DSC 2001 Proceedings of the 2nd International Workshop on Distributed Statistical Computing
March 15-17, Vienna, Austria
http://www.ci.tuwien.ac.at/Conferences/DSC-2001
K. Hornik & F. Leisch (eds.) *ISSN 1609-395X*

Literate Statistical Practice

A.J. Rossini*

Abstract

Literate Statistical Practice (LSP) is an method for statistical practice which suggests that documentation and specification occur at the same time as statistical coding. It applies literate programming Knuth (1992) to the practice of statistics. We discuss 2 different approaches for LSP, one currently implemented using Emacs with Noweb and Emacs Speaks Statistics (ESS), and the other developed based on eXtensible Markup Language (XML) tools. The interference needed to change an individual's habits comes at a high cost, and good tools are critical for encouraging people to switch to a more literate style. We discuss why LSP can help, and suggest how ESS can ease the burden placed on the analyst.

KEYWORDS: Data Analysis, Programming Tools, User Interfaces, Literate Programming, Reproducible Research.

1 Introduction

In statistics, the ability to document both programming language coding as well as mathematical thought is critical to understandable, explainable, and reproducible data analysis. We will refer to the activities involved in statistical research and data analysis as *statistical practice*. These activities often involve computing, ranging from finding reference and background material to programming and computation.

Literate Statistical Practice is a programming methodology, derived from Literate Programming, which encourages the construction of documentation for data management and statistical analysis as the code for it is produced. All code and

*Departments of Biostatistics, University of Washington and Fred Hutchinson Cancer Research Center, Seattle, WA, USA. This work supported by the Center for AIDS Research, UW; and the Statistical Center for HIV/AIDS Research and Prevention, FHCRC.

documentation is interweaved into each literate document. The resulting document should provide a clear description of the paths taken during the analyses to produce the working dataset, descriptive, exploratory, and confirmatory analyses. This should describe results and lessons learned, both substantiative and for statistical practice, as well as a means to reproduce all steps, even those not used in a concise reconstruction, which were taken in the analysis.

Literate Programming Knuth (1992) describes an approach for combining code and documentation in a source file which can be woven into a description of the processes, algorithms, and results obtained from the system and tangled into the actual code. The goal of literate programming is to flexibly the notion of *pretty-printed code plus documentation* to allow for maximum documentation efficiency rather than compiler or interpreter efficiency. The two primary steps in constructing the results from the literate document are **tangling**, which produces code files, and **weaving**, which produces documentation files. The initial work was done in a language-specific form for Pascal (Web), with the intention of documenting the TEXprogram. This was later extended to C (CWeb). Further research took two forms, language-independent work such as Noweb, Nuweb, and Funnelweb, and their supporting and contributed facilities, and language-specific implementations, which provide tighter integration.

Literate Programming has been applied in the past to statistical practice. Probably the earliest work was simply to use Literate Programming for writing statistical software in compiled languages such as C or Fortran. Further extensions include the use of literate programming for the production of dataset codebooks (V. Carey, Seminar, Fall 1993, Harvard). RevWeb, work on reviveable analyses Lang and Wolf (1997–2001), relates to the work on reproducible research Schwab et al. (1996) as done in a non-literate fashion by (Buckheit and Donoho, 1995), and discussed by (deLeeuw, 1996). However, much of this work was intended for reproduction rather than creation. In 1997, the creation of Emacs modes for Noweb support lead to the combination of ESS with Noweb, to provide a tool for allowing efficient interactive analysis. Mark Lunt refactored this code during 1999 to construct a better noweb interface and integrate this tightly with ESS, no longer requiring Noweb to run analyses. Current ideas include the development of tools which clean up the user interface.

Other programming methodologies do exist and describe different styles to producing code. Some of the more popular approaches include structured programming (Niklaus Wirth), aspect programming, and extreme programming. Structured programming is the approach of hiding tasks within reusable functions and subroutines. Extreme programming is a novel philosophy, which is realized in the use of pairs-programming (2 programmers per workstation), writing test-cases prior to code, and the use of disposable CRC (class, responsibility, and collaborator) cards for constructing the design. Aspect programming is the approach to object-oriented programming which emphasizes reflection and the modularization of cross-cutting concerns through the use of a new software module level, the *aspect*. It is difficult to see how these approaches can be applied to statistical practice, though it is possible that the extreme programming approach might work.

