheur Edward Tufte.

# Le variante Pandoc de Markdown

Dans cette partie, je présente les éléments de formatage du texte proposés par la syntaxe Pandoc de Markdown. L’ensemble est très bien présenté sur le [site de R Markdown](http://rmarkdown.rstudio.com/authoring_pandoc_markdown.html)[[13]](#footnote-99) 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. Parmi les nombreux tutoriels plus courts sur le sujet, je recommande celui de Sarah Simpkin (Simpkin, 2020) disponible en ligne à l’adresse suivante <https://programminghistorian.org/fr/lecons/debuter-avec-markdown>.

Avant de commencer, je tiens à préciser 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 l’antislash pour Markdown : “\”. 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 :

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

* 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~

Notez qu’il n’y a pas de balises pour le soulignement du texte. De manière générale, quand un élément de mise en page manque dans la syntaxe, il est toujours possible d’utiliser des commandes d’un langage. Par exemple, pour souligner dans un document qui sera produit en HTML, je peux utiliser <u>texte souligné</u> mais cela ne me permettra pas d’avoir un texte souligné en Word ou en PDF. De même que si j’utilise \underline{texte souligné} le texte sera souligné en PDF, mais pas en HTML ni en Word. Procéder de la sorte n’est pas toujours souhaité car le document R Markdown perd en généralité, en ce sens où il ne pourra pas être correctement généré dans tous les formats. Cela n’est cependant pas nécessairement un problème, par exemple, si vous souhaitez obtenir le document en un seul format, ce fonctionnement devient un atout puisque vous pouvez utiliser toute la gamme de mise en forme offerte par le langage en question.

## Les titres

Le plus simple est d’utiliser un nombre croissant de # (*ATX-Style headers* en anglais) 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 (*SETEXT-Style headers* en anglais). Cette option permet 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 aussi bien en Latex qu’en HTML. Notez que 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.

Et si j’utilise un espace entre les éléments de la liste alors le rendu change un peu, la liste est plus aérée, par exemple en HTML, une balise paragraphe, <p> </p> est ajoutée, ainsi

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

devient:

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

En utilisant une indentation de 4 espaces (ou une tabulation), il est possible de créer 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)

### Listes 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 de manière ordonnée, cela ne changera pas le résultat. Néanmoins, le premier nombre détermine le premier numéros de la liste, ainsi en utilisant :

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

## Sauts de ligne

En ajoutant une ligne (ou plus) vide entre deux paragraphes, les paragraphes sont correctement formatés[[14]](#footnote-108). Ainsi, avec les lignes suivantes :

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam.  
  
Quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

on obtient :

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam.

Quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Avec un simple retour à la ligne, aucun saut de ligne n’est inséré, les paragraphes sont mis bout à bout.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam.  
Quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam. Quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Il est cependant possible d’ajouter un retour à la ligne en utilisant un retour à la ligne et tabulation à la fin du premier paragraphe.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam.   
Quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam.  
Quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

La seule différence avec l’exemple précédent est l’ajout d’une tabulation après “veniam.”.

## 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.

Il est possible d’imbriquer une citation dans une citation en utilisant >> :

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

ce qui donne :

La citation de départ

une hiérarchie dans la citation

## Blocs de code

Pour l’ensemble des exemples qui suivent, 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[[15]](#footnote-111). 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, il est possible d’utiliser des blocs 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 est 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, ce que nous verrons dans la suite.

## Les barres horizontales

Une ligne barre horizontale peut être ajoutée comme suit :

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

ou encore :

\* \* \* \* \*

## 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 mêmes que celles proposées par [Latex](http://latex-project.org/intro.html) et qui seront utilisées par [MathJax](http://www.mathjax.org) (par défaut) pour générer 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 à l’article de Wikipedia sur le sujet <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)”[[16]](#footnote-120). Les références seront gérées correctement, mais le rendu visuel n’est pas nécessairement celui désiré. Ils existent différentes solutions pour palier ceci. Pour les documents HTML, il est possible d’utiliser le mécanisme de rendu HTML de bookdown (voir <https://bookdown.org/yihui/bookdown/markdown-extensions-by-bookdown.html>), ce qui revient à utiliser un autre type de format de sortie, mais qui reste du HTML. Sinon, il est possible d’utiliser le filtre [pandoc-eqnos](https://github.com/tomduck/pandoc-eqnos) (noter qu’il y a des filtres équivalents pour les tables et les figures). Enfin, si vous voulez simplement un PDF, vous pouvez travailler comme dans un document LaTeX (voir <https://www.overleaf.com/learn/latex/Cross%20referencing%20sections,%20equations%20and%20floats>).

## 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]

Pour faire des références aux figures, comme pour les équations, vous pouvez utiliser un filtre dédié, [pandoc-fignos](https://github.com/tomduck/pandoc-fignos), ou, si vous souhaitez simplement avoir les références dans un document PDF’ vous pouvez peut injecter une balise \\label{lelabel} et utiliser \\ref{lelabel}. Notez que R Markdown nous permet d’inclure aisément des graphiques produits avec R et même n’importe quelle image (voir plus bas).

## 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). Pour les références, vos avec les mêmes options que pour les figures et les équations, le filtre dédié étant [pandoc-tablenos](https://github.com/tomduck/pandoc-tablenos). Notez qu’avec R Markdown, vous pouvez facilement faire des tables à partir de data frames. Je reviens sur là-dessus plus bas.

## 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[[17]](#footnote-136) et une autre[[18]](#footnote-137).

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. Le premier crochet inclut le texte associé au lien et le second, le nom de la partie à laquelle le texte renvoie :

Référence à la [section sur les tables][Les tables]

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

On peut également utiliser le nom de la section dans le premier crochet (s’il est suffisant) 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. Pour les versions antérieures à Pandoc 2.11, les citation étaient gérées par le processeur de citation [pandoc-citeproc](https://github.com/jgm/pandoc-citeproc) qui a été depuis intégrée dans Pandoc. Un grand nombre de fichiers de bibliographie sont bien gérés[[19]](#footnote-141), dont le format bibtex .bib qui est utilisé pour ce document. Pour plus de renseignements, visitez la [page du manuel de Pandoc](https://pandoc.org/MANUAL.html#citations) et la [section du livre *R markdown cookbook*](https://bookdown.org/yihui/rmarkdown-cookbook/bibliography.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: ./assets/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 :

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

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

Pour les citations entre parenthèses, les citations sont insérées entre crochets et séparées par des points-virgules :

2. Dans la littérature [...] [@Oreskes1994] bien que [...] [@Lande1979; @Knauff2014].

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

Pour ajouter du texte dans la parenthèse, il suffit d’ajouter le texte dans le crochet.

3. Dans la littérature, [...] [voir @Oreskes1994] bien que [...] [@Lande1979 entre autres].

1. Dans la littérature, […] (voir Oreskes et al., 1994) bien que […] (Lande, 1979 entre autres).

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

Le package rmarkdown combine la syntaxe Pandoc Markdown avec les fonctionnalités du [package knitr](http://cran.r-project.org/web/packages/knitr/index.html) pour insérer non seulement du code R mais aussi 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. Cela permet, entre autres, de créer des rapports automatisés.

## Les blocs de code R

Il y a deux manières d’insérer des sorties R dans le document:

1. directement dans le texte (*inline* en anglais);
2. en utilisant un bloc de code dédié.

Pour inclure une sortie texte directement dans un paragraphe, on utilise  : `r expression`. Ainsi, par exemple, il est possible d’insérer l’heure et la date au moment de la compilation du document en utilisant la fonction *Sys.time()*, ainsi, `r Sys.time()`  nous donne 2021-12-23 15:00:41.

Le reste de cette section se concentre sur les blocs de code R (*code chunks* en anglais) qui s’utilisent comme les [blocs de code](#blocs-de-code) détaillés précédemment, mais, à la différence de ces-derniers, il est possible d’identifier le bloc de code (idbloc ci-dessous) et varier différents paramètres d’affichage du code source et des sorties (l’ensemble des paramètres sont disponibles à l’URL suivante <https://yihui.org/knitr/options/>). Typiquement, l’utilisation d’un tel bloc de code ressemble à ceci :

```{r, idbloc, param1 = val1, param2 = val2}  
ligne de code 1  
ligne de code 2  
[...]  
ligne de code n  
```

Notons que la casse n’est pas important pour le R de l’accolade et qu’il n’est pas obligatoire d’identifier les blocs de code:

```{R, param1 = val1, param2 = val2}  
ligne de code 1  
ligne de code 2  
...  
ligne de code n  
```

Dans ce cas, une identification par défaut sera ajoutée (unnamed-chunk-i pour le ième bloc non identifié). Personnellement, je recommande vivement d’identifier les blocs de code R, cela permet d’identifier les potentiels problèmes. Aussi, si deux blocs de code R ont le même identifiant,un message d’erreur sera retourné avant que la compilation ne débute.

