Guidelines for authors

October 3, 2016

1 Overview

This book is being written in LaTeX, using `knitr` for dynamic report generation, and git for version control. This document aims to describe what these things are, why they are useful, and how to use them. Learning these tools might seem time consuming, however, they will greatly save time in the long run (not only for this project, but also for future projects any of us may be involved in).

The source file for this document (`guidelines.Rnw`) uses many of these tools. It is worth comparing to the compiled PDF (`guidelines.pdf`) to understand what some of the code is doing. Feel free to make minor modifications and see what effects these have upon recompiling the document.

Compiling a document under this framework is straightforward in RStudio: first, ensure `knitr` is set to weave `.Rnw` files (Click `Tools` > `Sweave` > `Weave Rnw files using: knitr`), then simply open the `.Rnw` file and click "Compile PDF".

This document also provides various guidelines to aid consistency throughout the book, for example, by setting out some rules for notation and code style.

2 Repository structure

This project is hosted as a git repository on GitHub at `https://github.com/david-borchers/SCR-Book` (more about git and GitHub in Section 5). The general structure of this repository is as follows:

1. The main source file is called `scr-book.Rnw`, and is in the main directory of this repository.
2. Source files for individual chapters are found inside the directory `chapters/`.
3. Figures generated automatically by `knitr` are found inside the directory `figure/` (more on `knitr` in Section 4).
4. Figures *not* generated by `knitr` (e.g., any photos) are found inside the directory `keepfigure/`.
5. This book includes examples from various data sets. The `analysis/` directory contains data (in `analysis/data/`), code to fit models to these data sets (in `analysis/code/`), and `.RData` files generated from the code that contain R objects with information about the fitted models (in `analysis/objects/` (more on this in Section 6).

3 LaTeX

LaTeX is a freely available typesetting system that is widely used within scientific fields. As most of us use LaTeX fairly reguarly, this document will not cover how to use it. Various introductory guides are available online.

3.1 Guidelines

Here are some guidelines for LaTeX code style:

1. Label all chapters, sections, equations, figures, and tables so that we can reference them automatically. Labels for chapters should begin with `chap:`, those for sections should begin with `sec:`, and those for equations should begin with `eq:`. Labels for figures and tables are mentioned below in the `knitr` section. Provide informative labels; for example, `\label{chap:detector-types}` is good, and `\label{chap:chap4}` is bad.

2. Use these labels to reference chapters, sections, equations, figures, and tables. For example, "In Chapter `\ref{chap:detector-types}` we discuss..." is good, and "In Chapter 4 we discuss..." is bad.

3. Use `\eqref{}` instead of `\ref{}` to reference equations.

3.2 Bibliography

[Somebody write something for how we are doing references.]

4 knitr

In the preparation of a document, R code is often run to calculate values displayed in-text or in tables, and to produce plots. Often this R code is called separately, and the required values are copy-pasted into text or tables, and plots are inserted using `\includegraphics{}`.

This is not very efficient. If you wish to change a plot, or recalculate values to write in-text or insert into tables, you must find the R code you originally wrote, rerun it, copy-paste values across, and replace the file containing the image of the plot. The `knitr` package provides an alternative: it allows you to insert chunks of R code into your LaTeX source file, directly calculating values and generating plots to appear in the final document. This means everything used to create the document appears in one file, and if you want to make a change you only need to make a change in one place—the R code. It also allows us to neatly display any code we want to appear in the final document.

4.1 Plots

Below is a simple example of a `knitr` chunk that generates Figure 1 (but you'll only be able to see it in the source `guidlines.Rnw` document).

As you can see, a chunk starts with options enclosed in angled brackets (`<<>>`). Enclosed within these are a number of possible options, some of which have been used above:

1. The first is the name of the chunk, which also gets passed to any plots the chunk generates. Keep this an informative name (e.g., `frog-survey-area`, not `plot1`).

2. `echo`: Logical (i.e., either `TRUE` or `FALSE`). If `TRUE`, the code within the chunk is also displayed in the document. You can't see the R code that generated Figure 1 in this PDF because the this option was set to `FALSE`.