Tools are critical to literate statistical practice, and should be minimally intru-

sive to the statistician. One of the hardest tasks known is to convince a competent computer user to switch from a preferred computing environment to a new one. Users are quite hesitant, usually with good reason, to learn a new ways. This can be seen in the common *editor wars*, which are pointless attempts to convince people to switch to tools when they don't want to.

Integrated Development Environments (IDEs) combine features and tools in a single interface in an attempt to increase programmer productivity and efficiency. The increased speed in commercial software development time over the last decade can be partially attributed to the use of IDEs and similar Rapid Application Development (RAD) tools and methodologies. In the field of statistics, programming is an important skill which can be augmented and enhanced by the right tools and environment. This is especially true with the rise of computational analysis tools such as resampling methods (including bootstrap, jackknife approaches) as well as Markov Chain Monte Carlo (MCMC) sampling (Gibbs, Metropolis–Hastings algorithms).

With Literate Statistical Analysis, we primarily are interested in describing and documenting statistical practice, that is, the tasks and actions which make up the daily activity of a statistician, as well as provide a repository for results from the analysis. This is not the only application of Literate Programming to statistical practice; past work has included documentation of methods Lumley (1998) as well as report generation, referring to the process of reproducing analyses on a possibly changing data set.

This paper will focus on the use of literate programming techniques for daily work, which means that we will examine tools for removing the burden of the literacy tools from the statistical practitioner as well as focus on language-neutral tools. In the field of statistics, different analytic tools are targeted for particular styles of statistical analysis. This suggests that efficient work habits can require switching between data analysis tools.

The next section of this paper discusses applications of Noweb, focusing on extensions to Emacs which enhance Literate Statistical Practice. Approaches for fully XML-coded Literate Programming are discussed. We conclude with possible future developments and needs.

2 Noweb-based Approaches

Noweb Ramsey (1994) was probably the first literate programming tool employed for statistical practice. This is a tool for language-independent literate programming which focuses on building files, indices, using HTML or LaTeXfor pretty-printing. Noweb is quite suitable for data analysis, as seen by the many approaches which have been based on it. The primary criticism for Noweb are the particular tools available for automating document production as well as for generating documents for viewing on paper, or through electronic viewers using PDF or HTML results. Few people like manual construction.

Noweb is a document processing tool which takes a source file and produce both documentation files, which LaTeX or an HTML con verter can take to produce

human readable documents, and code files, which can be fed into a compiler or interpreter for constructing an executable object. Noweb views a literate program as being formed from chunks, which are sections of text. These chunks are categorized as documentation or code chunks; documentation chunks are pertinent to nearby code chunks, while code chunks form named, indexable, and reusable components which can be embedded in other code chunks to form sections of computer programs. The primary advantage is that code can be ordered to enhance presentation and doesn't have to be located near parts that it will lie next to in the final source file.

Applications of Noweb include using it for documenting compiled programs, combining it with statistical packages, or using the file format to provide input to statistics packages. The first can be thought of as just a programming application, without any particular statistical focus. The second approach has been realized with RevWeb Lang and Wolf (1997–2001), which uses Noweb and interfaces with S-PLUS and R. This application allows for in-lining of documentation using Noweb, and focuses on the interface with the S statistical language through the S-PLUSand R implementations. Recent extensions to RevWeb use R's Tcl/TK interface as a GUI. The third approach is to use the file format but necessarily use the Noweb program for all tasks. This approach has been implemented using Emacs with Noweb-mode to monitor the location of the cursor in the file and use either ESS as the editing mode for statistical coding and processing, or a TeX or SGML mode for documentation when editing in the documentation sections.

2.1 ESS and Noweb

ESS currently supports a number of interactive statistical programs as well as a few interpreted statistical languages, including the S family of languages (recent versions of S, S-PLUS, and R), SAS, STATA and XLISPSTAT (including extensions, ARC and ViSta). There are various levels of support depending on the capabilities of the program and the needs of users. Because ESS builds on the extremely powerful Emacs editing capabilities and the Emacs ability to communicate directly with a running process, ESS provides very powerful and uniform interaction with the statistical programs and languages.