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 (la virgule entre les deux premiers éléments est facultative), qui permet notamment de facilement identifier 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 donnent une large flexibilité dans la création des morceaux de code. Pour avoir accès à plus de précisions, reportez-vous aux ouvrages de référence (Xie et al., 2020, 2019; Xie, 2014) ou aux pages internet asssociées.

## Modifier l’affichage du code source et des sorties associées

Grâce aux différents paramètres des blocs de code, il est possible de sélectionner finement ce qui est affiché, aussi bien pour le code que pour les sorties associées. Cela permet à l’utilisateur de mettre en avant certaines parties du code, mais aussi de choisir finement les sorties R (figures, tables, etc) à ajouter dans le document.

Dans cette section, j’ai choisi de détailler quelques paramètres clés, mais il y en a d’autres (voir <https://yihui.org/knitr/hooks/>) qui permettent d’aller plus loin. Il est cependant important de noter que tout n’est pas nécessairement faisable avec ces paramètres. Ainsi, il est possible de sélectionner différentes coloration syntaxiques avec le paramètre “highlight” dans [l’en-tête YAML](#X5408e78486463c18291aa7ba9e4e59516c78e04)). Aussi, pour changer la police de font des blocs de code dans un document HTML, il faudra modifier certaines propriétés CSS du document.

### Modifier l’affichage du code source avec le paramètre echo

Prenons le bloc de code suivant (volontairement non identifié).

```{r}  
# une addition de variables avec R  
a <- 2  
b <- 3  
a + b  
```

# une addition de variables avec R  
a <- 2  
b <- 3  
a + b

## [1] 5

Pour enlever le code (et uniquement le code), il suffit d’utiliser echo = FALSE, (echo = TRUE par défaut). Ainsi, le bloc de code sanscode

```{r}  
# une addition de variables avec R  
a <- 2  
b <- 3  
a + b  
```

donne:

## [1] 5

Le paramètre echo peut aussi être utilisé pour sélectionner un sous-ensemble de ligne à afficher. Cela ce fait en utilisant un vecteur indiquant les positions des lignes à afficher. Par exemple, pour montrer uniquement les lignes 1 et 4 on utilise echo de la sorte

```{r}  
# une addition de variables avec R  
a <- 2  
b <- 3  
a + b  
```

ce qui donne:

# une addition de variables avec R  
a + b

## [1] 5

Il est aussi possible de sélectionner les lignes à ne pas montrer en utilisant des valeurs négatives. Ainsi, je peux enlever les trois premières lignes du bloc de code:

```{r}  
# une addition de variables avec R  
a <- 2  
b <- 3  
a + b  
```

ce qui donne:

a + b

## [1] 5

### Modifier l’affichage des sorties

#### Le paramètre comment

Les symboles devant les sorties console de R peuvent être modifiées en utilisant le paramètre comment. Avec le bloc comment1

```{r}  
# une addition de variables avec R  
a <- 2  
b <- 3  
a + b  
```

on obtient

# une addition de variables avec R  
a <- 2  
b <- 3  
a + b

$ [1] 5

De même, comment2  :

```{r}  
# une addition de variables avec R  
a <- 2  
b <- 3  
a + b  
```

il nous donne:

# une addition de variables avec R  
a <- 2  
b <- 3  
a + b

R>> [1] 5

#### Le paramètre results

Le paramètre results permet de choisir comment les sorties d’un bloc de code R vont être traitées. Par défaut, c’est le mode “markup” qui est utilis&nbsp;:

```{r}  
# une division avec R  
cat("Exemple de division avec R: 1/998.001 = ", 1/998.001)  
```

# une division avec R  
cat("Exemple de division avec R: 1/998.001 = ", 1/998.001)

## Exemple de division avec R: 1/998.001 = 0.001002003

* Avec result='asis', les résultats sont affichées comme un paragraphe du document texte principal :
* ```{r}  
   # une division avec R  
   cat("Exemple de division avec R: 1/998.001 = ", 1/998.001)  
   ```

# une division avec R  
cat("Exemple de division avec R: 1/998.001 = ", 1/998.001)

Exemple de division avec R: 1/998.001 = 0.001002003

* Avec result='hide', les sorties console ne sont pas ajoutées :
* ```{r}  
   # une division avec R  
   cat("Exemple de division avec R: 1/998.001 = ", 1/998.001)  
   ```

# une division avec R  
cat("Exemple de division avec R: 1/998.001 = ", 1/998.001)

* results='hold' permet d’afficher *toutes* les sorties après le morceau de code. Sans cette option, les sorties sont ajoutées au fur et à mesure de l’exécution du code et donc le bloc est interrompu. Par exemple, avec le code suivant’ :
* a <- 2  
  print(a)  
  b <- 3  
  print(b)  
  2 + 3
* nous obtenons :

a <- 2  
print(a)

## [1] 2

b <- 3  
print(b)

## [1] 3

2 + 3

## [1] 5

alors qu’avec results='hold',

```{r}  
a <- 2  
print(a)  
b <- 3  
print(b)  
2 + 3  
```

nous avons :

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

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

### Choisir les messages retournés

Il y a trois types de messages retournés par R:

1. les messages : une simple indication (voir la fonction message());
2. les avertissements qui soulignent que quelque chose est peut-être problématiques, mais le code est exécutable (voir la fonction warning());
3. les erreurs qui indiquent que quelques chose ne marche pas dans l’exécution du code (voir stop()).

Considérons le bloc suivant qui inclut un message, un avertissement, une erreur (en commentaire) et une addition.

```{r}  
message("Ceci est un message")  
warning("Ceci est un avertissement")  
# stop("Ceci est un problème")  
2 + 3  
```

Voila ce qu’il nous donne

message("Ceci est un message")

## Ceci est un message

warning("Ceci est un avertissement")

## Warning: Ceci est un avertissement

# stop("PCeci est un problème")  
2 + 3

## [1] 5

il est possible de supprimer le message en utilisant message = FALSE.

```{r}  
message("Ceci est un message")  
warning("Ceci est un avertissement")  
# stop("Ceci est un problème")  
2 + 3  
```

message("Ceci est un message")  
warning("Ceci est un avertissement")

## Warning: Ceci est un avertissement

# stop("Ceci est un problème")  
2 + 3

## [1] 5

De la même manière il est possible de supprimer l’avertissement avec warning = FALSE.

```{r}  
message("Ceci est un message")  
warning("Ceci est un avertissement")  
# stop("Ceci est un problème")  
2 + 3  
```

ce qui donne

message("Ceci est un message")

## Ceci est un message

warning("Ceci est un avertissement")  
# stop("Ceci est un problème")  
2 + 3

## [1] 5

Notez qu’en utilisant results = 'hide', les avertissements seront tout de même ajoutés et cette remarque est valable pour les messages.

```{r}  
message("Ceci est un message")  
warning("Ceci est un avertissement")  
# stop("Ceci est un problème")  
2 + 3  
```

ce qui donne

message("Ceci est un message")

## Ceci est un message

warning("Ceci est un avertissement")  
# stop("Ceci est un problème")  
2 + 3

Enfin, par défaut, si une erreur advient dans votre code, le document ne sera pas généré (c’est d’ailleurs la raison pour laquelle la ligne stop("Ceci est un problème") est commentée précédemment). Il est cependant parfois souhaitable de montrer une erreur (à des fins pédagogiques, par exemple). Pour ce faire, il faut utiliser error = TRUE:

```{r}  
message("Ceci est un message")  
warning("Ceci est un avertissement")  
stop("Ceci est un problème")  
2 + 3  
```

ce qui donne

message("Ceci est un message")

## Ceci est un message

warning("Ceci est un avertissement")

## Warning: Ceci est un avertissement

stop("Ceci est un problème")

## Error in eval(expr, envir, enclos): Ceci est un problème

2 + 3

## [1] 5

### Modifier le mode d’evaluation du code

Par défaut, avec rmarkdown, dans un bloc de code R, le code est exécuté par R et les sorties sont ajoutées dans le document. Ci-dessus, nous avons vu comment modifier ce qui est présenté dans le code source et changer certain aspect de la sortie. Il est également possible de modifier l’exécution du code.

#### Enlever toutes sorties

Dans certains cas, il peut s’avérer utile de montrer le code source sans l’exécuter. Pour empêcher l’évaluation du code, on utilise eval = FALSE. Ainsi, avec

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

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

install.packages(`rmarkdown`)

Il est aussi possible d’utiliser un [blocs de code](#blocs-de-code) sans accolade

```r  
install.packages(`rmarkdown`)  
```

install.packages(`rmarkdown`)

Utiliser eval = FALSE permet de suspendre l’évaluation d’un bloc de code tout en conservant les autres paramètres dans l’accolade, ce qui est très pratique dans des phases de développement de contenu qui prenne du temps à exécuter. Une autre stratégie possible dans ce genre de situation consiste à utiliser le cache (voir plus bas).