3. `include`: Logical. If `TRUE` output from the chunk is shown in the final document. For example, it is useful to set this to `FALSE` if the chunk is only being used to calculate values that are used in later chunks.

4. `eval`: Logical. If `TRUE` the code is run when compiling the document; otherwise, it is not. For example, it is useful to set this to `FALSE` if you want to display code in the final document, but you do not want to run it.

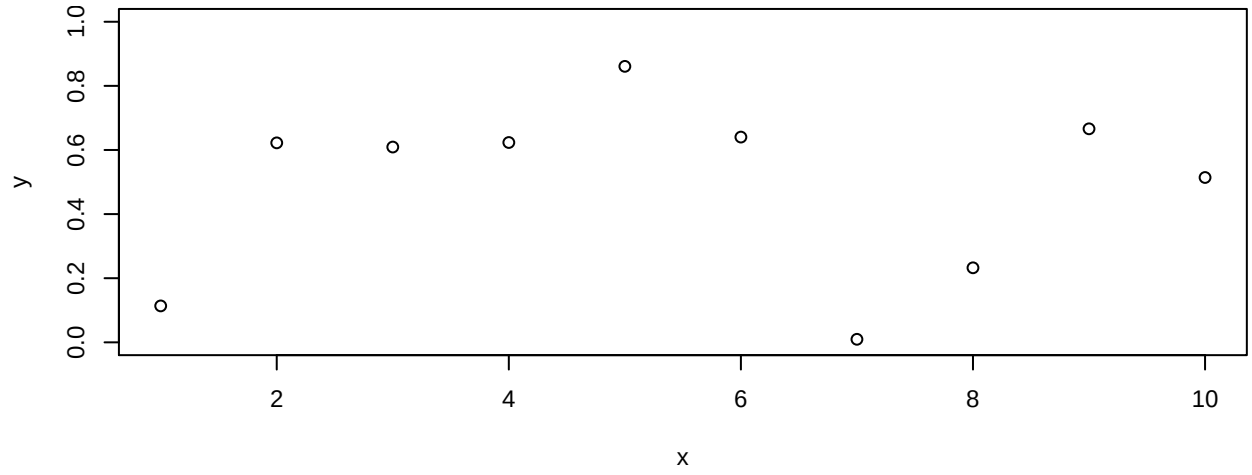

Figure 1: A simple caption.

5. `fig.width` and `fig.height`: The width and height of the PDF file generated by R. Note that in a code chunk at the top of this document the R objects `textwidth` and `textheight` have been specified as the width and height of the text on the page. This can be useful if you want to specify the width of the plot to exactly span the width of the text, or some proportion of the height of the page, as is the case here.

6. `out.width` and `out.height`: Slightly different to `fig.width` and `fig.height`, as these specify the dimensions of the image as they appear on the page (as opposed to the dimensions of the PDF generated by R). Many people ignore these options, but we should *always ensure they are the same as* `fig.width` and `fig.height`, but wrapped in the `add.in()` function. This is a little fiddly, but it ensures that any text appearing in any plots (e.g., axes labels) will be the same font size across all plots we generate, which is nice.

7. `fig.pos`: A character string that specifies where on the page to put the plot (as one would normally put in square brackets after `\figure` in regular LaTeX).

8. `fig.cap`: The figure caption.

Descriptions of other chunk options are available here. Note that we can reference the figure produced by a chunk using `\ref{fig:chunk-name}`. Here is an example: the simple plot above is labelled Figure 1.

4.2 Values in-text

Inserting values into text is straightforward using `\Sexpr`. For example, in the code chunk `simple-plot` a vector $\boldsymbol{y}$ was generated. The mean of $\boldsymbol{y}$ is 0.49. The standard deviation of $\boldsymbol{y}$ is 0.27. The largest element of $\boldsymbol{y}$ is 0.86. These values are being automatically inserted based on R calculations, rather than having to copy-paste them directly into the text. If we change the data $\boldsymbol{y}$, we do *not* need to change any of this text (which we might otherwise forget to do).

4.3 Tables

Tables can be inserted using `knitr` and the `xtable` R package. Below is an R chunk that shows a basic example resulting in Table 1, using the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ we generated in the previous code chunk.

Table 1: A simple table caption.