Emacs Speaks Statistics (ESS) provides a single keyboard interface for a family of statistical computing tasks. The primary user tasks for which ESS is optimized are statistical coding and interactive data analysis. Statistical coding is the writing of computer code for data analysis. This code might be in a compilable language, such as C or Fortran, or it might be in an interpreted language such as S-PLUS, SAS, R, XLISPSTAT, Perl or Python. The task of entering commands for interactive data analysis is similar. In either case, text is written in a computer language and sent to a computer program for compilation or interpretation. The primary difference is that the results of a small set of commands are of critical interest for review in the analysis phase, but the results of all commands are of interest in the coding phase. Both of these tasks can be present at the same time, for example in the use of compiled Fortran code for optimization, which is being called from an interpreted language, such as S-PLUS, R, or XLISPSTAT, containing the objective function to optimize.

In our work on ESS, we choose to focus on the editing interface, and work towards interfacing the editor with the statistics package, using the Noweb file format to determine whether to use a code or documentation editing mode. This allows the interface to remain language neutral, by detaching it from the programs. The ESS-Noweb interface extends ESS by

- providing a coding environment enhanced for the production of noweb documents
- adding options for submitting either a code chunk (including any or all embedded code chunks) to the running statistical process
- adding options to submit a thread of chunks to the running statistical process

The form of documentation can be satisfied through the use of LaTeX, SGML, HTML, or XML markup, which can then be processed into a human readable document.

3 XML

The Extensible Markup Language (XML), is an approach for marking up context, and is primarily used as a WWW-based data representation standard for generalized documents. It was based on Standardized General Markup Language (SGML), with the intention of being similar but more basic. XML gets converted by using the eXtensible Stylesheet Language (XSL). A program which applies XSL to XML to get a realization of a document is referred to as an XSLT (XSL Translator). The primary specification for an XML document is a Document-Type Definition (DTD) or similar schema language, for example XML-Schema. One of the important justifications for considering XML is the existence and rapid development of XML parsers and translators for most common and not-so-common computing environments.

There are a number of basic approaches for using XML for language-independent Literate Statistical Practice. The first is to consider XML as the documentation markup language, as suggested in the section on Noweb. The rest are to consider XML as the primary markup language for the entire literate document. In this approach, one can make the document, rather than just the documentation language. This latter approach would technically be referred to as an XML application for literate programming. In this, the literate programming component can be the entire application (as discussed in Section 3.1) or just a modular component (see Section 3.2).

SWEB, (Sperberg-McQueen, 1993–1996), was an early approach for Literate Programming, using SGML. It has a suggested tag set, but there is no real implementation of a translator. SGML is extensible and can include a tag set similar to Noweb, as demonstrated by SWEB. However, tools for handling SGML documents are difficult and complex to develop. This has inhibited the use of SGML, and encouraged the development of XML, which is a simplified form which allows for markup.

Current language-neutral approaches for this include ESS and the XML Authoring Environment for Emacs Kinnucan (2001), which uses XSLT for producing the final form of documentation. Some work is currently needed to construct, in general, the code files. Another approach is SNAKE, *Statistics Needs Another Krazy Environment*, currently under development. This latter is written in Python, and uses Narval and Piper, a python-based peer-to-peer control system, for constructing the XML document which contains the components for the literate document, as well as interfaces to statistical software systems.

Language-specific approaches are currently under development by the Omegahat project, and include a roundtrip environment based on embedding R within XALAN, an XSLT processor, and output from that. This should be easily extendible to any language with Omegahat support and extensions, including Omegahat, S, R, and LispStat. Initial work seems to be targeted at reproducible research and teaching, but probably will be extendible to general statistical practice.

3.1 LPML

There are a number of projects looking at approaches for language-neutral literate programming in XML. One such approach, LPML, (Roberts, 1998–2000) takes a literate programming presentation which is coded in markup language. We use their description for the tag-set used.