#### Exécuter le code silencieusement

R Markdown nous permet d’imbriquer du code dans un document dynamique et il est fréquent que la sortie importe plus que le code et pour obtenir une sortie, parfois des blocs de code intermédiaires doivent être exécutés. Dans de telles situation, ces blocs doivent pouvoir être exécutée de manière silencieuse: ni le code, ni les messages, ni les sorties ne doivent être affichés. C’est typiquement le cas pour les scripts et fonctions intermédiaires à charger pour être utiliser ultérieurement. D’après ce qui a été vu précédemment, nous pourrions combiner les options echo = FALSE, results = hide, message = FALSE et warning = FALSE ensemble dans l’accolade. Il existe cependant une manière plus rapide d’arriver à ce résultat en utilisant un seul paramètre, include = FALSE, qui exécute le code silencieusement. Ci-dessous je créé une fonction qui fait une addition simple et j’assigne le résultat à la variable res

{r include, include = FALSE} mon\_addition <- function(a, b) { return(a + b) } res <- mon\_addition(2, 3)

Ci-dessous j’utilise ce bloc de code …

… et, comme attendu, ni code ni sortie ne sont retournés, mais pourtant le code a bien été exécuté, et donc si je demande l’affichage de res

print(res)

## [1] 5

j’obtiens bien 5 et je peux également utiliser ma fonction mon\_addition() :

mon\_addition(12, 30)

## [1] 42

## Tables et figures

Les remarques de la section précédente sont valides pour tout type de sortie. Les tables et les figures créées avec R sont des sorties très souvent utilisées que nous pouvons personnaliser au besoin.

### Tables

Avec le package knitr, il est possible d’intégrer dans le texte une table créée sous R. Commençons par créer un objet data.frame :

var1 <- 20 \* runif(12)  
tab1 <- data.frame(  
 experience = paste0("traitement\_", rep(1:3, each = 4)),  
 replicat = rep(letters[1:4], 3),  
 var1 = var1,  
 var2 = var1 + rnorm(12)  
 )

La fonction *kable()* du package knitr nous permet d’obtenir notre object *tab1* en différents formats (dont “latex”, “html”, “markdown”). Ici, en utilisant “markdown” par défaut, nous obtenons une table qui sera convertie dans le format désiré par Pandoc. De plus, la fonction *kable()* inclut le paramètre caption pour ajouter une légende.

```{r table1}  
library(knitr)  
kable(tab1, caption = "Table créée à partir de \*df1\*")  
```

library(knitr)  
kable(tab1, caption = "Table créée à partir de \*df1\*")

Table créée à partir de *df1*

| experience | replicat | var1 | var2 |
| --- | --- | --- | --- |
| traitement\_1 | a | 6.4237925 | 4.2261574 |
| traitement\_1 | b | 19.4171188 | 19.5508665 |
| traitement\_1 | c | 6.9770028 | 7.1829615 |
| traitement\_1 | d | 3.6681290 | 6.0805621 |
| traitement\_2 | a | 10.4564611 | 10.1626664 |
| traitement\_2 | b | 10.7931042 | 10.3491235 |
| traitement\_2 | c | 9.6881760 | 9.8994129 |
| traitement\_2 | d | 0.2367445 | -0.1713871 |
| traitement\_3 | a | 1.6947278 | 3.1888882 |
| traitement\_3 | b | 0.3112151 | 0.3208445 |
| traitement\_3 | c | 1.9347036 | 2.7849680 |
| traitement\_3 | d | 15.4347570 | 15.2160255 |

Il y a bien d’autres paramètres dans la fonction *kable()*, par exemple, il est possible de choisir le nombre de chiffres utilisés après la virgule ou encore d’inclure les numéros de lignes.

```{r table2}  
kable(tab1, caption = "Table créée à partir de \*df1\*", digits = 3, row.names = TRUE)  
```

kable(tab1, caption = "Table créée à partir de \*df1\*", digits = 3, row.names = TRUE)

Table créée à partir de *df1*

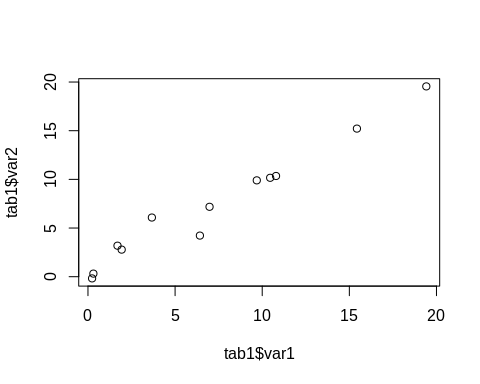
|  | experience | replicat | var1 | var2 |
| --- | --- | --- | --- | --- |
| 1 | traitement\_1 | a | 6.424 | 4.226 |
| 2 | traitement\_1 | b | 19.417 | 19.551 |
| 3 | traitement\_1 | c | 6.977 | 7.183 |
| 4 | traitement\_1 | d | 3.668 | 6.081 |
| 5 | traitement\_2 | a | 10.456 | 10.163 |
| 6 | traitement\_2 | b | 10.793 | 10.349 |
| 7 | traitement\_2 | c | 9.688 | 9.899 |
| 8 | traitement\_2 | d | 0.237 | -0.171 |
| 9 | traitement\_3 | a | 1.695 | 3.189 |
| 10 | traitement\_3 | b | 0.311 | 0.321 |
| 11 | traitement\_3 | c | 1.935 | 2.785 |
| 12 | traitement\_3 | d | 15.435 | 15.216 |

Pour en apprendre davantage, reportez-vous à la documentation de cette fonction. Je tiens au passage à signaler le package [kableExtra](https:/%20cran.r-project.org/web/packages/kableExtra/vignettes/awesome_table_in_html.html) qui offre de très nombreuses fonctionnalités pour créer des tableaux plus complexes, je vous recommande chaudement d’y jeter un coup d’œil.

### Figures

Avec le package rmarkdown, il est très facile d’insérer les figures produites avec R dans un document, le plus simple est de regarder ce que génère un simple plot généré avec la fonction plot(). Je reprends le data frame créé plus haut pour cela.

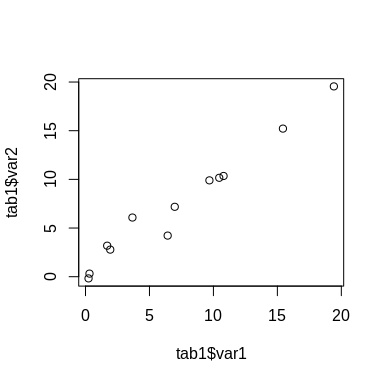
plot(tab1$var1, tab1$var2)



J’obtiens la figure demandée, avec des dimensions par défaut qui prennent une bonne partie de la largeur du document, mais pas son entièreté. 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%20reference.pdf). Dans le bloc de code ci-dessous, la taille de la figure et son alignement sont ajustés et une légende est ajoutée.

```{r figdim1, fig.cap = "Ceci est la légende de la figure", fig.height = 4, fig.width = 4, fig.align = "right"}  
plot(tab1$var1,tab1$var2)  
```

plot(tab1$var1, tab1$var2)



une figure de avec une légende

Les dimensions d’une figure sont toujours exprimées en [pouces](https://fr.wikipedia.org/wiki/Pouce_(unit%C3%A9)), si vous n’êtes pas habitué à manipuler les pouces, il faudra faire des conversations à la main (ou avec R) en gardant en tête que 1 pouce vaut 2.54 cm. Il est également possible de choisir une des deux dimensions et changer le rapport de forme, fig.aspect. Ainsi, pour une figure de  poucesx pouces, je peux utiliser une largeur de 9 pouces et un rapport de forme de 1.5.

```{r figdim2, fig.cap= "Figure 9x6", fig.width = 9, fig.aspect = 1.5}  
plot(tab1$var1,tab1$var2)  
```

plot(tab1$var1,tab1$var2)

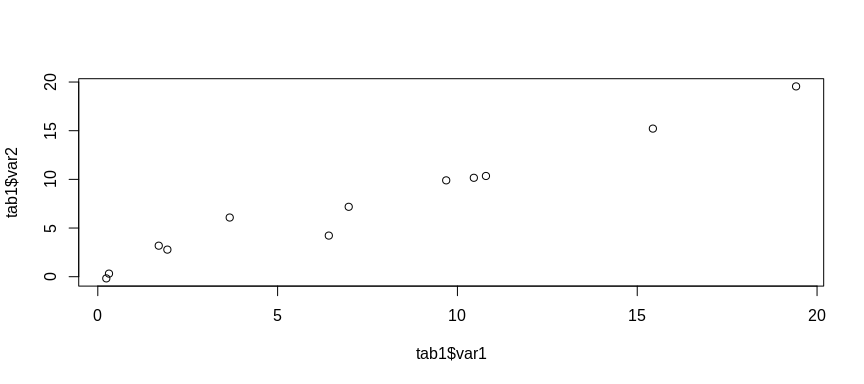
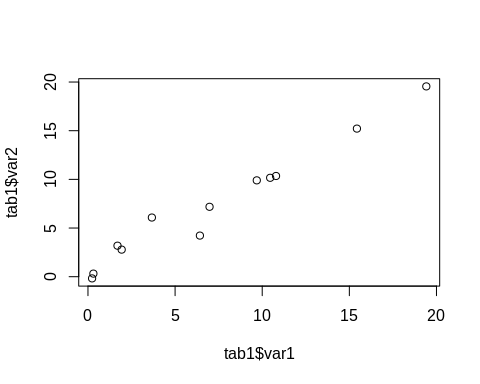