Vector	Mean	SD	Min	Max
$\boldsymbol{x}$	5.500	3.028	1.000	10.000
$\boldsymbol{y}$	0.489	0.275	0.009	0.861

```
library(xtable)
## Names of vectors.
names <- c("$\\bm{x}$", "$\\bm{y}$")
## Calculating means and standard deviations.
means <- c(mean(x), mean(y))
sds <- c(sd(x), sd(y))
## Calculating smallest and largest values.
smallest <- c(min(x), min(y))
largest <- c(max(x), max(y))
## Putting everything we want in our table into a data frame.
tab <- data.frame(names, means, sds, smallest, largest)
## Names of the columns in our table.
colnames(tab) <- c("Vector", "Mean", "SD", "Min", "Max")
caption <- "A simple table caption."
## Creating the table object.
tab.obj <- xtable(tab, label = "tab:simple-table", caption = caption, digits = 3)
## Printing the table object.
print(tab.obj, type = "latex", include.rownames = FALSE, hline.after = c(-1, 0, 2),
      caption.placement = "top", booktabs = TRUE, table.placement = "tb",
      sanitize.text.function = identity, math.style.negative = TRUE)
```

A few things to note:

1. Set the chunk option `results = "asis"`, otherwise the LaTeX code `xtable` has generated will appear in the document, rather than the table itself.
2. The `hline.after` argument specifies after which rows we'd like a horizontal line in the table. This part is a little weird, but the first row is the one for $\boldsymbol{x}$. If we want a horizontal line *before* the *first* row, we need to tell `xtable` to put a horizontal line *after* the 0th row. If we want a line *before* the column headers (the 0th row), we need to tell `xtable` to put a horizontal line *after* the -1st row. Hence, `hline.after = c(-1, 0, 2)`.
3. You can use LaTeX code in the table (e.g., to generate the $\boldsymbol{x}$ and $\boldsymbol{y}$ row names in Table 1), but the `print()` function must have `sanitize.text.function = identity`.
4. Setting the argument `booktabs = TRUE` means that LaTeX uses the `booktabs` package to make the table, which results in something slightly more aesthetically pleasing than default LaTeX tables.
5. You can do fairly complex tables with `xtable`, but the code can get pretty fiddly. Ben has spent many hours frustratingly learning the intricacies of `xtable`, and is happy to help out.

5 git

Version control allows a project's changes to be tracked over time. The program git is a very popular piece of software used for version control. There are various advantages to using git. Here are a few:

1. An entire version history is kept, and it is easy to switch between changes. If you rewrite a passage but think you made it worse, it simple to return back to the original. If you had some code that was working perfectly fine, but then you broke it and you can't remember what you did, again you can revert back.

2. Someone can rewrite something that was originally written by another author. If the original writer doesn't like it, they can easily discard their changes. Essentially, we don't have to be careful and polite about rewriting other people's stuff—we can easily pick and choose which changes to keep.

3. It manages conflicts. If two people change the same file at the same time, when they both "push" their changes back to the repository, git will merge the changes together to create a new file. If the people have changed the same parts, it will flag this and ask you to fix the conflict.

A git repository can be hosted anywhere (e.g., on your laptop or desktop, or on some server). Many people host theirs on the website GitHub (`https://github.com/`). GitHub provides a nice interface for looking at what is in your repository, and how it is changed over time.

Git is easy to use when working in RStudio. [Somebody write something for how git works in RStudio for the benefit of those who do not yet use git.]

General guidelines to be expanded upon:

1. On its way to compiling the PDF, LaTeX, `knitr`, and `xtable` generate a lot of extra files (e.g., `.aux`, `.toc`, and PDFs of plots in `figure/`). We do not want to be commiting these every time we make changes. Only commit the *source* files that you have modified yourself, and not any files that have been automatically generated by LaTeX, `knitr`, or `xtable`. To make these junk files invisible to git, add them to the list in the `.gitignore` file, found within the main directory of the repository.

