Reproducible Publishing — Reference Poster Implementation

github.com/TheChymera/RepSeP

Horea-Ioan Ioanas[1], Markus Rudin[1]

[1]Institute for Biomedical Engineering, ETH and University of Zurich

Abstract

As scientific work becomes increasingly collaborative and automated, reproducibility becomes increasingly vital for sharing and integrating scientific results. For most researchers, however, reproducibility remains a nebulous ideal, the benefits of which are considered more theoretical and indirect, than practical and immediate. Here we showcase an open-source reference implementation of a technology stack which makes the benefits of reproducibility accessible via a reusable document template.

The prevalent and currently most accessible medium of exchange for high-level (i.e. semantic) scientific results is that of the document. As this medium (including e.g. posters and articles) is static, it encourages the creation of work which is unreproducible. We present an infrastructure which addresses this issue, without compromising content sharing standards, by automatically generating variable article elements (e.g. figures and statistics) directly from code and data.

Dependency Management

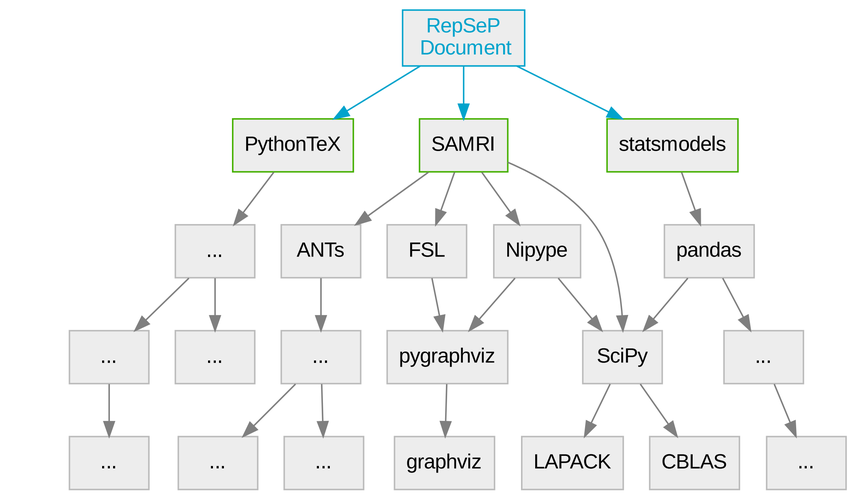

Figure 1: Small excerpt and conceptual representation of software dependencies for research articles doing even only simple neuroimaging analysis. First-level dependencies are highlighted in green.

Explicit, unambiguous declaration (e.g. conforming to the Package Manager Specification [1]) of first-level dependencies is vital for software environment reproducibility. Given such a specification, an effective package manager can automatically resolve all downstream steps (typically extending into the hundreds or thousands of packages) [2].

Technologies

Arbitrarily complex plots leveraging the full capacity of matplotlib [3], higher-level packages building on it, or other Python plotting packages, are generated live on document compilation.

The core technology of this infrastructure is provided by PythonTeX [4], which comes complete with *codeblock* dependency tracking (n.b. not to be confused with dependency management for the software itself). All figures are generated on the initial execution of a LaTeX compile script, and subsequently regenerated when either code, data, or styling dependencies are changed.