Figure 9x6

Il est aussi possible de contrôler la largeur de la sortie en utilisant des pourcentages, pour cela on utilise out.width

```{r figdim3, out.width = "100%"}  
plot(tab1$var1, tab1$var2)  
```

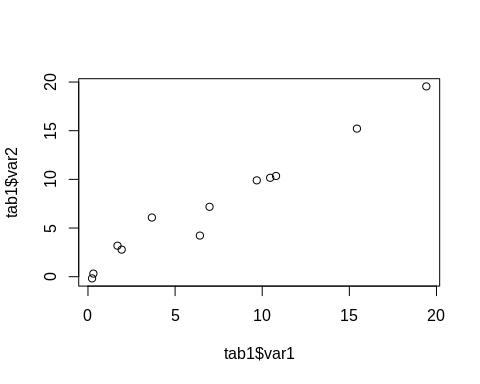
plot(tab1$var1, tab1$var2)



Il est aussi possible de changer le format de sortie des figures avec le paramètre dev (la liste complète des formats est disponible à <https://yihui.org/knitr/options/>).

```{r figdim4, out.width = "100%", dev = "jpeg"}  
plot(tab1$var1, tab1$var2)  
```

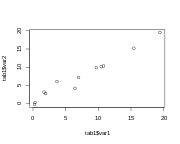
plot(tab1$var1, tab1$var2)



Pour les formats dits matriciels, il est possible de spécifier la résolution de l’image avec fig.dpi

```{r figdim5, out.width = "100%", dev = "png", fig.cap= "Faiblrésolution", dpi = 36}  
plot(tab1$var1, tab1$var2)  
```

plot(tab1$var1, tab1$var2)

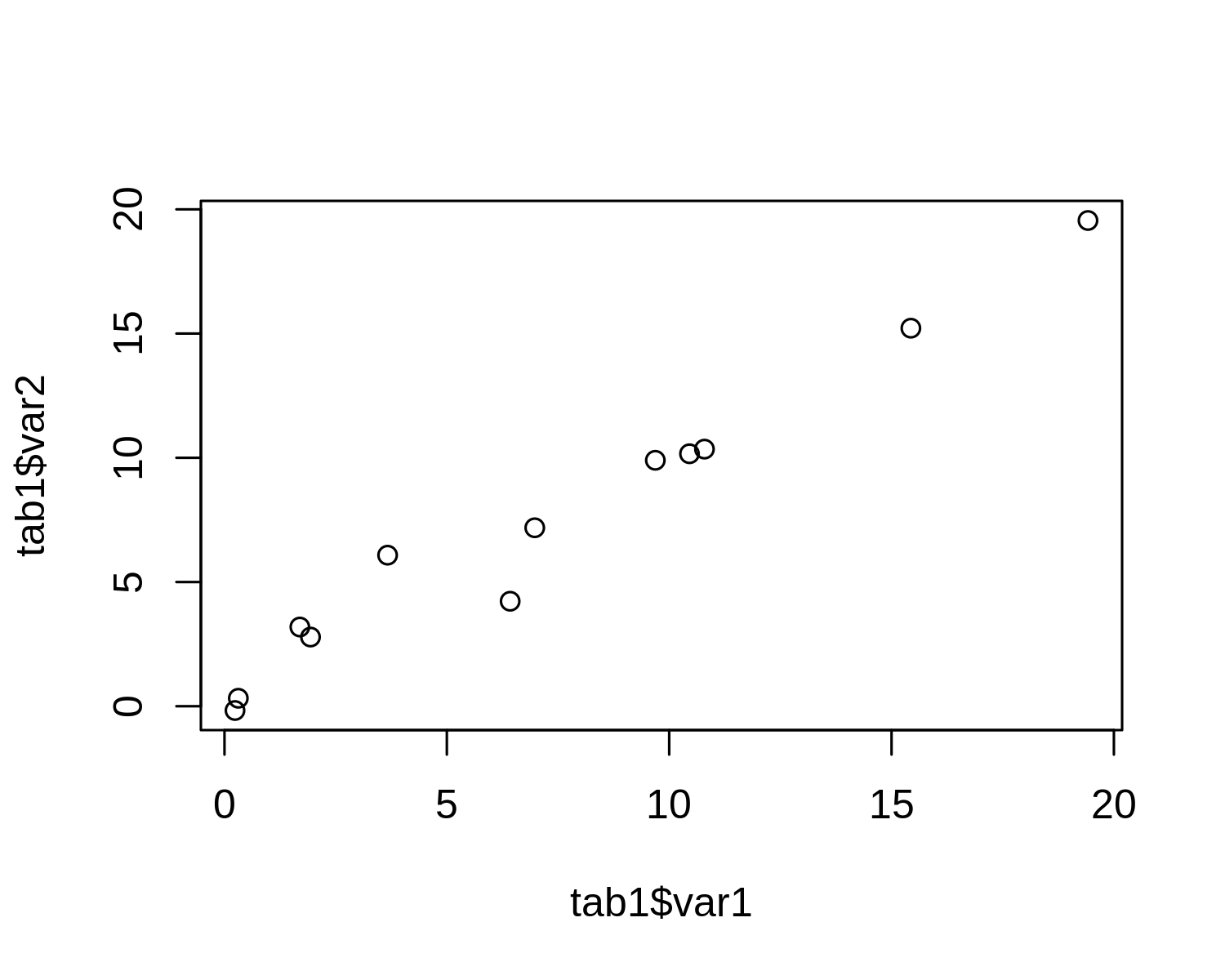


Faible résolution

```{r figdim6, out.width = "100%", dev = "png", fig.cap= "Fortrésolution", dpi = 300}  
plot(tab1$var1, tab1$var2)  
```

ce qui donne

plot(tab1$var1, tab1$var2)

