Reproducibility

Dom Owens

02/10/2019

There are three key problems with computing projects to consider:

- The amount of code, which is often very large and quickly growing - this can be divided into packages
- Code changes over time. Version control can coordinate this.
- Code must be read by humans.

Principles of reproducible research and literate programming are introduced to address these.

Reproducible Research

> **Ideally, computational analysis should be reproducible.** Research is **reproducible** when authors provide all the necessary data and the computer codes to run the analysis again, re-creating the results. *Claerbout & Karrenbach*

Often, research does not meet these criteria. This can be because of code being run across multiple machines, because of the many steps involved in cleaning data which may not be recorded, or due to random simulations being required. Access to high-performance computers and parallelisation software, as well as the possiblility of automating workflows mean these issues can be solved with technology.

We should consider the replication crisis, in which many published research papers offer results which other researchers have been unable to recreate in seperate settings. Researchers who follow the tenets of reproducible research stand to gain in many ways, five of which were outlined by Markowetz (https://www.researchgate.net/publication/286395745_Five_selfish_reasons_to_work_reproducibly):

1. Avoiding disaster
2. Writing papers easier
3. Convincing reviewers
4. Facilitating continuity of work
5. Building your reputation

We should distinguish this from **replicable** research, which the same authors define as a study that arrives at the same scientific findings as another study, collecting new data (possibly with different methods) and completing new analyses. This is also an important concept, and should not be neglected.

Literate Programming

A **Literate Program** is written with source code which produces two outputs: a plain-language document explaining the function of the code, and a program that can run. This is very useful for both communicating the ideas underlying the code, and maintaining consistency between code and documentation. Considering these strengths in relation to the three problems discussed in the introduction, literate programming is a useful tool for practicing reproducible research.

Rmarkdown is one tool for literate programming. Code can be embedded in the same source as plain language, allowing explanation of the program at a conceptual level. Note, this is not appropriate for making code for production, or for writing low-level functions.

Let's see an example:

Euler Problem 2: Even Fibonacci numbers

We wish to find the sum of the even Fibonacci numbers less than or equal to 4,000,000. We create a vector of the numbers less than 4,000,000

```
fibs <- c(1, 2)
n <- 2 # length of fibs
x <- fibs[1] + fibs[2] # value of the next Fibonacci number
while (x < 4e6) {
  n <- n + 1
  fibs <- c(fibs, x)
  x <- fibs[n-1] + fibs[n]
}
fibs
```

```
##  [1]       1       2       3       5       8      13      21      34
##  [9]      55      89     144     233     377     610     987    1597
## [17]    2584    4181    6765   10946   17711   28657   46368   75025
## [25]  121393  196418  317811  514229  832040 1346269 2178309 3524578
```

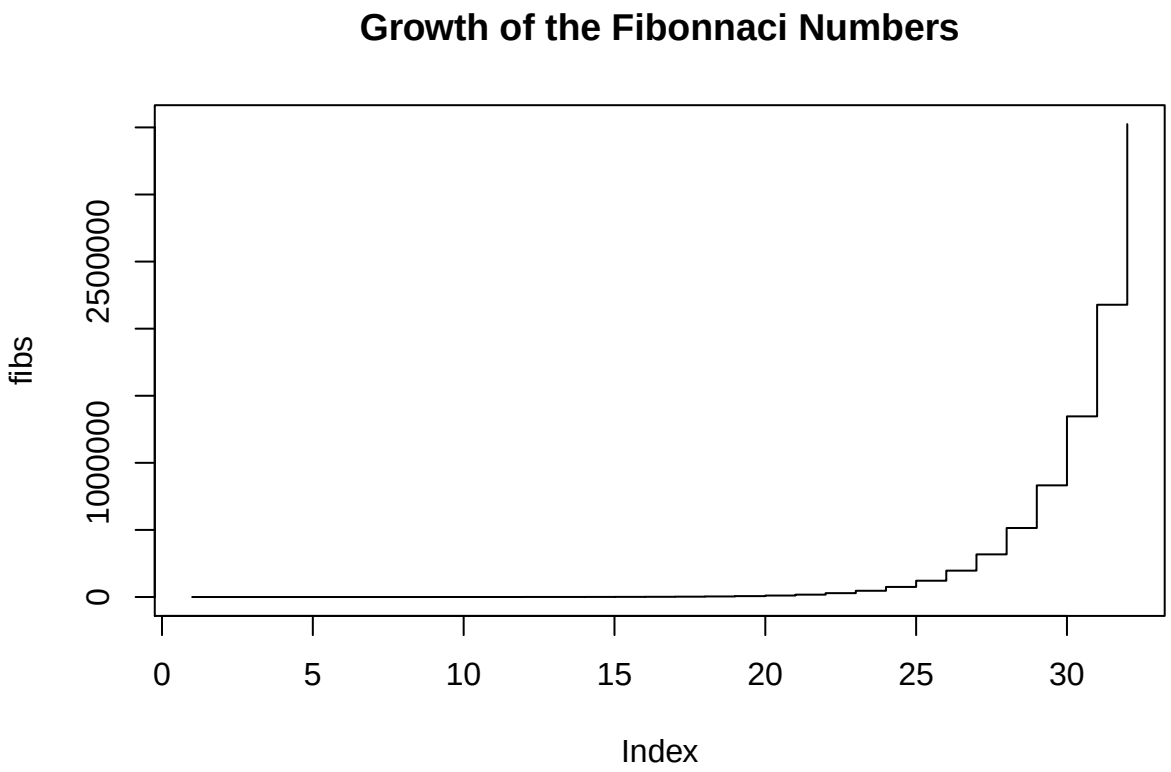
And sum all the numbers in the fibs vector that are even

```
s <- sum(fibs[fibs %% 2 == 0])
s
```

```
## [1] 4613732
```

