lpEdit: an editor to facilitate reproducible analysis via literate programming

Adam J Richards[¶*], Andrzej S. Kosinski[§], Camille Bonneaud[‡], Delphine Legrand[‖], Kouros Owzar[**]

http://www.youtube.com/watch?v=1HCeSwMirIA

Abstract—There is evidence to suggest that a surprising proportion of published experiments in science are difficult if not impossible to reproduce. The concepts of data sharing, leaving an audit trail and extensive documentation are fundamental to reproducible research, whether it is in the laboratory or as part of an analysis. In this work, we introduce a tool for documentation that aims to make analyses more reproducible in the general scientific community.

The application, lpEdit, is a cross-platform editor, written with PyQt4, that enables a broad range of scientists to carry out the analytic component of their work in a reproducible manner—through the use of literate programming. Literate programming mixes code and prose to produce a final report that reads like an article or book. lpEdit targets researchers getting started with statistics or programming, so the hurdles associated with setting up a proper pipeline are kept to a minimum and the learning burden is reduced through the use of templates and documentation. The documentation for lpEdit is centered around learning by example, and accordingly we use several increasingly involved examples to demonstrate the software's capabilities.

We first consider applications of lpEdit to process analyses mixing R and Python code with the LaTeX documentation system. Finally, we illustrate the use of lpEdit to conduct a reproducible functional analysis of high-throughput sequencing data, using the transcriptome of the butterfly species *Pieris brassicae*.

Index Terms—reproducible research, text editor, RNA-seq

Introduction

The ability to independently reproduce published works is central to the scientific paradigm. In recent years, there has been mounting concern over the number of studies that are difficult if not impossible to reproduce [Ioannidis05], [Prinz11]. The reasons underlying a lack of reproducibility in science are numerous and it happens that with regards to funding and publication preference there is an emphasis on discovery with little reward for studies that reproduce results [Russell13].

* *Corresponding author: adam.richards@stat.duke.edu*
¶ *Biostatistics & Bioinformatics, Duke University Medical Center, Durham, NC, 27710, USA and Station d'Ecologie Experimentale du CNRS, Moulis, 09200, France.*
§ *Biostatistics & Bioinformatics, Duke University Medical Center, Durham, NC, 27710, USA.*
‡ *Station d'Ecologie Experimentale du CNRS, Moulis, 09200, France and Centre for Ecology and Conservation, University of Exeter Cornwall, Penryn, UK.*
‖ *Station d'Ecologie Experimentale du CNRS, Moulis, 09200, France.*
** *Duke Cancer Institute, Duke University Medical Center, Durham, NC, 27710, USA.*

The difficulties in reproducing a study can be broadly categorized as experimental and analytic. Whether it is in the laboratory or on a computer, problems with replication can be minimized through the use of three key concepts: (1) data sharing, (2) leaving an audit trail and (3) documentation. Data sharing refers to all raw data and appropriate metadata, provided under a convenient set of standards, ideally through a free and open repository, like the Gene Expression Omnibus [Edgar02]. Laying an audit trail in the laboratory can be done through the careful use of electronic notebooks, and for code, as is already commonplace in many fields, through the use of version control systems like Git http://git-scm.com or Mercurial http://mercurial.selenic.com.

Massive data sharing efforts are underway [Butler12] and the advantages of electronic systems for documenting changes are self-evident. The third aspect, documentation, can be carried out in the laboratory with electronic notebooks easily enough. However, the analyses that go along with experiments are far more difficult to properly document, and unsurprisingly this aspect of reproducible research remains a major obstacle particularly in the life-sciences.

Apart from data sharing, leaving an audit trail and documentation, there are other important aspects of reproducible research to consider such as the over-reliance on p-values [Ioannidis08], [Gadbury12] and the use of inappropriate statistical tests. Statistical problems would be drastically easier for other scientists to identify if the original data and well-documented code were made readily available. In computer science, extensively documented code is often produced through the use of literate programming [Knuth84].

