Dynamic Documents
Using knitr

Benjamin Hofner

benjamin.hofner@fau.de
Institut für Medizininformatik, Biometrie und Epidemiologie (IMBE)
Friedrich-Alexander-Universität Erlangen-Nürnberg

Biometrisches Kolloquium 2015
A very short history of “Literate Programming”

- The name *Literate Programming* (mixing the source code with documentation) was first coined by Knuth [1984].
- *Sweave*, a function that is shipped with base R, was created by Leisch [2002].
- In the following years, *Sweave* gained much attention but showed some shortcomings. Several add on packages to *Sweave* tried to fix these issues (e.g. syntax highlighting, better graphics and more devices, cache)
- *knitr* (vers. 0.1) was released in 01/2012 by Xie [2014, 2015] to enhance (and replace?) *Sweave*.

▶ We will focus on *knitr* in this tutorial.
Why should one use literate programming?

- Instead of copy and paste output, figures, etc. to your document, they are part of the document.
  - It is by far less error prone.

- It is easier to update the report, e.g., if the data changes.
  - In summary it is often faster to use literate programming even if it requires more work in the first place.
Getting started

- Open RStudio
- Install `knitr`, e.g., via `install.packages("knitr")`
- Go to Tools ▶ Global Options ... ▶ Sweave and select `knitr` as default.
Basic Idea

The basic idea is to mix code and narratives (aka text). \texttt{knitr} can be used in combination with \LaTeX, HTML, Markdown etc. To make the code identifiable it is wrapped in special tags. In \LaTeX these are:

\begin{verbatim}
<< options >>=
code
@
\end{verbatim}

Code can also be used in-line by typing \code{\Sexpr{code}} in \LaTeX. The latter is supposed to produce a single (textual) result.

If we run \texttt{knitr} on the document, it searches for the special tags, extracts the code, evaluates it and replaces it with the results (depending on the code and on the options).
A minimal example in RStudio

- Go to new file (red circle)
- Select R Sweave
- Now we can save this file (use the default file extension `.Rnw` which stands for R noweb) and start working.
A minimal example in RStudio (ctd)

- To insert text simply write some text (after \begin{document} )
- To insert text with special markup use the Format drop down menu (red circle).
- To insert code, use the Chunks drop down menu (blue circle).
- To select chunk options hit the Tab key when the cursor is between << and >>=. This works also for auto-completion, i.e., after you started typing an option name.
Chunk options

- Chunk options follow the rule: `name = value`.
- The value can be R code itself to allow conditional executions.
- The first option might not have a `name`, in this case it is the chunk label:

```r
<<first_chunk, echo=TRUE, results='markup'>>=
print("123")
@
```

- For more on chunk labels and their application see below.
Important chunk options

- **eval**: evaluate the chunk? (TRUE / FALSE)
- **tidy**: tidy up the code? (FALSE / TRUE ▶ red circles)
- **results**: type of result, either R-like output (‘markup’), as is (‘asis’ ▶ blue circles), push everything to the end of the chunk (‘hold’) or ’hide’ it.
- **highlight**: Use syntax highlighting? (TRUE / FALSE ▶ green rectangle)
- **echo**: Show R command? (TRUE / FALSE ▶ green rectangle is missing)
- **include**: Include the chunk in the output? (TRUE / FALSE)

Details and more options: [http://yihui.name/knitr/options/](http://yihui.name/knitr/options/)

\(^1\)first option is always the default
Global options

- Errors are reported to the output document.
- Note that the knitr process isn’t stopped on errors!
- Reporting of error, warning and message (TRUE / FALSE) should usually be turned off if one writes a report, at least the last two options.
- One can do this separately for each chunk.
  - Better do this globally, i.e., for all chunks.
    One can use global chunk options which are set in a chunk at the beginning of the document (and which will hold from this point onwards):

```r
<<init, include=FALSE>>=
opts_chunk$set(warning = FALSE, message = FALSE)
@
```

- Obviously, all other options can be changed globally as well.