6 Code

The book will include output from models fitted to various example data sets. We could put the code to fit these models directly into our `knitr` chunks, but this would mean that every time we compile our PDF we would have to wait for every single model in the book to be fitted. Instead, it is better we fit these models in advance, save the resulting R objects in `.RData` files, which are then loaded inside our `knitr` chunks using `load("file.RData")`. (Of course some of this code will be presented in this book to show the reader how to fit various models, but these will appear in `knitr` chunks with `eval = FALSE`, so that they are not actually run upon compilation of the PDF.)

Furthermore, it is important that we monitor any changes to any output presented in our book that are caused by (1) the introduction of errors to our code, or (2) updates to any software that we are using. For example, if an update to an R package causes our code to throw an error, then we want to change our code. If a package maintainer introduces a major bug and a bunch of estimates we present change drastically, we want to know about this as soon as possible (this would be a minor embarassment for the package maintainer, but a major one for us if the book went to print in such a state).

It is therefore important that we keep any code that fits models presented in our book in the same location, and test it to ensure it is working as we'd expect. Here is the general idea:

1. Put any data sets in `analysis/data/` (do we want a particular format like `.csv` or `.RData`, or is a mixture OK?).

2. Put code into `analysis/code/`. This code should load the relevant data, fit models, and save `.RData` files that include R objects with model results. Make sure these `.RData` files are saved to `analysis/objects/`. These can then be loaded by `knitr` code in our `.Rnw` files.

3. Write a test or two in `/analysis/code/tests.r` (using the `testthat` package framework) for each model fit to ensure the results are what we'd expect.

4. Ben's desktop will automagically run the code within all files in `analysis/code/` on a regular basis (daily or weekly or something). Any tests that fail can be investigated.

For everyone to successfully compile book's PDF, we'll all need the same R packages installed on each of our computers. To make this as easy as possible, the script `analysis/code/packages.r` is designed to install any required R packages that are not already on the user's machine. If any code you write requires a package that does not yet appear in the first line of this script, please add it. This applies to both code in any files within `analysis/code/packages.r` or within any `knitr` chunks within the book's source files. The first `knitr` chunk in the book's main `.Rnw` will run this script to check that the computer compiling the PDF has installed all packages it requires.

7 Code style

We want to have code that looks consistent throughout the book. (It would also be nice to be consistent throughout all the code in `analysis/code/`, but that is less important).

Here are a couple of style guides we could use:

1. Google's R style guide

2. Style guide in Hadley Wickham's Advanced R book. This is a slightly modified version of the Google style guide.

8 Writing style

Quick notes to be expanded upon:

1. Some notes from Eric:

   - Use British spelling ('centre', not 'center', global search-and-replace 'z' for 's', etc.).
   - Use British punctuation (punctuation outside quotation marks*, no commas after acronyms* like 'e.g.' and 'i.e.').
   - Italics for non-English words and abbreviations* (*et al*, *e.g.*, *etc.*).

2. Do we want to abide by a particular style guide so that we have recommendations for almost all possible questions we may have regarding style?

   - Ben typically abides by the Chicago Manual of Style (CMOS). He has a physical copy if anyone wants to see it. Otherwise, the university has an online subscription; visit this link.
   - *Note that starred points in notes made by Eric are not consistent with CMOS recommendations.
   - The *New Oxford Style Manual* might be more consistent with Eric's notes. There is an online document by Oxford (nowhere near as comprehensive as the book), which sketches a few style recommendations that seem to roughly follow Eric's notes.

3. There are some writing style guidelines in `SpringerStyleFiles/`; see the `instruct.pdf` document, along with other PDFs and templates (e.g., `chapter.Rnw` in the `templates` directory).

9 Notation

Quick notes to be expanded upon:

1. Use square brackets for any PDFs or PMFs (e.g., $[y]$, $[y|x]$).
2. How do we boldify vectors? I use the `bm` package so they are still italicised ($\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{\theta}$). Other people just use `\bf` ($\mathbf{x}$, $\mathbf{y}$, θ). I like the former because it also boldifies Greek letters (as can be seen with θ here), but I am not fussy.
3. Do we distinguish between random variables and fixed values? Upper vs lowercase?
4. Do we distinguish between vectors and matrices, or are they all just boldified? We can do matrices in uppercase and vectors in lowercase (but only if we have not already made the distinction of fixed vs random in the same way).