In general, literate programming is the mixing of programming code and prose to produce a final report that reads in a natural way. In this work, we differ from most of the available resources for literate programming in that our focus is on producing reports that are intended for non-programmers, yet still embracing many of the important tenets of literate programming. For those with an extensive computing background there are a number of great tools like Org-mode [Schulte12] that are available. Often, biologists, chemists and other wet-lab scientists, however, lack the time to adequately learn a complicated environment and the prospect of learning is daunting when it comes to many of the available tools.

The environment we have developed here, literate programming edit (lpEdit), is a cross-platform application that enables a broad range of scientists to carry out the analytic component of their work in a reproducible manner. This work is not intended for those already well-versed in the use of text editors and literate

programming environments, although the simplicity and ability to use either the application programming interface (API) version or a graphical user interface (GUI) version has appeal to a variety of researchers.

lpEdit: a literate programming editor

Many of the tools available for literate programming do not provide a graphical editor, which is a barrier for adoption by non-specialists. Other tools depend on a particular operating-system and only a handful of tools can switch freely between several programming languages. The motivation to build lpEdit arose because there was no apparent library/tool that fit these three criteria in a simple and intuitive way.

We have developed here an environment for literate programming, based on the model-view-controller (MVC) software architecture pattern. The only major difference from conventional realizations of MVC patterns is that instead of the user interacting directly with the controller in a non-GUI mode, we have developed a convenience class called NoGuiAnalysis for this purpose.

The GUI editor portion of lpEdit is written with PyQt4 http://www.riverbankcomputing.com/software/pyqt, which are Python bindings to the widget toolkit Qt http://qt.digia.com. For the basic editing component of the software we use the Qt port of Scintilla http://www.scintilla. org called QScintilla http://www.riverbankcomputing.com/software/qscintilla. The additional prerequisites are the Python packages for numeric computing (*NumPy*) [Oliphant07] and the ubiquitous documentation tool Sphinx http: //sphinx-doc.org.

The software is available under the GNU General Public License version 3.0 or later from http://bitbucket.org/ajrichards/reproducible-research. The accompanying documentation can be found at http://ajrichards.bitbucket.org/lpEdit/index.html.

LaTeX and reStructuredText

Perhaps the most widely used literate programming tool is Sweave [Leisch02] which embeds R code into a LaTeX document. Due to its popularity and because Sweave is now part of the R project [RCore12], the Sweave environment may be used from within lpEdit. Another notable projects that mixes R and LaTeX is knitr http://yihui.name/knitr. RStudio [RStudio] is a graphical editor that supports Sweave and knitr.

R is a standard language for statistics, but for other common computational tasks, like text processing and web-applications, it is used less frequently than scripting languages. We opted to add Python, a scripting language, because it is being increasingly used in the life-sciences [Bassi07] and because it has a clean syntax that ultimately aids transparency and reproducibility. Several well-featured literate programming tools exist for Python including PyLit http://pylit.berlios.de and like PyLit our software uses reStructuredText (reST) http://docutils.sourceforge.net/rst.html, although we additionally allow arbitrary Python code to be included in LaTeX source documents. Another powerful tool for reproducible research using Python is the IPython notebook [Perez07].

There are three types of file extensions currently permitted for use with lpEdit: the Sweave extension (`*.rnw`); a Noweb [Ramsey94] inspired syntax (`*.nw`); and the reST file extension (`*.rst`). By selecting an embedded language and a file type there are a number of different workflows available as shown in Figure 1.