We can plot the Fibonacci numbers to see how they grow

```
plot(fibs, type = "s", main = "Growth of the Fibonnaci Numbers")
```



We have seen that the code for solving the problem is executable, and each stage is accompanied by natural language to explain the purpose of the program. Code is split into chunks to group related commands.

Literate programming with LaTeX

We can incorporate R code into typeset LaTeX documents with knitr. Use the file extension .Rnw. The following code, when entered into a single document, will produce a PDF.

```
\documentclass{article}

\setlength{\parskip}{\medskipamount}
\setlength{\parindent}{0pt}

\title{A simple Rnoweb Document}
\author{Dom Owens}
\date{}

\begin{document}

\maketitle

This mostly a normal \LaTeX document.

For example, you can use mathematics: for $i \in {1,\ldots,n}$,
\[ S_n = \sum_{i=1}^n X_i. \]

The difference is that you can insert R code chunks

<<chunk-name, fig.height=5>>=
xs <- seq(0,2*pi,0.01)
ys <- sin(xs)
plot(xs, ys, type="l")
@
\end{document}
```

Which looks like this:

```
knitr::include_graphics("Literate_Programming_with_LaTeX.pdf")
```

This is useful for producing publication-standard documents, and for combining R content with mathematical figures.

A simple Rnoweb Document

Dom Owens

This mostly a normal LaTeXdocument.

For example, you can use mathematics: for $i \in 1, \ldots, n$,

$$S_n = \sum_{i=1}^{n} X_i.$$

The difference is that you can insert R code chunks

```
xs <- seq(0,2*pi,0.01)
ys <- sin(xs)
plot(xs, ys, type="l")
```

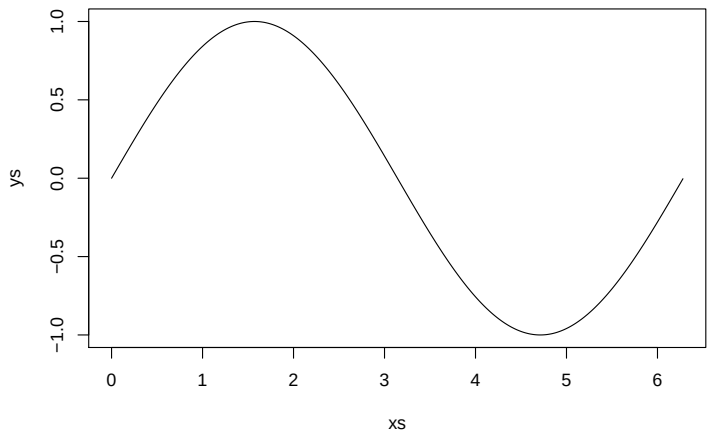


Figure 1: LaTeX Document in R Markdown