Reproducible Research: What is it? Why should we do it? How?

Christophe Pouzat

MAP5, Université de Paris et CNRS

christophe.pouzat@parisdescartes.fr

IDEEV, Gif/Orsay, November 15 2019

What is "Reproducible Research"?

- A short explanation: this is an approach aiming at reducing the gap between an ideal—research results should be reproducible—and reality—it is often hard, even for the authors, to reproduce published results—.
- In pratice it consists in providing to articles and books readers the complete set of data and codes together with an algorithmic description of how the codes were applied to the data to obtain the results.

At that stage, usually, two questions are asked:

- Why should I bother making my work reproducible if no one asks for it?
- Great, but how should I do it?

Some remarks

- In practice, what is meant by "reproducibility" here is what comes *after* data collection—it would indeed be more appropriate to speak about reproducible data analysis—.
- But "reproducible research" as we just defined it requires "free access" to data; the latter become therefore open to criticism and comparable: that's a big step towards data reproducibility per se.

The *Journal of Money, Credit and Banking*

- In the early 80s the editors of the *Journal of Money, Credit and Banking* started requesting from their authors the codes and data used for the "empirical papers" as well as the agreement to give access to those upon request (a project supported by the *National Science Foundation*).
- In 1986 a paper was published reporting a systematic attempt to reproduce the 54 empirical papers published between 1982 and 1984 (Dewald, Thursby and Anderson, 1986, *The American Economic Review* 76: 587-603)... only 2 could be reproduced.
- Another systematic study of the 62 papers published between 1996 and 2003 (McCullough, McGeary and Harrison, 2006, *JMCB* 38: 1093-1107) found 14 of them reproducible.

Shortcomings of this approach

- Data and code handing-in was essentially dependent of the authors' good will.
- Neither a precise data format nor a data description was requested from the authors.
- No code description was requested—but anyone who programs knows that most undocumented codes are impenetrable, even by their own authors, after 2 to 6 months—.
- No description of the way the codes were applied to the data was requested.

Public debt and (economy) growth rate

- More recently economists made headlines (of reproducible research) with a controversy on the link between public debt and growth rate (Reinhart et Rogoff 2010, *AER* 100: 573–78) ;
- Herndon, Ash and Pollin later argued that the original paper was dubious: "While using RR's working spreadsheet, we identified coding errors, selective exclusion of available data, and unconventional weighting of summary statistics" (2014, *Cambridge Journal of Economics* 38: 257–279).
- We should nevertheless credit Reinhart and Rogoff for making their codes and data (`Excel` spreadsheets!) available.

The *Stanford Exploration Project*

In 1992, Jon Claerbout et Martin Karrenbach in a communication at the *Society of Exploration Geophysics* wrote:

> *A revolution in education and technology transfer follows from the marriage of word processing and software command scripts. In this marriage an author attaches to every figure caption a pushbutton or a name tag usable to recalculate the figure from all its data, parameters, and programs. This provides a concrete definition of reproducibility in computationally oriented research. Experience at the Stanford Exploration Project shows that preparing such electronic documents is little effort beyond our customary report writing; mainly, we need to file everything in a systematic way.*

The key features of Claerbout and Karrenbach idea was later reformulated by Buckheit and Donoho (1995) who wrote:

> *An article about computational science in a scientific publication is **not** the scholarship itself, it is merely **advertising** of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.*

Stanford Exploration Project tools

SEP geophysicists analyse large data sets and make "complex" simulations of geophysical models (PDE based); they are therefore:

- used to compiled languages like `ratfor` (a `fortran` dialect) and `C`,
- using build automation with `Cake` derived from `Make`,
- writing their papers with TEX and $LATEX$.

Their key idea was to use build automation not only for binary generation but also for applying the codes to the data—generating thereby the paper's figures and tables—before compiling the `.tex` file.

The SEP has since then developed Madagascar, a powerful and "versatile" tool, directed towards geophysics, built on top of the build-automation utility SCons –a `Python` based software–.

Strong and weak features

Strong features:

- Everything (data, source codes, scripts, text) is kept in a directory arborisation making the whole work easy to archive and to distribute.
- A marked emphasis on open-source software was present right at the beginning.