3D Plots

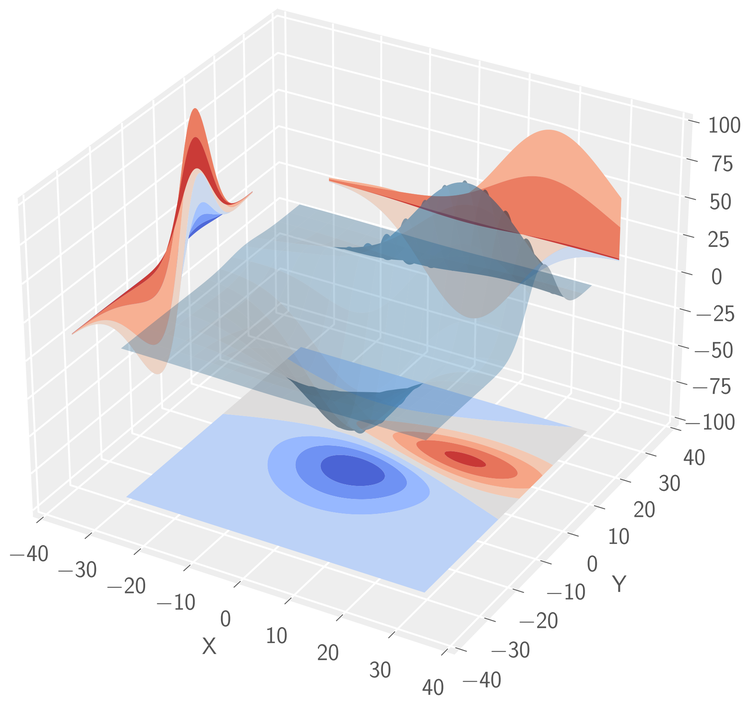

Figure 2: A 3D plot, from the matplotlib exampes [3].

The above image is dynamically generated on document compilation by inserting the following code in the TeX document source:

```
\py{pytex_fig(
        'scripts/3dplot.py',
        conf='poster/3dplot.conf', label='3dplot',
        caption='A 3D plot. Plot script from matplotlib exampes \cite{matplotlib}.',
        )}
```

Outlook

- We are looking for **testers**, to apply this reference implementation to their own work (we have used it to great satisfaction in numerous articles, including [5]).
- We are looking for potential **co-authors** for the reference article implementation, which is to be submitted to an academic journal.
- We are looking for potential **web developers** or **co-founders** to launch a platform offering RepSeP-based publishing services (yes, a journal of reproducible code-based articles!).

Why do we need this?

Why do we need code-based publishing?

- Transparency ⟶ verifiability
- Reproducibility ⟶ hackability, reusability
- Version management ⟶ sustainability

Why do we need distributed publishing?

- No external entry barrier ⟶ citizen science
- No institutional bias ⟶ free science
- *Less* publication bias ⟶ honest science

Split Violin Plots

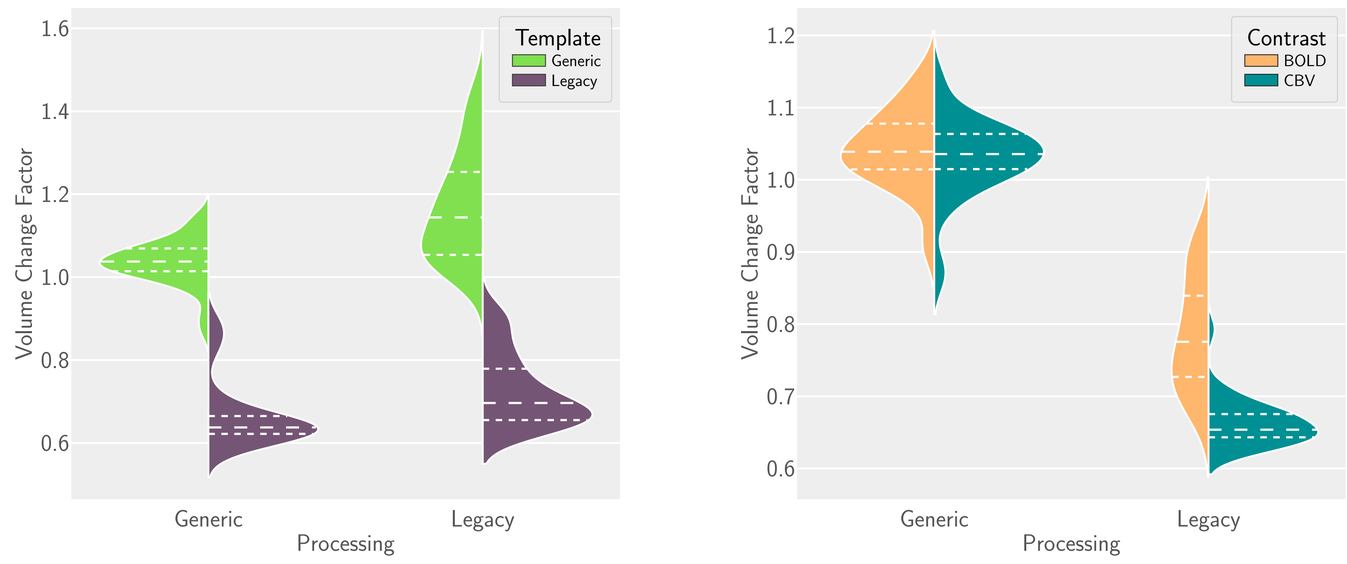

Figure 3: Violin plots highlighting both the distribution densities and quartiles in a multifactorial comparison (from [5]). The style is adapted in the source code of this document to improve quartile styling in excess of the capabilities offered by the upstream package, seaborn [6].