- `<litprog>` This is simply the root element. Takes no attributes.
- `<format name="NAME">` The format element defines a named format. Each item may then specify which format it wants to use to generate its page. If no other format is specified, then the 'default' format is used.
- `<object name="NAME" language="LANGUAGE" item="START" variant="VARIANT"/>` The object element defines an output file. The variant feature will allow selection of certain pieces and rejection of others when tangling multiple objects from a single presentation. If a piece is shared by two variants, then it won't have a variant tag.
- `<item name="NAME" label="LABEL" format="FORMAT">` Each main item defines a page of documentation; sub-items may follow each main item. A sub-item is defined as an item whose name has a period in it The documentation and code for a sub-item is appended to its parent's page under an italicized heading, and a link anchor is created using the post-period part of its name, so that in the above example, the page `"tangle.html"` will have a link anchor `<a name="tangle_one">`. The list of items is available in the `[##itemlist##]` tag; sub-items are in a second level of list. Each item's list heading contains a link to its documentation page and this effectively creates a table of contents.
- `<piece variant="VARIANT" add-to="ADDTO">` Items define the documentation, and pieces define the code output. Each piece must be embedded in an item. If a piece has a variant attribute, then it will only be included in objects

for which it matches the variant spec. It will be filtered out of others. For finer control over variants, you can also embed a `<variant>` tag into a piece. The primary application for the variant system is documentation of porting projects across OS or languages. Both variants can thus be tangled from the same presentation, and yet still share patterns

The add-to attribute is optional; if present, it causes the piece to be added, not to the currently active item, but to the item named, similar to the Noweb feature.

- `<variant name="VARIANT">`

  When used within a ¡piece¿, makes a part of the piece subject to suppression. Useful for operating system or language-specific code.

- `<insert name="NAME"/>`

  Used for tangling together pieces of code, as done in Noweb and other literate programming systems, to construct compilable or interpretable files.

3.2 LSPML and Modular DTDs

The alternative approach to constructing a single DTD, is to extend a modular DTD with markup tags for literate programming. For this approach, we consider the Noweb and Nuweb approaches of code chunks and scraps, while letting XHTML, DocBook, or similar text-style DTDs, provide the overall document and code documentation structure. In this approach, we restrict ourselves to 3 additional tags:

- `<lspml:codechunk name="" href="" variant="" file=""/>` `Code</lspml:codechunk>`

  For specifying code to be tangled. Variant allows for setting conditional language, operating system, or similar computing-environment settings, based on the results desired.

- `<lspml:insertCodeChunk name="" href=""/>`

  For specification of recursive tangling.

- `<lspml:index name="" href="">Code or Docs to index</lspml:index>`

  Specifically for hyperlinks and indexing of the program-specific code and documentation.

One critical decision is whether to have code as character data (CDATA) or parsed character data (PCDATA) as specified above. One allows for the use of `<` as a character, and the other insists on `&lt` as the representation; this is true for all special XML entities used for denoting markup. This is primarily of importance for document editing, since the translator will not care how it ends up producing the final code file. Current work suggests the use of CDATA for readability, but it will be simple to have editors and editing modes handle this appropriately with PCDATA. Another critical issue is the use of specific indexing tags. While we provide an example above, there are XML indexing systems and tools which provide

the needed resources, including RDF (Resource Definition Format) and RSS (Real Simple Syndication) which allow for the construction of indices and summaries from the XML document. Generally, it is better to let modular components and thier corresponding tools handle tasks as needed.

4 Literary Styles

There are many literary styles that are possible. The style reflects a combination of how the statistical practitioner views the problem as well as the message to be conveyed to the target audience. While literate programming assumes a minimal level of programming competence by the reader, literate statistical practice has many possible targets, including other statisticians, students actively learning the desired material, as well as scientists with minimal statistical background. We present 2 styles which can be considered examples of use.

The first style will be referred to as the *Consultant's Style*. This style uses the LaTeXarticle format, sectioning the various components of a short-term consulting project. The sections are ordered as such

1. introduction to the problem,
2. planned approach,
3. data management,
4. descriptive statistics,
5. inferential statistics,
6. conclusions and lessons for the analyst to remember, including discussion, description, and bibliography of methods employed and the practical issues (coding, data handling) associated with the methods,
7. the concluding report for the consulting client.