Weak features:

- TEX (or $LATEX$) is not super practical for "note taking" and is a real obstacle outside of maths and physics communities.
- The whole approach is perhaps too "heavy" for daily exploratory data analysis.

Personal assessment of build-automation based RR

- If your work leads you to develop "a lot" of compiled code your are already implementing build-automation.
- Including paper generation (`.tex` file compilation) in the process is no big deal.
- This is what I use, together with a standardised language (like `C`, `C++`, `Fortran`), when I want things to last!
- If you're happy with `Python` or the `JVM` (with `Clojure`) check the d'ActivePapers concept, you won't have to deal explicitly with build-automation or directory arborisation.

A detour through Literate Programming

When Donald Knuth received in 1976 the proofs of the second volume of his *opus magnum* (*The Art of Computer Programming*), he was horrified by their appalling typographic quality; this lead him to:

1. develop $\TeX$ a typesetting system;
2. introduce the idea of literate programming and to develop `WEB`, the software making this idea usable.

With literate programming, the code and its documentation are intermingled—in order to make the code easily understandable by a human being, as opposed to a compiler—in a single `ASCII` (now `UTF-8`) file. Two outputs can be generated:

- a `.tex` file containing the "classical" printable code documentation $\rightarrow$ `Weave` ;
- a source file in `C` (originally in `Pascal`, but that can be in any language) that will be compiled into a binary $\rightarrow$ `Tangle`.

Literate programming explained by D Knuth

I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be works of literature. Hence, my title: "Literate Programming". Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do.

The practitioner of literate programming can be regarded as an essayist, whose main concern is with exposition and excellence of style. Such an author, with thesaurus in hand, chooses the names of variables carefully and explains what each variable means. He or she strives for a program that is comprehensible because its concepts have been introduced in an order that is best for human understanding, using a mixture of formal and informal methods that reinforce each other.

— Donald Knuth, Literate Programming.

Diverting literate programing: `R` and the `Sweave` function

`R` is a langage / software under a *GPL* licence, described as follows on the FAQ page:

> *R is a system for statistical computation and graphics. It consists of a language plus a run-time environment with graphics, a debugger, access to certain system functions, and the ability to run programs stored in script files.*

For programmers, `R` stems from `scheme` with a `C` like syntax:

- we write `2+2` ;
- not `(+ 2 2)`.

Sweave

- `Sweave` is an `R` function.
- `Sweave` processes text files mixing "prose" typeset with $\LaTeX$ or `HTML` with `R` code.
- `Sweave` copies verbatim the prose part of the file into a new file, runs the code and adds the results (tables, figures) in this new (`.tex` or `.html`) file.
- `Sweave` files syntax is close to the one of noweb files, the modern version of Knuth's `web`.

With Sweave we "replace" the figures and tables of a manuscript by the code that generates them.

Example

A part of a Sweave file looks like:

```
\subsection{Data summary}
The \textit{five number summary} of data set
a is:
<<summary-a>>=
summary(a)
@
We see that \textbf{saturation is absent}...
```

The true link between `Sweave` and literate programming is the common syntax bracketing code blocks in the source file:

- a block starts with `«block-name»=` ;
- and ends with `@`.

Shortcomings

- You must know `R` and LaTeX (or `HTML`).
- For me LaTeX is great to write papers and tutorials, but it is a bit too heavy for my lab-book.
- Changing from one output format (`PDF`) to another (`HTML`) requires extra tools like TeX4ht.
- True literate programming in Knuth's sense cannot be implemented—you can describe how code is applied to data but you can't develop code per se—.

Recent developments: lightweight markup languages

A major drawback of previous approaches, the necessity to write the "prose part" in LaTeX or `HTML`, has now disappeared with the development of lightweight markup languages like:

- `Markdown`
- `reStructuredText`
- `Asciidoc`
- `Org mode` (with which this talk was prepared).

With pandoc, developed by John MacFarlane (a philosopher), it is moreover easy to translate one of these languages into the others; with the pandoc extension of Markdown a beginner with one hour of practice can generate a LaTeX file bluffing an expert.

- For `R` users, `Markdown` syntax can be used:
  - with the `RMarkdown` package
  - or, even simpler, with RStudio.
- `Python` users have:
  - `Pweave`,
  - or `jupyter` notebooks.