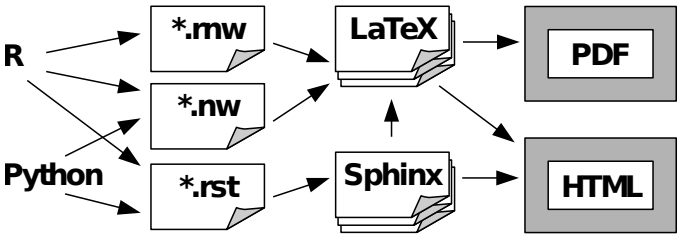

Fig. 1: Summary of the possible workflows using lpEdit. *First, a language, either R or Python is selected then it is embedded into a specific document (`*.rnw`, `*.nw` or `*.rst`). Next a LaTeX or Sphinx project is built for the document, which then allows for both HTML and PDF output formats.*

lpEdit as a library

lpEdit has a simple API, which facilitates the use of unit testing and exposes the functions of this library for those who are not in need of a text editor. In this section, we explain how to create a project and build reports using the command line, in order to illustrate the basic mechanics of lpEdit. The following example script, BasicPython.nw, is bundled with the package lpEdit. To build a project and compile it into report form only a few commands are needed.

```
1 from lpEdit import NoGuiAnalysis
2 nga = NoGuiAnalysis()
3 nga.load_file("BasicPython.nw",fileLang="python")
4 nga.build()
5 nga.compile_pdf()
6 nga.compile_html()
```

First the class is imported (line 1) from the module lpEdit and then it is instantiated (line 2). The file is then loaded and the language may be specified (line 3). The `build()` method creates a directory to contain the project in the same folder as `BasicPython.nw`. The build-step also creates a **.tex* document. This directory is what lpEdit refers to as a project and it is where both reST and LaTeX projects are managed. The `compile_pdf()` command either uses `sphinx-build` or `pdflatex`. The `compile_html()` command defaults to `sphinx-build` or `latex2pdf` depending on the project type. In most cases the default paths for `pdflatex`, `python`, `R`, and `sphinx-build` are found automatically, however, they may be customized to a user's preference. To modify these variables without the GUI, there is a configuration file corresponding to the current version of lpEdit located in the user's home directory.

```
import os
os.path.join(os.path.expanduser("~"),".lpEdit")
```

lpEdit as an editor

The primary purpose of lpEdit as a text editor was to benefit students and those who are learning to program statistical analyses. In order to make it easier on these user groups, we provide as part of lpEdit's documentation a number of examples that illustrate different statistical tests. We have left out features found in other editors or literate programming environments to make it easier to focus on report content.

Documenting by example

Like Sweave, lpEdit uses a Noweb [Ramsey94] inspired syntax. The advantages are that due to a simplified syntax, the flow of the document is only minimally interrupted by the presence of code.

Also, to reduce the learning burden on new users, we suggest they concentrate on learning LaTeX, reST and the embedded programming language of choice instead of lpEdit-specific tricks to embed plots, tables or other convenient features. For `*.rnw`, `*.nw` and `*.rst` documents, we embed code in the following way.

```
<<label=code-chunk-1>>=
print("Hello World!")
@
```

Although this particular example may not be executed in lpEdit because it is not a valid LaTeX or reST document, it illustrates that code, in this case just a print statement, is included by placing it between "« txt »=" and "@", where `txt` is any arbitrary string, preferably something informative. Note that under Sweave `txt` is a place where options may be passed. Refer to the official documentation for more comprehensive examples.

Documents written in LaTeX, or reST are written as they normally would be although now there is a way to execute embedded code within the document. There is no limit to the number of code chunks and lpEdit will execute them in sequential order, preserving the variable space. The building step is where code chunks are executed and output gathered. There is one thing to keep in mind when working with projects, and that is the idea of scope. Suppose, there are two documents `document1.rst` and `document2.rst`. If we build `document1.rst` then `document2.rst`, the results from `document1.rst` will be preserved, which is convenient when there are code chunks that take significant time to run.