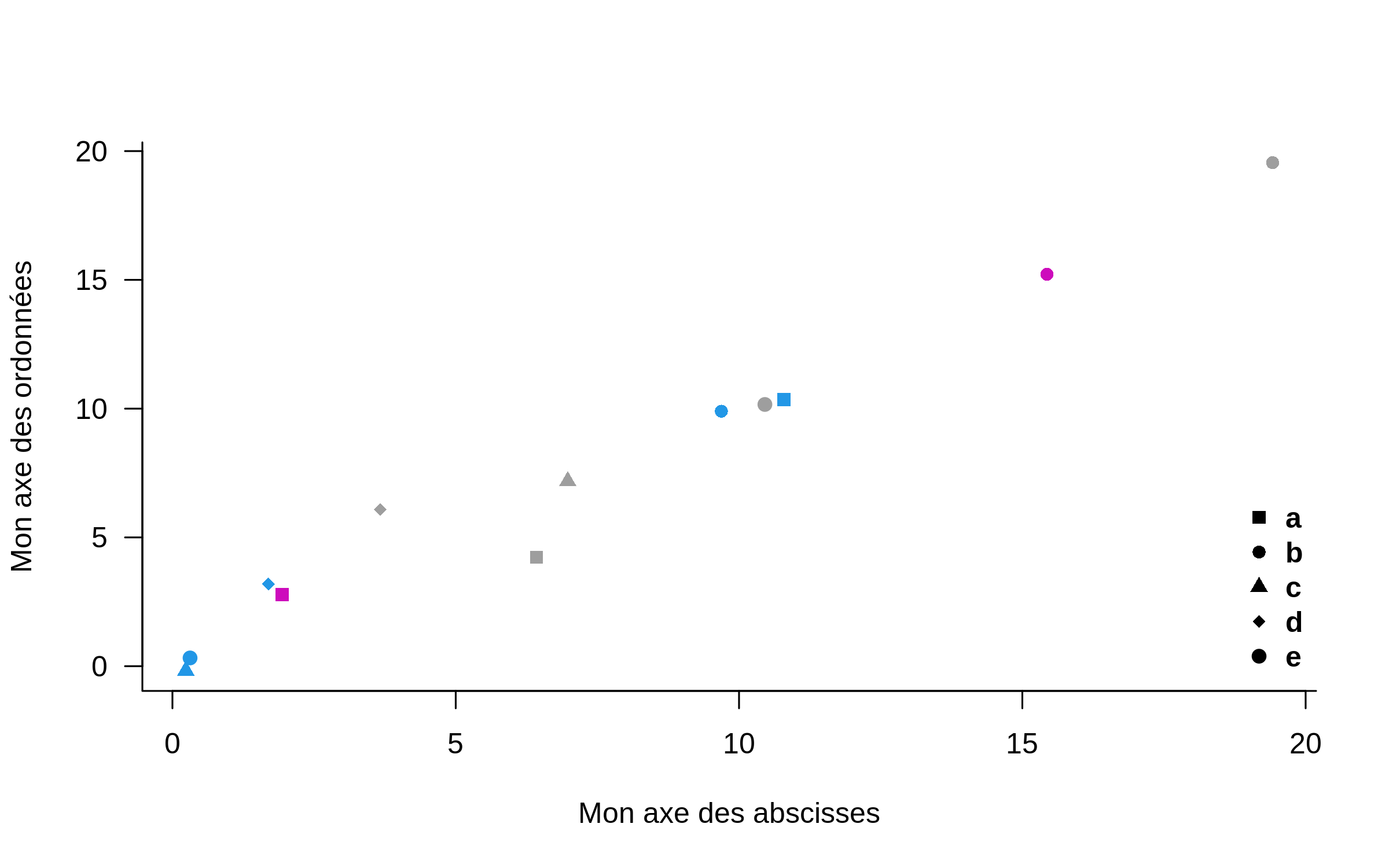


Forte résolution

Encore une fois, il y a [d’autres options listées sur le site de knitr](https://yihui.org/knitr/options/), il est possible, entre autres, de choisir l’emplacement où les figures seront sauvegardées avec fig.path et d’ajouter un hyperlien sur la figure avec l’option fig.link. L’ensemble de ces paramètres nous permettent de facilement bénéficier de la large gamme de fonctions graphiques de R. Ainsi, avec R Markdown, il est possible, par exemple, de personnaliser la figure précédente avec quelques lignes de code supplémentaires, d’en choisir la taille, le format, la position, la légende, d’ajouter un hyperlien et de masquer le code qui la génère. Voici le bloc qui nous permet de modifier toutes ces options&nbsp;:

{r figfinale, dev = "png", dpi = 300, fig.height = 5, fig.width =8, echo = FALSE, fig.link = "https://github.com/KevCaz/Rmarkdowndocfr", fig.cap = "Figure associé à la table 1. Le gris pour l'expérience 1, bleu pour la seconde et violet pour la troisième."}  
par(bty="l", font = 2, las = 1)  
plot(tab1$var1, tab1$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")  
```

et ce dernier nous donne:

[](https://github.com/KevCaz/Rmarkdowndocfr)

### Graphiques non générés par R

Dans la section traitant des [images](Les_images) avec Pandoc Markdown nous avons vu comment insérer une image dans un document. Il est aussi possible de faire une telle inclusion grâce à la fonction include\_graphics() de knitr. L’avantage de cette manière de procéder est qu’elle permet de traiter le fichier extérieur comme un graphique produit par R. Par exemple, pour ajouter le logo de R utilisé plus haut (./images/Rlogo.png), il est possible d’utiliser:

```{r iclgraph, fig.align = "center", out.width = "40%", echo = FALSE}  
include\_graphics("./images/Rlogo.png")  
```



## Application: modèle linéaire dynamique

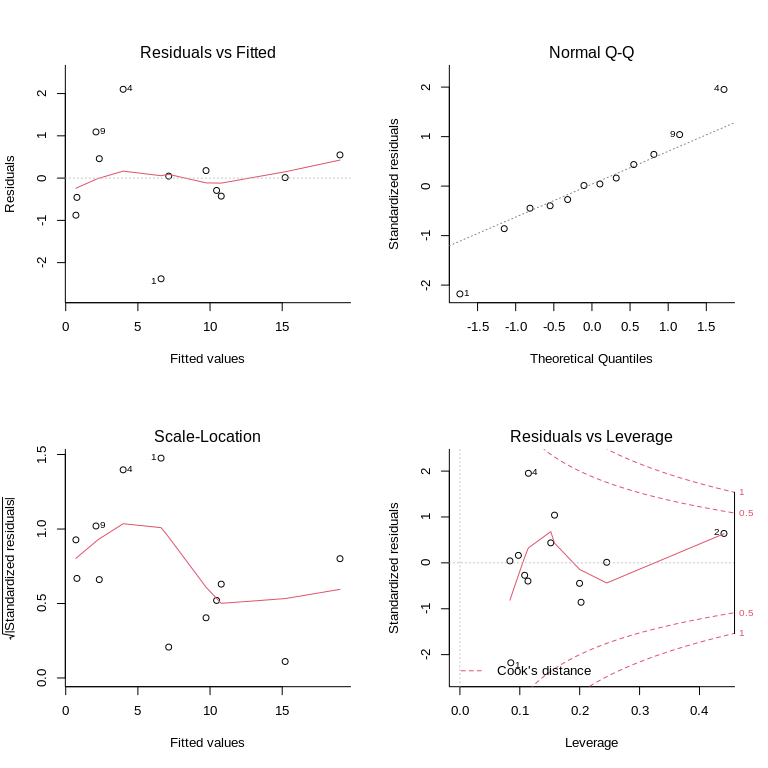
Supposons que je souhaite faire un modèle linéaire avec la variable explicative *var1* et la variable à expliquer *var2* de notre *tab1*. Pour ce faire je vais utiliser lm() et afficher les coefficients ainsi que les figures associées. J’utilise alors le bloc de code suivant&nbsp;:

```{r application, echo = FALSE, fig.cap = "Mon modèle linéaire", fig.width= 8,  
fig.height = 8, fig.align = "center"}  
mod1 <- lm(var2 ~ var1, data = tab1)  
kable(summary(mod1)$coefficients)  
par(bty = "l", mfrow = c(2, 2))  
plot(mod1)  
```

avec le commentaire suivant: “La pente de la regression est de `r format(summary(mod1)$coefficients[2, 1], digits = 3)` (et je mettrai le résultat en gras).

Le résultat est le suivant&nbsp;:

|  | Estimate | Std. Error | t value | Pr(>|t|) |
| --- | --- | --- | --- | --- |
| (Intercept) | 0.4805576 | 0.5244976 | 0.9162246 | 0.3811135 |
| var1 | 0.9539094 | 0.0562021 | 16.9728420 | 0.0000000 |



Mon modèle linéaire

La pente de la regression est de **0.954**.

Si je change les données de tab1, la table, les graphiques et mon commentaire seront adéquatement modifiés, ce qui peut s’avérer très utile pour créer rapidement des rapports similaires, par exemple un rapport mensuel pour lequel seules les données changent.

# Organiser son document et son code

Jusqu’ici nous avons vu comment utiliser Markdown et les bloc de code R pour rendre le document dynamique. Ici, nous allons passer en revue différentes fonctionnalités qui nous permettent de mieux structurer notre document, surtout quand il est long et que beacoup de code est utilisé.

## Inclure un fichier R Markdown dans un fichier R Markdown

Pour des raisons d’organisation, il est parfois plus simple d’organiser un long document en différents fichiers. Par exemple, quand différentes personnes travaillent avec sur différentes parties d’un rapport. L’options chlid nous permet d’inclure un fichier R Markdown dans un autre document R Markdown. Par exemple pour inclure le fichier [part.Rmd](https://github.com/KevCaz/Rmarkdowndocfr/scripts/part.Rmd), nous utilisons

```{r include\_part, child = "scripts/part.Rmd"}  
  
```

et voici ce que nous obtenons:

*début du document part.Rmd*

print("Ceci est le code R du fichier part.Rmd")

## [1] "Ceci est le code R du fichier part.Rmd"

*fin du document part.Rmd*

## Utiliser un script externe

Pour insérer du code dans le document R Markdown, il peut parfois être plus pratique d’utiliser un script externe, autrement avoir une partie (ou l’entièreté) du code d’un fichier .R (ou .r). Plusieurs manières de procéder existent, plus ou moins pertinentes selon ce que nous voulons faire. Si le code est simplement un ensemble de lignes à exécuter de manière silencieuse, alors faire appel à la fonction source() est probablement une bonne option. Il y a cependant une petite chose à savoir : il est préférable d’utiliser l’argument local pour s’assurer que le code sera exécuté dans le bon environment, voir le chapitre *Source external R scripts* de Xie et al. (2020) (aussi disponible à l’URL suivante <https://bookdown.org/yihui/rmarkdown-cookbook/source-script.html>). Le script à charger est disponible à l’URL suivante <https://github.com/KevCaz/Rmarkdowndocfr/scripts/scr_externe1.R>, il contient les lignes suivantes&nbsp;:

var1 <- 2  
var2 <- 10\*var1  
var3 <- cos(var2)  
var4 <- sin(var2)

que nous chargeons avec la fonction source()

```{r source-script, include = FALSE}  
source("scripts/scr\_externe1.R", local = knitr::knit\_global())  
```

Une fois chargé, nous pouvons accéder aux variables qui y sont déclarées, par exemple&nbsp;:

var4

## [1] 0.9129453

Comme vous l’aurez peut-être compris, utiliser source() ne nous permet pas d’afficher le code contenu dans le script. Si pour une raison ou une autre il est préférable d’afficher le code, il est possible d’utiliser l’option code pour enter le code à ajouter.