(In Sweave one needs to use the \LaTeX-like command `\SweaveOpts{}`)
Tables as output

- To create tables, one needs to use other packages which return a \LaTeX-table, e.g., `xtable` [Dahl, 2014], `Hmisc` [Harrell, 2015], `tables` [Murdoch, 2014].
- Use these packages and their functions in a chunk with option `results='asis'`.
- Easier reporting of descriptive statistics (including figures) and model summaries (with confidence intervals, pretty p-values, ...) is implemented in `papeR` [Hofner, 2015].
- An improved development version can be found on GitHub: [http://GitHub.com/hofnerb/papeR](http://GitHub.com/hofnerb/papeR).
- To install from GitHub use:

  ```r
  library("devtools")
  install_github("hofnerb/papeR")
  ```
**Figures as output**

- **knitr** is build to achieve a nice work-flow for graphics.
- Graphics can be created as if you were using the R console directly.
- If a chunk contains a graphic, it is automatically created and inserted in the document.

```r
<<test>>=
plot(1:10, main = "a first test")
@
```

- The label is used as file name for the figure.
- Usually one should use `echo = FALSE` for figure chunks.
Important chunk options for figures

- **fig.path**: store the figures here. Defaults to `'figure/'`. Can also be e.g. `'folder/prefix-'`, where `folder` will be automatically created.
- **fig.width, fig.height**: width and height of figure device in inches. Both default to 7.
- **out.width, out.height**: width and height of figure in output. E.g. `out.width='0.8\textwidth'` (Attention: use `\`).
- **fig.show**: (how to) show the figure. `'asis'` / `'hide'` / ...
- **dev**: plotting device. Default depends on the output: `'pdf'` (for \LaTeX) or `'png'` (for HTML and Markdown). Interesting device for \LaTeX: `'tikz'`.
- **fig.keep**: plots are produced as a sequence. Which plots should be kept and displayed? `'high'` / `'all'` / `'last'` / ...

Details and more options: [http://yihui.name/knitr/options/](http://yihui.name/knitr/options/)
Cache

- To speed up computation, one can use cached versions of chunks.
- This can be turned on by

```r
<<chunk1, cache=TRUE>>=
plot(1:10, main = "a first test")
```

Per default this option is **FALSE**.

- Code is only executed at the first time and if it is changed.

- What happens if not the code inside the chunk but the data, pre-computed objects (i.e., code in other chunks), the package versions, ... change?
- Per default, the code chunk is **not** re-evaluated.
Cache – Dependencies

- To define dependencies to other chunks one can use the option `dependson`. One can either specify a numeric vector such as `1:5` or `c(-1, -2)` or a vector containing the chunk names.
- Dependencies are only possible for cached chunks.
- The (still experimental) option `autodep` tries to automatically determine dependencies if set to `TRUE`.
Cache – Further dependencies

- To cache further changes these need to be included as chunk options.

**Note:** One can specify arbitrary chunk options.

- To check if the data set has changed on disk, one could use, e.g., the modification time as option

```r
c<<chunk1, cache=TRUE, data_version=file.info("dataset.csv")$mtime>>=
data <- read.csv("dataset.csv")
@
```

- Similarly, we could also depend on the value of a variable `x` from a different chunk by including, e.g., `cache.dep = x`.
Cache – Random numbers

- Usually, the current state of the variable `.Random.seed` is also part of the cached objects. This helps to make simulations in combination with `cache = TRUE` reproducible.

- All random numbers drawn after the cached chunk will be exactly the same as if the chunk was executed.