Involved analyses

Analyses can take the form of long complicated pipelines, that may not reasonably be reproduced at the click of a button. This may happen if, for example, a database needs to be populated before an analysis can be carried out or perhaps there is a hardware constraint, such as the requirement of a high-performance computing infrastructure. In these cases, lpEdit or another documentation software may still be used to document details that would not normally be present in the methods section of a published manuscript. For analyses that are accompanied by substantial code and/or data, we provide the keyword INCLUDE which simply tells lpEdit that a given file is part of the current project. For example, files may be included in a `*.nw` or `*.rnw` document by

```
%INCLUDE MyFunctions.py, MyData.csv
```

where the `INCLUDE` statement is preceded by a comment indicator. For reST documents ".. " is used. At build time symbolic links are created. For a reST document, `INCLUDE` is preceded by the comment indicator. With increasingly involved analyses, the readability of documentation should not deteriorate and to this end prose may be simplified by including code and data as links. Other than `INCLUDE` and the syntax to embed code, reST and LaTeX, documents are written as they normally would be, which has the important benefit of minimizing the learning burden.

Analyzing the *Pieris brassicae* transcriptome

The analysis of high-throughput sequencing data has the earmarks of a highly involved analysis pipeline. The appeal of high-performance sequencing [Margulies05], referred to as RNA-seq, when applied to messenger RNA, is that a large number of genes

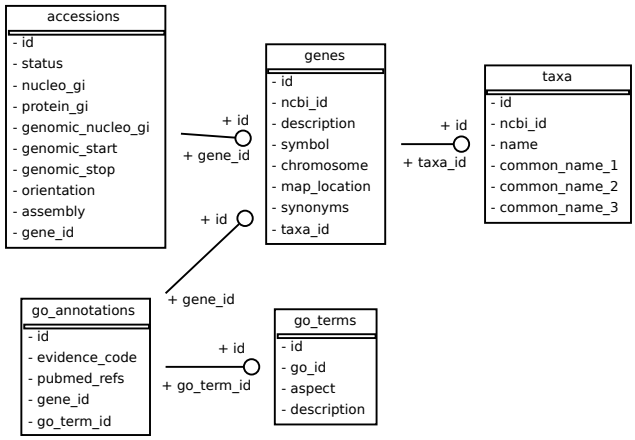

Fig. 2: Database entity diagram. *A gene-centric relational database for data available through NCBI's FTP website.*

are quickly examined in terms of both expression and genetic polymorphisms. For RNA-seq the sheer quantity of data and diversity of analysis pipelines can be overwhelming, which substantiates all the more a need for transparent analysis documentation. Here we describe the transcriptome of the cabbage butterfly (*Pieris brassicae*) [Feltwell82], a species prevalent throughout much of Europe, that is an interesting model for studying species mobility with respect to different selection pressures [Ducatez12].

cDNA library construction

Messenger RNA was extracted from the thorax, head and limbs of 12 male and female *P. brassicae* and pooled to construct a normalized cDNA library (BioS&T, Montreal, Canada). This library was subsequently sequenced using a Roche 454 pyrosequencing platform and because there is no reference genome for *P. brassicae* a *de novo* assembly pipeline was followed. The sequencing and assembly was carried out at the sequencing center Genotoul http://bioinfo.genotoul.fr and made available using the NG6 [Mariette12] software environment. Prior to assembly, the reads were filtered to ensure quality-a step that included a correction for replicate bias [Mariette11]. The assembler Newbler [Margulies05], was then used to align and order the reads into 16,889 isotigs and 11,891 isogroups.

Analysis database and environment

Because *P. brassicae* is a species without a reference genome, the assembled isotigs must be compared to species that have functional descriptions. In order to make time-efficient comparisons we first created a database using PostgreSQL http://postgresql.org (version 9.1.9). The database contained gene, accession, taxon, and functional ontology information all of which is available through the National Center for Biotechnology Infomation (NCBI) FTP site http://www.ncbi.nlm.nih.gov/Ftp. The database is detailed in Figure 2. The interaction with tables in the database was simplified through the use of the object relational mapper available as part of the python package SQLAlchemy http://www.sqlalchemy.org. The schema figure was generated using the Python package *sqlalchemy_schemadisplay* https://pypi.python.org/pypi/sqlalchemy_schemadisplay.