```{r showextcode, code ="var1 <- 1"}  
```

var1 <- 1

Ce qui est pratique avec cette option, c’est qu’il est possible d’utiliser une fonction qui va lire un fichier. Il devient alors facile d’insérer le contenu d’un script R. C’est possible de le faire avec la fonction readLines(), ou encore avec la function read\_utf8() du package [xfun](https://CRAN.R-project.org/package=xfun) qui retourne un avertissement si le fichier contient des caractères qui ne sont pas encodés en UTF-8. Ainsi, le bloc suivant

```{r showextcode, code = readLines("scripts/scr\_externe1.R")}  
```

nous donne

var1 <- 2  
var2 <- 10\*var1  
var3 <- cos(var2)  
var4 <- sin(var2)

Une limite de cette approche est qu’elle nous permet pas d’exécuter différente partie du script dans différent bloc de code, ce qui peut, à des fins pédagogiques, être utile. Il existe cependant un moyen simple d’utiliser différentes parties d’un script R en l’annotant avec des balises @knitr. Prenons le script [scr\_externe2.R](https://github.com/KevCaz/Rmarkdowndocfr/scripts/scr_externe2.R)

# @knitr part1  
var1 <- 2  
var2 <- 10\*var1  
# @knitr end\_part1  
  
# @knitr part2  
var3 <- cos(var2)  
var4 <- sin(var2)  
# @knitr end\_part2

Le code est séparé en deux parties, chacune limitée par des balises @knitr qui suivent le format suivant

# @knitr id  
code  
# @knitr end\_id <-- NB: optionnel! ouvrir une nouvelle balise fermera la précédente

Une fois les balises ajoutées dans le script, pour l’utiliser, la première étape est de charger le script, ce qui ce fait avec la fonction read\_chunk().

read\_chunk("scripts/scr\_externe2.R")

La seconde étape est d’ajouter les blocs de code qui contiennent le ou les parties à utiliser. Pour cela on utilise un bloc de code R avec l’identifiant de la partie à utiliser entre chevrons doublés. Par exemple, pour ajouter la partie 1, nous utilisons

```{r part1}  
<<part1>>  
```

ce qui donne

var1 <- 2  
var2 <- 10\*var1

Nous utilisons la même approche pour la partie 2

```{r part2}  
<<part2>>  
```

ce qui nous donne

var3 <- cos(var2)  
var4 <- sin(var2)

Notons que les commandes ajoutées à la suite seront ignorées.

## Extraire du code R

La function purl() du package knitr offre la possibilité d’extraire le code R d’un fichier RMD. Pour ce faire, il suffit d’utiliser la fonction de la manière suivante&nbsp;:

purl(input = "UtiliserRMarkdown.Rmd", output = "code.R")

L’ensemble du code R extrait de UtiliserRMarkdown.Rmd sera écrit dans le fichier code.R qui peut-être utilisé tel quel. Les blocs de code seront, par défaut, identifiés avec des commentaires incluant le nom donné au bloc de code, ainsi que les options choisies. Par exemple, ci-dessous est présenté le bloc de code de la section [Modifier l’affichage du code source avec le paramètre echo](#X21e0296aa15067dbd604642410e06bad0c71838).

## ----sanscode, echo = FALSE--------------------------------------------------  
# une addition de variables avec R  
a <- 2  
b <- 3  
a + b

Il est possible d’ajuster le niveau de documentation ajoutée dans le script R produit par purl() grâce au paramètre documentation. Avec documentation = 0, les balises qui permettent d’identifier les blocs de code sont enlevées et avec documentation = 2, tout le texte du document est ajouté et commenté.

L’entièreté du script est disponible à l’URL suivante&nbsp;: <https://kevcaz.github.io/Rmarkdowndocfr/code.R>. Cette fonctionnalité peut se révéler vraiment efficace pour commenter un script un peu compliqué et le partager pour utilisation. Dans de tels cas d’usage, il y a souvent des blocs de code qui peuvent être superflus, par exemple des bouts de code ajoutés pour améliorer les explications, mais qui ne sont pas utiles pour la bonne exécution du script. Dans ce cas, il est possible d’utiliser l’option purl pour enlever les blocs de code. Par défaut, l’option est réglée sur TRUE et donc tous les blocs de code sont ajoutés. Le bloc de code suivant sera ajouté dans le document R Markdown avec les sorties associées, mais pas dans le fichier code.R.

{r nopurl, purl = FALSE} # ce bloc sera dans le document mais pas dans code.R cos(pi)

# ce bloc sera dans le document mais pas dans code.R   
cos(pi)

## [1] -1

# Utilisations avancées

Dans cette section nous allons voir comment créer nos propres options pour bloc de code.

## Modifier les options par défaut

Le package knitr nous offre aussi la possibilité de changer non seulement les options par défaut mais aussi le comportement des options existantes ainsi que la possibilité de créer nos propre options <https://yihui.org/knitr/objects/>

* opts\_chunk: manages options for code chunks
* opts\_current:
* opts\_knit: manages options for the knitr package
* knit\_hooks: manages hook functions
* knit\_patterns: manages regular expressions to extract R code from the input document
* knit\_engines: functions to deal with other languages

function ont une méthode get() pour accéder aux différente valeur, une function set() pour changer les

## Créer des points d’ancrage personnalisés

WIP

Comme nous l’avons vu plus haut, il y a de nombreuses options disponibles pour ajuster l’affiche des blocs de code et des sorties associées. Le package knitr nous offre aussi la possibilité de changer non seulement les options par défaut mais aussi le comportement des options existantes ainsi que la possibilité de créer nos propre options. Cela se fait grâce à des points d’ancrage, *hooks* en anglais[[20]](#footnote-193), (<https://yihui.org/knitr/hooks/>) Il en existe différents types:

1. les points d’ancrange pour les bloc de code (*chunk hooks*): knit\_hooks
2. les points d’ancrange des sorties (*output hooks)*: knit\_hooks
3. les points d’ancrage des options (*options hooks*): opts\_chunk

Ci-dessous, nous détaillons comment les utiliser.

### Changer les options par défaut

Les options par défaut d’un bloc de code sont stockées dans la liste opts\_chunk et nous avons accès à ses valeurs avec la fonction get() de cette liste.

knitr::opts\_chunk$get()