- To re-run the cached chunk if the current state of the seed changes (e.g. by drawing another random number before that chunk) one needs to add a dependency on `.Random.seed`

```r
<<chunk1, cache=TRUE, cache.seed=.Random.seed>>=
```

or set this global option:

```r
opts_chunk$set(cache.seed = rand_seed)
```
Cache – Side effects

- Note that “side effects” are usually not cached.
  - E.g. global options (`options()`), graphics options (`par()`), or global `knitr` options (`opts_chunk$set()`) might affect the execution of all following code.
  - These settings are, however, not recorded and cached.
  - Global options and other functions with side effects (e.g. `setwd()`) need to be set in an un-cached chunk!

For details on `cache` see [http://yihui.name/knitr/demo/cache/](http://yihui.name/knitr/demo/cache/). Please read the notes there as well, as using `cache = TRUE` might have unintended side effects!
Cross references

Re-using chunks

- Chunks can be easily re-used (to avoid code duplication).
- If a labeled (!) chunk is defined it can be used in any other chunk as follows:

  ```
  <<par, eval = FALSE>>=
  par(mfrow = c(2, 2))
  @
  
  <<myfig>>=
  <<par>>
  for (i in 1:4)
    plot(i)
  @
  ```
Re-using chunks

Code chunks can also be used within another command as long as they are only preceded by spaces.

```r
<<mysum, eval = FALSE, echo = FALSE>>=
sum(1:10)
@
```

```r
<<total, echo = FALSE>>=
sum(11:20) +
  <<mysum>>
@
```

```r
## [1] 210
```

```r
<<NOT_WORKING>>=
sum(11:20) + <<mysum>>
@
```
Cross references

Re-using whole chunks

- Code chunks can also be re-used as a whole.
- Same way as above,
- or use an empty chunk with the same label,

```r
<<mysum, eval = TRUE>>=
@

sum(1:10)
## [1] 55

or use the option `ref.label`

```r
<<mysum2, ref.label=c("mysum", "total"), tidy=TRUE>>=
@

sum(1:10)
## [1] 55

sum(11:20) + sum(1:10)
## [1] 210
```
Cross references

Using external .Rnw files

- Similar result as \input{file.tex} but for \texttt{knitr} files.
- External file \texttt{external.Rnw} (a mixture of code and narratives):

\begin{quote}
This text is an external text.
A random normal sample:
<<ext_chunk>>=
set.seed(1503)
rnorm(2)
@
\end{quote}

- We use the chunk option \texttt{child} with an empty chunk

\begin{quote}
<<use_external_file, child = "external.Rnw">>=
@
\end{quote}

\textbf{This text is an external text.}  A random normal sample:

\begin{verbatim}
set.seed(1503)
rnorm(2)
## [1] -0.2142477  1.2526165
\end{verbatim}
Using *knitr* with *markdown*

- Instead of *\LaTeX* we can use Markdown to write the narratives.
- Markdown is a simple markup language (with many different flavors or dialects).
- For *R markdown* basic syntax see the RStudio help or [http://rmarkdown.rstudio.com/authoring_basics.html](http://rmarkdown.rstudio.com/authoring_basics.html).
- For further details see [http://rmarkdown.rstudio.com/](http://rmarkdown.rstudio.com/).
- The standard document suffix for markdown is `.md`, for *knitr* type markdown files it becomes `.Rmd`.
- Note that the chunk tags change:
  
  **Open chunk:** `<<label, opts>>=`  
  
  **Close chunk:** `@`  
  
  **In-line code:** `\Sexpr{code}`  

- One can *knit* the *R markdown* files to HTML, PDF or even Word.
• Open new R Markdown file:

• If doing this for the first time, one needs to install some packages:

• Set up file:

• To knit the file or change the options use these icons:
Further resources & References

- Online manual: http://yihui.name/knitr
- Chunk options: http://yihui.name/knitr/options
- More on Reproducible Research in R: http://cran.r-project.org/web/views/ReproducibleResearch.html


Benjamin Hofner. *papeR: A Toolbox for Writing knitr, Sweave or Other LaTeX-based Papers and Reports*, 2015. URL http://github.com/hofnerb/papeR. R package version 0.6-0.