Functional characterization of the transcriptome

For each isotig, functional annotations were found by using the Basic Local Alignment Search Tool (BLAST) [Altschul90] via NCBI's BLAST+ command line interface [Camacho09]. Specifically, each isotig was locally aligned to every sequence in the Swiss-Prot database [UniProtConsortium12] then using our local database, accession names were mapped to gene names and corresponding functional annotations were gathered. The handling of sequence data was done using the classes and functions provided by BioPython [Cock09].

Of the nearly 17,000 isotigs that were examined, 11,846 were considered hits (E-value $\leq$ 0.04). The isotigs were then mapped to 6901 unique genes. The appropriate Gene Ontology [Ashburner00] annotations were then mapped back to the isotigs. A navigable version of the analyses and results is available as part of the online supplement http://ajrichards.bitbucket.org/lpedit-supplement. The supplement is the documentation produced using lpEdit. All scripts that were used in this analysis are provided therein and the supplement details the individual steps in this process in a way that is impossible to include as part of a manuscript methods section.

Conclusions and future work

The RNA-seq example demonstrates that involved analyses may be well- documented in a way that is interesting for those who understand the technical details of the analysis and those who do not. In the future, more languages, even compiled ones, may be integrated into the project, which is feasible because lpEdit uses the Python package `subprocess` to make arbitrary system calls. It is not our intention for lpEdit to evolve to be a replacement for already established tools, like Org-mode. Rather, it is meant as a simple tool to help newcomers with programming and statistics. With the API version of lpEdit there remains the possibility that it may be adapted as a plug-in or extension to existing text editors.

Given that the target user-base for lpEdit are those with limited computing background, there are a number of power-user features left out of the current version for the sake of a nearly 'push button approach'. Despite this restricted approach, lpEdit is free to use, fork and modify as the community would like and over time more interesting features will make it into the project without sacrificing the important idea of simplicity. Being a community-driven effort, we are open to feature requests and will adapt to the needs of the general user population.

Acknowledgments

We would like to thank Eric Pante and Michel Baguette for helpful comments and discussion. The research carried out here was partially supported by the Duke Cancer Institute (DCI). Additional support for this work was provided by the Agence Nationale de la Recherche (ANR; France) MOBIGEN [ANR- 09-PEXT-003]. The opinions, findings and recommendations expressed in this work are those of the authors and do not necessarily reflect the views of the DCI, CNRS or other affiliated organizations.

References

[Altschul90] S. F. Altschul, W Gish, W Miller, E W Myers, and D. J. Lipman. *Basic local alignment search tool*, Journal of Molecular Biology, 215:403-410, 1990.

[Ashburner00] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock. *Gene ontology: tool for the unification of biology*, Nature Genetics, 25(1):25-29, May 2000.

[Bassi07] S. Bassi. *A primer on python for life science researchers*, PLoS Computational Biology, 3(11):e199, 2007.

[Butler12] D. Butler. *Drug firm to share raw trial data*, Nature, 490(7420):322, Oct 2012.

[Camacho09] C. Camacho, G. Coulouris, V. Avagyan, N. Ma, J. Papadopoulos, K. Bealer, and T. L. Madden. *BLAST+: architecture and applications*, BMC Bioinformatics, 10:421, 2009.

[Cock09] P. J. A. Cock, T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski, and M. J. L. de Hoon. *Biopython: freely available Python tools for computational molecular biology and bioinformatics*, Bioinformatics, 25(11):1422-1423, Jun 2009.

[Ducatez12] S. Ducatez, M. Baguette, V. M. Stevens, D. Legrand, and H. Freville. *Complex interactions between paternal and maternal effects: parental experience and age at reproduction affect fecundity and offspring performance in a butterfly*, Evolution, 66(11):3558-3569, Nov 2012.

[Edgar02] R. Edgar, M Domrachev, and A E Lash. *Gene expression omnibus: NCBI gene expression and hybridization array data repository*, Nucleic Acids Research, 30(1):207-210, Jan 2002.

[Feltwell82] J. Feltwell. *Large white butterfly: The Biology, Biochemistry and Physiology of Pieris brassicae (Linnaeus)*, Springer, 1982.