Example (R Markdown version)

The former example with R Markdown becomes:

````
## Data summary
The _5 number summary_ of data set a is:
```{r summary-a}
summary(a)
```
We see that __saturation is absent__...
````

For comparison, the previous version (with `Sweave`) :

```
\subsection{Data summary}
The \textit{five number summary} of data set
a is:
<<summary-a>>=
summary(a)
@
We see that \textbf{saturation is absent}...
```

A quick comparison

The following tools allow anyone already familiar with `R`, `Python` or `Julia`, to be quickly productive for exploratory or interactive data analysis:

- With `jupyter` notebooks these three languages can be used *separately*, but there is no real editor in jupyter, a major drawback in my opinion.
- RMarkdown with RStudio provides a straightforward access to reproducible research with `R` and (a little bit) with `Python`; the editor is also reasonably good.
- SageMath based on `Python` 2, with `R`, `Maxima` and roughly 100 scientific libraries "under the hood" is probably the most comprehensive solution to date, *but the development stopped*.

A drawback: all these tools are "script oriented" and not really designed for code development (even without going as far as literate programming).

The mode `Org` of the editor `GNU emacs` combines the benefits of `SageMath` with the capability to implement genuine literate programming; its only drawback: you must learn `emacs`...

Version control

Keeping in line with the diversion of software development tools approach, reproducible research practitioners become often dependent on version control software:

- git is *de facto* becoming the standard tool;
- with github and gitlab, even beginners can use it.

Large data sets

When we start working on "real" data we often have to face two problems:

- The data are inhomogeneous (scalar, vectors, images, etc).
- The data require a lot of memory.

What we want to keep from text files: metadata

- Text format allows us to easily store data plus a lot of extra information...
- $\Rightarrow$ we can add to the file:
  - where the data come from;
  - when were they recorded;
  - what is their source;
  - etc.
- These information on the data are what is called metadata.
- They are essential for (reproducible) research.

Binary formats for heterogeneous data with metadata

What we need is binary formats allowing us to:

- work with large heterogeneous data;
- keep metadata with the data;
- fix the endianness once and for all.

FITS and HDF5

- The Flexible Image Transport System (FITS), created in 1981 is still maintained and regularly updated.
- The Hierarchical Data Format (HDF), developed by the National Center for Supercomputing Applications, reached its fifth version, HDF5.

Data repository

A scientist working with experimental data (as opposed to simulations) will most likely sooner or later face a problem when implementing reproducible research: how can large data sets become easily accessible by anyone? Luckily many public (and free) data repository appeared these last years:

- RunMyCode
- Zenodo (that's the one I'm using)
- The Open Science Framework
- Figshare
- DRYAD
- Exec&Share.

A "real size" example

If there is enough time, we will discuss the "supplementary material" of the manuscript *A system of interacting neurons with short term plasticity*. The former can be downloaded from:

`https://plmlab.math.cnrs.fr/xtof/interacting_neurons_with_stp`

Some references

- The Mooc "Reproducible research: Methodological principles for a transparent science"!
- *Implementing Reproducible Research*, a book edited by V Stodden, F Leisch and R Peng, that can be legally downloaded on the web ; it also discusses thoroughly *workflows* (a popular approach in Biology).
- The *Reproducibility* page of the `Madagascar` website.
- The *ReScience* journal whose aim is to publish replications of computational papers.
- *Top 10 Reasons to Not Share Your Code (and why you should anyway)* a great talk by Randy LeVeque, both fun and deep.

Conclusions

- Regardless of the type of work you have to do—large code development or "tailored scripts"—there are now tools allowing you to smoothly implement the reproducible research paradigm.
- We will now have to start a more "political" struggle so that this approach gets the recognition it deserves (in my view), for that we will need:
  - a change in editorial policies;
  - a request for reproducible research implementation from the funding agencies;
  - to make it count when research activity is evaluated.

Thanks

I want to thank:

- the IDEEV for this invitation;
- my employer, the CNRS, for letting spending a lot of time making my work reproducible even if no one is asking for it;
- the developers of the free software mentioned in that talk;
- you for listening to me.