## $eval  
## [1] TRUE  
##   
## $echo  
## [1] TRUE  
##   
## $results  
## [1] "markup"  
##   
## $tidy  
## [1] FALSE  
##   
## $tidy.opts  
## NULL  
##   
## $collapse  
## [1] FALSE  
##   
## $prompt  
## [1] FALSE  
##   
## $comment  
## [1] "##"  
##   
## $highlight  
## [1] TRUE  
##   
## $size  
## [1] "normalsize"  
##   
## $background  
## [1] "#F7F7F7"  
##   
## $strip.white  
## [1] TRUE  
##   
## $cache  
## [1] FALSE  
##   
## $cache.path  
## [1] "UtiliserRMarkdown\_cache/docx/"  
##   
## $cache.vars  
## NULL  
##   
## $cache.lazy  
## [1] TRUE  
##   
## $dependson  
## NULL  
##   
## $autodep  
## [1] FALSE  
##   
## $cache.rebuild  
## [1] FALSE  
##   
## $fig.keep  
## [1] "high"  
##   
## $fig.show  
## [1] "asis"  
##   
## $fig.align  
## [1] "default"  
##   
## $fig.path  
## [1] "/home/runner/work/Rmarkdowndocfr/Rmarkdowndocfr/public/UtiliserRMarkdown\_files/figure-docx/"  
##   
## $dev  
## [1] "png"  
##   
## $dev.args  
## NULL  
##   
## $dpi  
## [1] 96  
##   
## $fig.ext  
## NULL  
##   
## $fig.width  
## [1] 5  
##   
## $fig.height  
## [1] 4  
##   
## $fig.env  
## [1] "figure"  
##   
## $fig.cap  
## NULL  
##   
## $fig.scap  
## NULL  
##   
## $fig.lp  
## [1] "fig:"  
##   
## $fig.subcap  
## NULL  
##   
## $fig.pos  
## [1] ""  
##   
## $out.width  
## NULL  
##   
## $out.height  
## NULL  
##   
## $out.extra  
## NULL  
##   
## $fig.retina  
## NULL  
##   
## $external  
## [1] TRUE  
##   
## $sanitize  
## [1] FALSE  
##   
## $interval  
## [1] 1  
##   
## $aniopts  
## [1] "controls,loop"  
##   
## $warning  
## [1] TRUE  
##   
## $error  
## [1] FALSE  
##   
## $message  
## [1] TRUE  
##   
## $render  
## NULL  
##   
## $ref.label  
## NULL  
##   
## $child  
## NULL  
##   
## $engine  
## [1] "R"  
##   
## $split  
## [1] FALSE  
##   
## $include  
## [1] TRUE  
##   
## $purl  
## [1] TRUE

knitr::opts\_current$get()

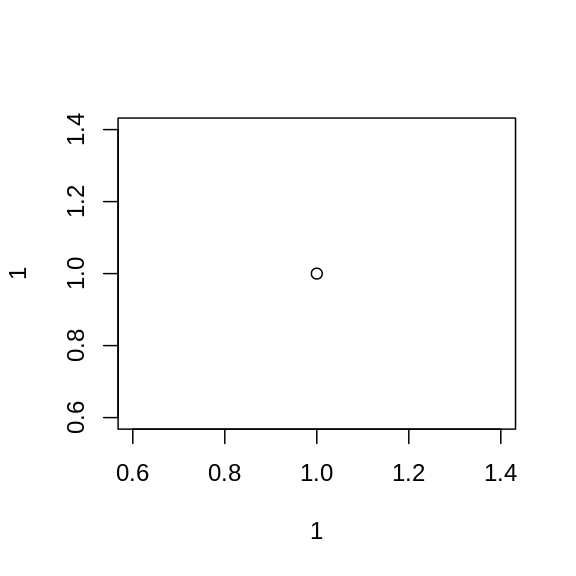
#R> $eval  
#R> [1] TRUE  
#R>   
#R> $echo  
#R> [1] TRUE  
#R>   
#R> $results  
#R> [1] "markup"  
#R>   
#R> $tidy  
#R> [1] FALSE  
#R>   
#R> $tidy.opts  
#R> NULL  
#R>   
#R> $collapse  
#R> [1] FALSE  
#R>   
#R> $prompt  
#R> [1] FALSE  
#R>   
#R> $comment  
#R> [1] "#R>"  
#R>   
#R> $highlight  
#R> [1] TRUE  
#R>   
#R> $size  
#R> [1] "normalsize"  
#R>   
#R> $background  
#R> [1] "#F7F7F7"  
#R>   
#R> $strip.white  
#R> [1] TRUE  
#R>   
#R> $cache  
#R> [1] 0  
#R>   
#R> $cache.path  
#R> [1] "UtiliserRMarkdown\_cache/docx/"  
#R>   
#R> $cache.vars  
#R> NULL  
#R>   
#R> $cache.lazy  
#R> [1] TRUE  
#R>   
#R> $dependson  
#R> NULL  
#R>   
#R> $autodep  
#R> [1] FALSE  
#R>   
#R> $cache.rebuild  
#R> [1] FALSE  
#R>   
#R> $fig.keep  
#R> [1] "high"  
#R>   
#R> $fig.show  
#R> [1] "asis"  
#R>   
#R> $fig.align  
#R> [1] "default"  
#R>   
#R> $fig.path  
#R> [1] "/home/runner/work/Rmarkdowndocfr/Rmarkdowndocfr/public/UtiliserRMarkdown\_files/figure-docx/"  
#R>   
#R> $dev  
#R> [1] "png"  
#R>   
#R> $dev.args  
#R> NULL  
#R>   
#R> $dpi  
#R> [1] 96  
#R>   
#R> $fig.ext  
#R> NULL  
#R>   
#R> $fig.width  
#R> [1] 5  
#R>   
#R> $fig.height  
#R> [1] 4  
#R>   
#R> $fig.env  
#R> [1] "figure"  
#R>   
#R> $fig.cap  
#R> NULL  
#R>   
#R> $fig.scap  
#R> NULL  
#R>   
#R> $fig.lp  
#R> [1] "fig:"  
#R>   
#R> $fig.subcap  
#R> NULL  
#R>   
#R> $fig.pos  
#R> [1] ""  
#R>   
#R> $out.width  
#R> NULL  
#R>   
#R> $out.height  
#R> NULL  
#R>   
#R> $out.extra  
#R> NULL  
#R>   
#R> $fig.retina  
#R> [1] 1  
#R>   
#R> $external  
#R> [1] TRUE  
#R>   
#R> $sanitize  
#R> [1] FALSE  
#R>   
#R> $interval  
#R> [1] 1  
#R>   
#R> $aniopts  
#R> [1] "controls,loop"  
#R>   
#R> $warning  
#R> [1] TRUE  
#R>   
#R> $error  
#R> [1] FALSE  
#R>   
#R> $message  
#R> [1] TRUE  
#R>   
#R> $render  
#R> NULL  
#R>   
#R> $ref.label  
#R> NULL  
#R>   
#R> $child  
#R> NULL  
#R>   
#R> $engine  
#R> [1] "R"  
#R>   
#R> $split  
#R> [1] FALSE  
#R>   
#R> $include  
#R> [1] TRUE  
#R>   
#R> $purl  
#R> [1] TRUE  
#R>   
#R> $label  
#R> [1] "unnamed-chunk-5"  
#R>   
#R> $code  
#R> [1] "knitr::opts\_current$get()"  
#R>   
#R> $out.width.px  
#R> [1] 480  
#R>   
#R> $out.height.px  
#R> [1] 384  
#R>   
#R> $params.src  
#R> [1] " comment = \"#R>\""  
#R>   
#R> attr(,"class")  
#R> [1] "knitr\_strict\_list"

Nous pouvons changer ces valeurs avec set() et donc nous pouvons a tout moment changer le comportement du reste des bloc de code. Par exemple, avec

knitr::opts\_chunk$set(  
 comment = "#R>",  
 dev = "png",  
 dpi = 144,  
 fig.align = 'center',  
 fig.width = 4  
)

Avec ces nouveaux réglages par défaut (voir ce que retourne knitr::opts\_chunk$get() maintenant), les commentaires seront affichés en commençant pas #R>, les figures seront exportées en format PNG avec une résolution de 155&nbsp;dpi (*dot per inch*, point par pouce en français), elles seront alignées au centre avec une largeur de 4 pouces. Par exemple

plot(1, 1)



A tout moment, il est possible de restaurer les valeurs par défaut contenues dans le package knitr en appelant la fonction restore() de la liste (opts\_chunk).

knitr::opts\_chunk$restore()

### Ajouter une nouvelle option

Dans le context d’utilisation de knitr, un point d’ancrage est une fonction qui sera appelée quand l’option à laquelle elle a été assignée sera non null. Elle permet de rouler du code avant ou après le code chunk en utilisant éventuellement les autres options du code chunk. Comme expliquée par l’auteur de knitr, (voir <https://yihui.org/knitr/hooks/#chunk-hooks>), la déclaration se fait en ajoutant une fonction de la manière suivante

knit\_hooks$set(mon\_hook = function(before, options, envir) {  
 if (before) {  
 ## code à exécuter avant le bloc de code   
 } else {  
 ## code à exécuter après le bloc de code   
 }  
})

Quelques exemples sont donnés au lien suivant: <https://github.com/yihui/knitr-examples/blob/master/045-chunk-hook.md>. Nous les reprenons avec quelques modifications.

knit\_hooks$set(hook1 = function(before, options, envir) {  
 if (before) {  
 "\_Code exécuté avant le bloc!\_\n\n"  
 } else {  
 "\n\n\_Code exécuté après le bloc...\_"  
 }  
})