[Gadbury12] G. L. Gadbury and D. B. Allison. *Inappropriate fiddling with statistical analyses to obtain a desirable p-value: tests to detect its presence in published literature*, PloS One, 7(10):e46363, 2012.

[Ioannidis05] J. P. A. Ioannidis. *Why most published research findings are false*, PLoS Medicine, 2(8):e124, Aug 2005.

[Ioannidis08] J. P. A. Ioannidis. *Effect of formal statistical significance on the credibility of observational associations*, American Journal of Epidemiology, 168(4):374-383; discussion 384-390, Aug 2008.

[Knuth84] D. E. Knuth. *Literate programming*, The Computer Journal, 27:97-111, 1984.

[Leisch02] F. Leisch. *Sweave: Dynamic generation of statistical reports using literate data analysis*, In Comp-stat 2002 - Proceedings in Computational Statistics, pages 575–580. Physica Verlag, Heidelberg, 2002.

[Margulies05] M. Margulies, M. Egholm, W. E. Altman, S. Attiya, J. S. Bader, L. A. Bemben, J. Berka, M. S. Braverman, Y-J. Chen, Z. Chen, S. B. Dewell, L. Du, J. M. Fierro, X. V. Gomes, B. C. Godwin, W. He, S. Helgesen, C. H. Ho, G. P. Irzyk, S. C. Jando, M. L. I. Alenquer, T. P. Jarvie, K. B. Jirage, J-B. Kim, J. R. Knight, J. R. Lanza, J. H. Leamon, S. M. Lefkowitz, M. Lei, J. Li, K. L. Lohman, H. Lu, V. B. Makhijani, K. E. McDade, M. P. McKenna, E. W. Myers, E. Nickerson, J. R. Nobile, R. Plant, B. P. Puc, M. T. Ronan, G. T. Roth, G. J. Sarkis, J. F. Simons, J. W. Simpson, M. Srinivasan, K. R. Tartaro, A. Tomasz, K. A. Vogt, G. A. Volkmer, S. H. Wang, Y. Wang, M. P. Weiner, P. Yu, R. F. Begley, and J. M. Rothberg. *Genome sequencing in microfabricated high-density picolitre reactors*, Nature, 437(7057):376-380, Sep 2005.

[Mariette11] J. Mariette, C. Noirot, and C. Klopp. *Assessment of replicate bias in 454 pyrosequencing and a multi-purpose read-filtering tool*, BMC Research Notes, 4:149, 2011.

[Mariette12] J. Mariette, F. Escudie, N. Allias, G. Salin, C. Noirot, S. Thomas, and C. Klopp. *NG6: Integrated next generation sequencing storage and pro cessing environment*, BMC Genomics, 13:462, 2012.

[Oliphant07] T. E. Oliphant. *Python for scientific computing*, Computing in Science & Engineering, 9(3):10-20, 2007.

[Perez07] F. Perez and B. E. Granger. *IPython: a system for interactive scientific computing*, Computing in Science & Engineering, 9(3):21-29, May 2007.

[Prinz11] F. Prinz, T. Schlange, and K. Asadullah. *Believe it or not: how much can we rely on published data on potential drug targets?*, Nature Reviews. Drug Discovery, 10(9):712, Sep 2011.

[RCore12] R Core Team. *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2012.

[RStudio] *RStudio: Integrated development environment for R*, Boston, MA.

[Ramsey94] N. Ramsey. *Literate programming simplified*, IEEE Software, 11(5):97-105, 1994.

[Russell13] J. F. Russell. *If a job is worth doing, it is worth doing twice*, Nature, 496(7443):7, Apr 2013.

[Schulte12] E. Schulte, D. Davison, T. Dye, and C. Dominik. *A multi-language computing environment for literate programming and reproducible research*, Journal of Statistical Software, 46(3):1-24, 1 2012.

[UniProtConsortium12] UniProt Consortium. *Reorganizing the protein space at the universal protein resource (UniProt)*, Nucleic Acids Research, 40(Database issue):D71-5, Jan 2012.