Appendices might contain the dataset, the codebook, and links to or copies of client-supplied background material, if not incorporated in the introduction. The general tone is that of a scholarly article.

The second style is referred to as the institutional memory approach. This style primarily consists of documenting, in a program of research, how mathematics is implemented. Often, the simplest appraoch to describing mathematics is not the simplest approach for computing the same mathematics. This becomes an issue when a researcher starts working with new colleagues on an ongoing project; for example, new students of a professor who will be working in an area that numerous others have done research in. This approach documents both intelligence as well as past folly in an attempt to minimize repeated mistakes. This document style, as implemented, uses the LaTeXreport style, with the initial chapter reading like a journal article, and subsequent chapters containing individual work by collaborating researchers working on pieces of the research program. One prime example of this

is the work on Reproducible Research by the Stanford Exploratory Project group in the Stanford Geophysics department.

There are many other possible styles for literate statistical practice, and within a literary style, there is much flexibility in terms of presentation. However, common sense should suggest that content such as figures and plots should have the corresponding code-chunk accessible if not actually nearby, and that while data management can be moved to an appendix, the actual code for reconstructing the dataset from a copy of the original is very critical for documenting the analysis process.

5 Remarks

Literate Statistical Practice has a number of obvious benefits. First, the results are contained in a thorough and presentable document of the thought process involved. In addition, there is an obvious approach for reproducing the results. Tools are currently under development to remove the burden on the statistical practitioner.

However, there are a number of problems with LSP. The first and foremost is the need to change one's development tools. While this seems to have the least philosophical effect, it is a high-impact practical problem, necessitating a slow-down in productivity while new tools and practices are developed into habits. Another severe cost is that LSA front-loads the need to think; unless sufficient consideration of the approach is developed at the beginning, prior to actual coding or data analysis, large delays can be created as useless code is constructed. There is an argument that enforcing discipline by penalizing the lack of discipline is a positive learning experience, but it can be very discouraging to the practitioner who is experimenting with the approach. The wasted time when this happens is definitely not positive feedback. So the gains from literate statistical analysis are not free, in that they require a great deal of discipline in the construction of the document.

LSP forms a component which assists in the production of carefully considered data analysis. Experience by the author has suggested that LSP is quite useful and efficient for documenting statistical consulting as well as for collaboration revolving around a single study, with perhaps multiple datasets. The key unit of documentation, in the author's experience, is a single or closely related group of scientific questions. The use of Noweb-derived approaches, using ESS and AUC-TeX with targeted Makefiles, has resulted in a platform for efficient production.

References

Jonathan B. Buckheit and David L. Donoho. Wavelab and reproducible research. Technical report, Stanford University Statistics Department, 1995.

Jan deLeeuw. Reproducible research: The bottom line. Technical Report 301, UCLA Statistics Department, 1996. Available from J. Deleeuw, deleeuw@stat.ucla.edu; recently added to the UCLA tech report list, hence the difference between the number and the year written.

Paul Kinnucan. Xae: Xml authoring environment for emacs. WWW, 2001.

Donald E. Knuth. *Literate Programming*. Number 27 in CSLI Lecture Notes. Center for the Study of Language and Information, 1992.

Lorenz Lang and Hans Peter Wolf. *The REVWEB Manual for Splus in Windows*. Uni. Bielefeld, Germany, 1997–2001. http://www.wiwi.uni-bielefeld.de/StatCompSci/software/revweb/revweb.html.

Thomas Lumley. Survival analysis in xlisp-stat: a semiliterate program. *Journal of Statistical Software*, 3(2), 1998.

Norman Ramsey. Literate programming simplified. *IEEE Software*, 11(5):97–105, September 1994.

Michael Roberts. WWW, 1998–2000. http://www.vivtek.com/lpml/language.html.

M. Schwab, M. Karrenbach, and Jon Claerbout. Making scientific computations reproducible. Technical report, Stanford University, Stanford Exploration Project, 1996.

C. M. Sperberg-McQueen. Sweb: an sgml tag set for literate programming. WWW, 1993–1996. http://www.uic.edu/~cmsmcq/tech/sweb/sweb.html.