Custom plotting options can also be used, by distributing customization code inside the article source. For fig. 3, a patched module from the original code is distributed in `lib/categorical.py`, and preferentially imported in the script files (e.g. `scripts/violin.py`).

Statistics and Tables

Automatically computed and formatted inline statistics:

- $F_{1,268} = 10.97,\ p = 0.0011$
- Processing Factor: $F_{1,268} = 72.8,\ p = 1.07 \times 10^{-15}$
- Template Factor: $F_{1,268} = 1333,\ p = 5.13 \times 10^{-106}$
- Processing:Template Interarction: $F_{1,268} = 10.97,\ p = 0.0011$

Onset [s]	Duration [s]	Frequency [Hz]	Pulse Width [s]	Wavelength [nm]
333.05	20.0	20.0	0.005	488.0
513.05	20.0	20.0	0.005	488.0
693.05	20.0	20.0	0.005	488.0
873.05	20.0	20.0	0.005	488.0
1053.05	20.0	20.0	0.005	488.0

Table 1: BIDS [7] event file table, from [5]

Text elements can also be auto-generated from code and data, allowing inline statistics to be dynamic. Such elements can be based on single scripts (e.g. `\py{pytex_printonly('scripts/anova.py')}`), or parameterized script calls (e.g. `\py{boilerplate.fstatistic('Processing', condensed=True)}`), allowing the same model to be used and different factors to be reported in different locations.

```
\begin{table}[]
      \py{
            pytex_tab(
                  script='scripts/stim_table.py', label='sp',
                  caption='BIDS \cite{bids} event file table, from \cite{irsabi}',
                  options_pre='\\centering \\resizebox{0.9\\textwidth}{!}{',
                  data='data/JogB.tsv', options_post='}',
                  )
          }
\end{table}
```

Additionally, scripts such as the one invoked in the code block to the left allow tab or comma separated value files to be automatically read and typeset as LaTeX tables.