```{r ex\_hook1, hook1 = TRUE}  
1 + 1  
```

*Code exécuté avant le bloc!*

1 + 1

## [1] 2

*Code exécuté après le bloc…*

options(c("opt1"))

## $opt1  
## NULL

# voir https://github.com/yihui/knitr-examples/blob/master/045-chunk-hook.md  
knit\_hooks$set(foo2 = function(before, options, envir) {  
 if (!before) {  
 z = capture.output(str(options[c("eval", "foo2", "bar1")]))  
 z = paste(" ", z, sep = "", collapse = "\n")  
 paste("Some chunk options in the above chunk are:\n\n", z, sep = "")  
 }  
})

Pour déclencher la fonction, il suffit d’utiliser une valeur non null, par exemple

```{r, foo2 = TRUE, bar1 = "cool"}  
```

Some chunk options in the above chunk are:

List of 3  
 $ eval: logi TRUE  
 $ foo2: logi TRUE  
 $ bar1: chr "cool"

ou une valeur

```{r, foo2 = 10, bar1 = "cool"}  
```

Some chunk options in the above chunk are:

List of 3  
 $ eval: logi TRUE  
 $ foo2: num 10  
 $ bar1: chr "cool2"

et donc on peut utiliser toute les valeurs dans le *hook!*

## Exécuter des blocs de code dans un autre language

### Les engins disponibles

Comme nous l’avons vu jusqu’ici, R Markdown est un outil privilégié pour R, les blocs de code sont exécutés et les sorties y sont ajoutées. Un des principes fondamentaux de R est de s’interfacer avec différents langages (Chambers, 2016). R est ainsi installé avec des outils pour rendre possible cette interface (par exemple la fonction .Call(), .C() et .Fortran()) et il existe packages qui facilitent l’utilisation d’autres languages dans R. Le plus populaire de ces packages est [Rcpp](https://CRAN.R-project.org/package=Rcpp) qui offre une intégration poussée avec [C++](https://isocpp.org/). Il en existe d’autres comme [reticulate](https://CRAN.R-project.org/package=reticulate) (Ushey et al., 2021) qui connecte R avec [Python](https://www.python.org/) et [JuliaCall](https://CRAN.R-project.org/package=JuliaCall) (Li, 2019) qui fait le lien entre R et [Julia](https://julialang.org/). Grâce à ces packages qui connectent R avec d’autres languages, il est possible de les intégrer directement dans R Markdown en sélectionnant le bon engin[[21]](#footnote-206). La liste des engins disponibles est contenue dans knit\_engines qui se manipule comme opts\_chunk.

names(knitr::knit\_engines$get())

## [1] "awk" "bash" "coffee" "gawk" "groovy" "haskell"   
## [7] "lein" "mysql" "node" "octave" "perl" "psql"   
## [13] "Rscript" "ruby" "sas" "scala" "sed" "sh"   
## [19] "stata" "zsh" "asis" "asy" "block" "block2"   
## [25] "bslib" "c" "cat" "cc" "comment" "css"   
## [31] "dot" "embed" "fortran" "fortran95" "go" "highlight"  
## [37] "js" "julia" "python" "R" "Rcpp" "sass"   
## [43] "scss" "sql" "stan" "targets" "tikz" "verbatim"

Nous allons prendre pour exemples Python (voir aussi <https://github.com/rstudio/reticulate#python-in-r-markdown>) et Julia (voir <https://cran.r-project.org/web/packages/JuliaCall/vignettes/Julia_in_RMarkdown.html>). Il est donc important que les deux packages soient installés, ainsi qu’une version de Python et de Julia.

install.packages(c("reticulate", "JuliaCall"))

Voici les lignes qui vous permettent de vérifier si tout est correctement et de vérifier quelles versions de Python et Julia sont installées.

reticulate::py\_config()

## python: /usr/bin/python3  
## libpython: /usr/lib/python3.8/config-3.8-x86\_64-linux-gnu/libpython3.8.so  
## pythonhome: //usr://usr  
## version: 3.8.10 (default, Nov 26 2021, 20:14:08) [GCC 9.3.0]  
## numpy: [NOT FOUND]  
##   
## python versions found:   
## /usr/bin/python3  
## /usr/bin/python

JuliaCall::julia\_command("VERSION")

## v"1.7.0"

### Utiliser Python

Pour Python, voici l’engin utilisé.

knitr::knit\_engines$get()$python

## function (options)   
## {  
## if (isFALSE(options$python.reticulate)) {  
## eng\_interpreted(options)  
## }  
## else {  
## if (!loadable("reticulate"))   
## warning2("The 'python' engine in knitr requires the reticulate package. ",   
## "If you do not want to use the reticulate package, set the chunk option ",   
## "python.reticulate = FALSE.")  
## reticulate::eng\_python(options)  
## }  
## }  
## <environment: namespace:knitr>

Et voici un exemple de comment utiliser un bloc de code Python dans R Markdown.

```{python py1, comment = "#py>"}  
[2, 2] \* 3  
```

[2, 2] \* 3

#py> [2, 2, 2, 2, 2, 2]

Si vous reprenez la ligne de l’engin utilisé, vous comprenez que toutes les options du bloc de code et les données sont passées à la fonction eng\_python() du package reticulate qui s’occupe d’exécuter le code et de réunir les sorties.

Voici un deuxième exemple avec cette fois si une fonction pour calculer la longueur du grand côté d’un triangle rectangle (hypoténuse) à partir de la connaissance de la longueur des deux autres côtés.

def hypothenuse(a, b):  
 return (a \*\* 2 + b \*\* 2) \*\* 0.5

Une fois déclaré, nous pouvons l’appeler dans une autre balise qui utilise l’engin `python.

hypothenuse(4, 3)

#py> 5.0

### Utiliser Julia

Pour Julia, voici l’engin utilisé.

knitr::knit\_engines$get()$julia

## function (options)   
## {  
## JuliaCall::eng\_juliacall(options)  
## }  
## <environment: namespace:knitr>

Dans ce cas ci, le code et les options sont passées à a fonction eng\_juliacall() du package JuliaCall.

```{julia jl1, comment = "#jl>"}  
[2, 2] .\* 3  
```

[2, 2] .\* 3

#jl> 2-element Vector{Int64}:  
#jl> 6  
#jl> 6

Voici la même function hypothenus() en julia

function hypothenuse(a, b)  
 return (a^2 + b^2)^0.5  
end

#jl> hypothenuse (generic function with 1 method)

hypothenuse(4, 3)

#jl> 5.0

Notons qu’il n’y a pas de conflit entre les deux fonctions hypothenuse, l’une est associé aux blocs de code python et l’autre aux blocs de code julia.

Ici nous venons de montrer que nous pouvons utiliser d’autre langages dans des blocs de code utilisant des engins distincts. Cela nous a permis de constater que R Markdown peut être un outil très efficace pour comparer comment différentes opérations sont réalisées par différents languages. Pour Julia et Python, les engins utilisés sont issus de packages qui implémentent une interface avec R. Grâce à ces packages, nous pouvons utiliser du code Python et du code Julia directement dans R, ce qui est très bien expliqué dans la documentation de ces packages. Ici, nous nous sommes limités à utiliser des engins dont le but est de générer des blocs de code comme ci nous faisions un appel direct (sans passer par R) à ces languages.

# Conclusion

TODO

R Markdown permet de convertir un document en de nombreux formats. Aujourd’hui, il y a en fait tout un écosystème de packages qui utilisent rmarkdown pour permettre de réaliser autres types de support. Voici quelques exemples, probablement les plus populaire

* [bookdown](https://CRAN.R-project.org/package=bookdown) git book
* [blogdown](https://CRAN.R-project.org/package=blogdown) Hugo, Jekyll
* [pagedown](https://CRAN.R-project.org/package=pagedown) pagejs
* [xaringan](https://CRAN.R-project.org/package=xaringan) remark

# Références

Chambers, J.M., 2016. Extending R, The R Series. CRC Press, Boca Raton London New York.

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.

Li, C., 2019. JuliaCall: an R package for seamless integration between R and Julia. The Journal of Open Source Software 4, 1284. <https://doi.org/10.21105/joss.01284>

Oreskes, N., Shrader-Frechette, K., Belitz, K., 1994. Verification, Validation, and Confirmation of Numerical Models in the Earth Sciences. Science (80-. ). 263, 641‑646.

Simpkin, S., 2020. Débuter avec Markdown. Programming Historian en français. <https://doi.org/10.46430/phfr0007>

Ushey, K., Allaire, J., Tang, Y., 2021. [reticulate: Interface to ’Python’](https://CRAN.R-project.org/package=reticulate).

Xie, Y., 2017. Dynamic Documents with R and Knitr. CRC Press, Boca Raton, Florida.

Xie, Y., 2014. Dynamic Documents with R and Knitr, Chapman & Hall/CRC the R Series. CRC Press, Taylor & Francis, Boca Raton.

Xie, Y., Allaire, J.J., Grolemund, G., 2019. R Markdown: The Definitive Guide. CRC Press, Taylor and Francis Group, Boca Raton.

Xie, Y., Dervieux, C., Riederer, E., 2020. R Markdown Cookbook, 1ʳᵉ ed, The R Series. Taylor and Francis, CRC Press, Boca Raton.

1. <https://en.wikipedia.org/wiki/John_Gruber>, consulté le 1er juin 2020. [↑](#footnote-ref-24)
2. <https://fr.wikipedia.org/wiki/ReStructuredText>, consulté le 1er juin 2020. [↑](#footnote-ref-27)
3. <https://blog.codinghorror.com/responsible-open-source-code-parenting/>, consulté le 1er juin 2020. [↑](#footnote-ref-29)
4. Par exemple 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-37)
5. La version 1 n’utilise pas Pandoc. [↑](#footnote-ref-40)
6. Pandoc est d’ailleurs capable de gérer différentes variantes de Markdown. [↑](#footnote-ref-42)
7. <https://pandoc.org/MANUAL.html#pandocs-markdown>, 1er juin 2020. [↑](#footnote-ref-44)
8. voir <https://pandoc.org/MANUAL.html#templates>, consulté le 1er juin 2020. [↑](#footnote-ref-58)
9. du contenu préliminaire. [↑](#footnote-ref-74)
10. voir ><https://fr.wikibooks.org/wiki/LaTeX/%C3%89crire_des_math%C3%A9matiques>>, consulté le 2 juin 2020. [↑](#footnote-ref-75)
11. Voir la section “Math” à l’URL suivant ><https://pandoc.org/MANUAL.html#pandocs-markdown>>, consulté le 2 juin 2020. [↑](#footnote-ref-78)
12. <https://cran.r-project.org/web/packages/yaml/index.html>, consulté le 8 juin 2020. [↑](#footnote-ref-85)
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-99)
14. Par exemple, avec HTML les deux paragraphes seront chacun des deux paragraphes seront des éléments paragraphes (balise <p>). [↑](#footnote-ref-108)
15. Voir la section *Fenced code blocks* du manuel de référence <https://pandoc.org/MANUAL.html>. [↑](#footnote-ref-111)
16. 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-120)
17. la première [↑](#footnote-ref-136)
18. la seconde [↑](#footnote-ref-137)
19. Les logiciels de gestion de bibliographie génèrent ces fichiers. [↑](#footnote-ref-141)
20. La traduction de *hook* est crochet mais point d’ancrage est peut-être plus approprié ici. [↑](#footnote-ref-193)
21. traduit de *engine* en anglais. [↑](#footnote-ref-206)