Manual Anchors

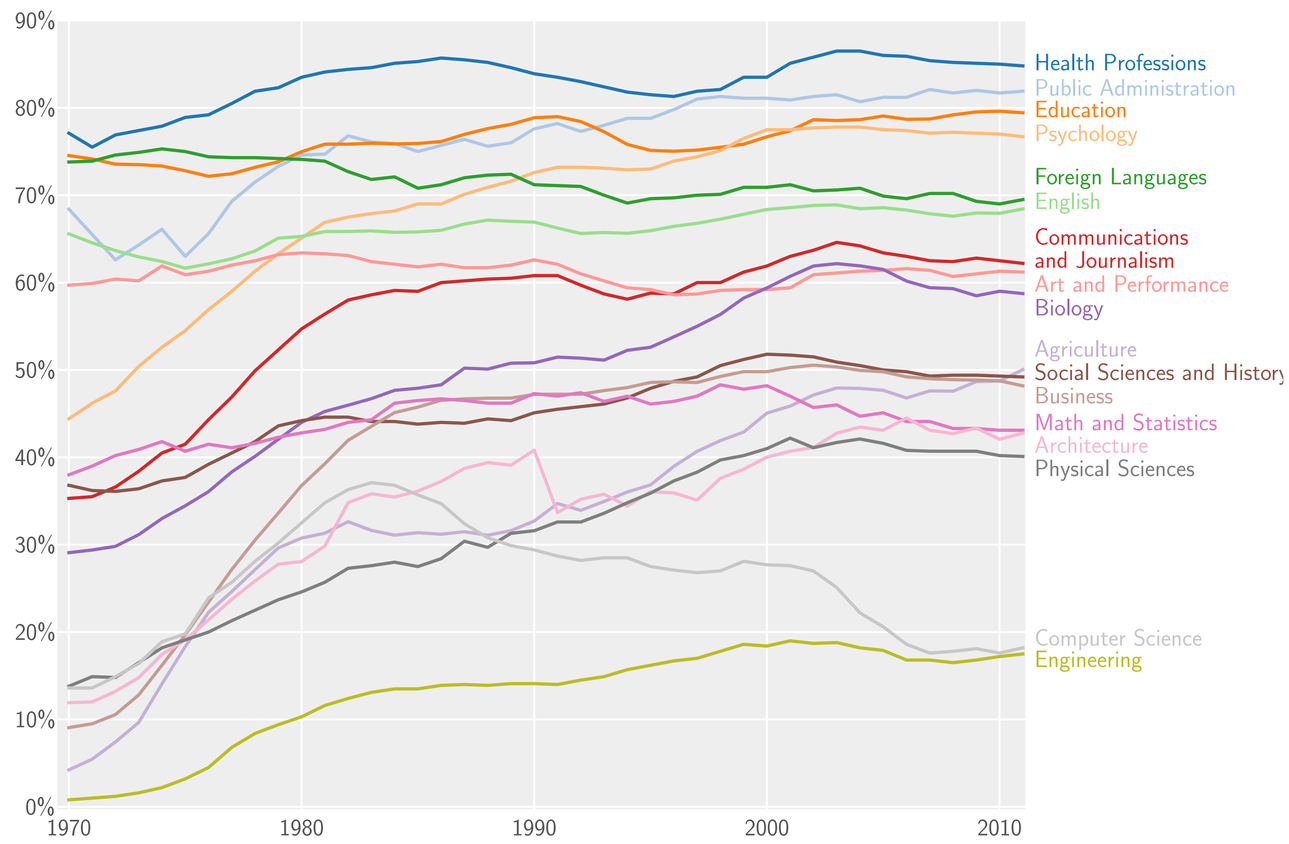

Figure 4: Percentage of Bachelor's degrees conferred to women in the U.S.A. by major (1970-2011). Plot script from matplotlib exampes [3].

The style application via hierarchical matplotlib configuration files (global, per-document, per-script — in ascending order of priority) allows the selfsame script results to be adapted to individual document types. Multiple views of the same data analysis summary (e.g. a plot) can thus rely on the same code, avoiding divergent editing. Even sensitive plot elements, such as anchors, remain stable throughout various style applications, as exemplified here.

References

[1] S. P. Bennett, C. Faulhammer, C. McCreesh, and U. Müller, "Package manager specification." online, 04 2017.

[2] H.-I. Ioanas, B. Saab, and M. Rudin, "Gentoo linux for neuroscience - a replicable, flexible, scalable, rolling-release environment that provides direct access to development software," *Research Ideas and Outcomes*, vol. 3, p. e12095, Feb. 2017.

[3] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, vol. 9, pp. 90–95, June 2007.

[4] G. M. Poore, "Pythontex: reproducible documents with latex, python, and more," *Computational Science & Discovery*, vol. 8, no. 1, p. 014010, 2015.

[5] H.-I. Ioanas, M. Marks, M. F. Yanik, and M. Rudin, "An optimized registration workflow and standard geometric space for small animal brain imaging," *bioRxiv*, 2019.

[6] M. Waskom, O. Botvinnik, D. O'Kane, P. Hobson, S. Lukauskas, D. C. Gemperline, T. Augspurger, Y. Halchenko, J. B. Cole, J. Warmenhoven, J. de Ruiter, C. Pye, S. Hoyer, J. Vanderplas, S. Villalba, G. Kunter, E. Quintero, P. Bachant, M. Martin, K. Meyer, A. Miles, Y. Ram, T. Yarkoni, M. L. Williams, C. Evans, C. Fitzgerald, Brian, C. Fonnesbeck, A. Lee, and A. Qalieh, "Seaborn: v0.8.1," Sept. 2017.

[7] K. J. Gorgolewski, T. Auer, V. D. Calhoun, R. C. Craddock, S. Das, E. P. Duff, G. Flandin, S. S. Ghosh, T. Glatard, Y. O. Halchenko, *et al.*, "The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments," *Scientific Data*, vol. 3, p. 160044, June 2016.

DOI: 10.5281/zenodo.3477675

Open Innovation in Life Sciences — 2019-10-10

ioanas@biomed.ee.ethz.ch