Literate Programming using noweb

Terry Therneau

August 28, 2024

1 Introduction

> Let us change or 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 *humans* what we want the computer to do. (Donald E. Knuth, 1984).

This is the purpose of literate programming (LP for short). It reverses the ususal notion of writing a computer program with a few included comments, to one of writing a document with some embedded code. The primary organization of the document can then revolve around explaining the algorithm and logic of the code. Many different tools have been created for literate programming, and most have roots in the WEB system created by D. Knuth [2]. Some of these have been language specific, e.g. CWEB or PascalWeb; this article focuses on Norman Ramsey's *noweb*, an simple LP tool that is language agnostic [3, 1].

Most R users will already be familiar with the basic structure of a noweb document, since the noweb system was the inspiration or Sweave.

2 Why use LP for S

Documentation of code is often an afterthought, particularly in a high level language like S. Indeed, I have one colleague who proclaims his work to be "self-documenting code" and eschews comment lines. The counter argument is proven any time we look at someone else's code (what *are* they doing?), and in fact by looking at any of our own code after a lapse of time. When we write code the thought process is from an overall structure to algorithm to R function to code; the result is clear and simple *as long as* that overall structure remains in our thought, but reconstructing that milleau is not easy given the code alone. For a larger project like a package, documentation is even more relevant. When I make a change to the survival package I ususally find that the revision is 2/3 increased commentary and only 1/3 modified code, and a major portion of the time was spent puzzling out details that once were obvious.

My use of LP methods was motivated by the coxme package. This is the most mathematically challenging part of the surival suite, and due to the need

to use sparse matrix methods it is algorithmically complex as well. It was one of the better programming decisions I've made.

The old adage "more haste, less speed" holds for R code in general, but for packages and complex algorithms in particular. Many of you will have had the experience of puzzling over a coding or mathematics issue, then finally going to a colleage for advice. Then, while explaining the problem to them a solution suddenly becomes clear. The act of explaining was the key. In the same way, writing down and organizing the program logic within a document will get one to the endpoint of a working and reliable program faster.

The literate programming literature contains more and better stated arguments for the benefit of this approach.

3 Coding

Like an Sweave file, the noweb file consists of interleaved text and code chunks, the format is nearly identical. Here is the first section of the coxme code (after the document's header and and an introduction).

```
\section{Main program}
The [[coxme]] code starts with a fairly standard argument list.
<<coxme>>=
coxme <- function(formula,  data,
        weights, subset, na.action, init,
        control, ties= c("efron", "breslow"),
        varlist, vfixed, vinit, sparse=c(50,.02),
        x=FALSE, y=TRUE,
        refine.n=0, random, fixed, variance,  ...) {

    time0 <- proc.time()    #debugging line
    ties <- match.arg(ties)
    Call <- match.call()

    <<process-standard-arguments>>
    <<decompose-formula>>
    <<build-control-structures>>
    <<call-computation-routine>>
    <<finish-up>>
}

The arguments to the function are described below,
omitting those that are identical to the \texttt{coxph} function.
\begin{description}
\item ...
```

The typeset code looks like this:

```
⟨coxme⟩=
 coxme <- function(formula,  data,
         weights, subset, na.action, init,
         control, ties= c("efron", "breslow"),
         varlist, vfixed, vinit, sparse=c(50,.02),
         x=FALSE, y=TRUE,
         refine.n=0, random, fixed, variance,  ...) {

     time0 <- proc.time()    #debugging line
     ties <- match.arg(ties)
     Call <- match.call()

     ⟨process-standard-arguments⟩
     ⟨decompose-formula⟩
     ⟨build-control-structures⟩
     ⟨call-computation-routine⟩
     ⟨finish-up⟩
 }
```

In the final pdf document each of the chunks is hyperlinked to any prior or later instances of that chunk name.

The structure of a noweb document is very similar to Sweave. The basic rules are

1. Code sections begin with the name of the chunk surrounded by angle brackets: `<<chunk-name>>=`; text chunks begin with an ampersand `@`. The primary difference with Sweave is that the name is required — it is the key to organizing the code — whereas in Sweave it is optional and usually omitted. There are no options within the brackets.

2. Code chunks can refer to other chunks by including their names in angle brackets *without* the trailing = sign. These chunks can refer to others, which refer to others, etc. In the created code the indentation of a reference is obeyed. For instance in the above example the reference to "`<<finish-up>>`" is indented four spaces; when the definition of finish-up is plugged in that portion as a whole will be moved over 4 spaces. When the `<<finish-up>>` chunk is defined later in the document it starts at the left margin. As an author what this means is that you don't have to remember the indentation from several pages ago, and the standard emacs indentation rules for R code work in each chunk de-novo.

3. Code chunks can be in any order.

4. The construct `[[x<- 3]]` will cause the text in the interior of the brackets to be set in the same font as a code chunk.

5. Include `\usepackage{noweb}` in the latex document. It in turn makes use of the fancyvrb and hyperref packages.

6. One can use either .Rnw or .nw as the suffix on source code. If the first is used then emacs will automatically set the default code mode to S, but is not as willing to recognize C code chunks. If the .nw suffix is used *and* you have a proper noweb mode installed [1], the emacs menu bar (noweb:modes) can be used to set the default mode for the entire file or for a chunk.

The ability to refer to a chunk of code but then to defer its definition until later is an important feature. As in writing a textbook, it allows the author to concentrate on presenting the material in *idea* order rather than code order.

To create the tex document, use the R command `noweave(file)` where "file" is the name of the .Rnw source. It is necessary to have a copy of `noweb.sty` available before running the latex or pdflatex command, a copy can be found in the inst directory of the distribution. Optional arguments to noweave are

out The name of the output file. The default is the name of the input file, with the file extension replaced by ".tex".

indent The number of spaces to indent code from the left margin. The default value is 1.

To extract a code chunk use the `notangle` command in R. Arguments are

file the name of the .Rnw source file.

target the name of the chunk to extract. By default, notangle will extract the chunk named "*", which is usually a dummy at the beginning of the file that names all the top level chunks. If this is not found the first named chunk is extracted. During extraction any included chunks are pulled in and properly indented.

out The name of the output file. By default this will be the input filename with the file extension replaced by ".R".

For Unix users the stand alone noweb package is an alternative route. I was not able to find a simple installation process for MacIntosh, and no version of the package at all for Windows. For R users the package option is simpler, although the standalone package has a longer list of options. Many of these, however, are concerned with creating cross-references in the printed text, which is mostly obviated by the hyperlinks.

The noweave program will create a tex file with the exact same number of lines as the input, which is a help when tracking back any latex error messages — almost. The R version fails at this if the `@` that ends a code chunk has other text after the ampersand on the same line. Most coders don't do this so it is rarely an issue.

[1] The phrase *proper noweb mode* requires some explanation. The classic nw mode for emacs has not been updated for years, and does not work properly in emacs 22 or higher. However, in versions 2 and earlier of ESS Rnw mode was built on top of nw mode, and ESS included a `noweb.el` file that was updated to work with later emacs versions. If you are using ESS 2.15, say, then noweb mode works fine. The newer ESS version 12 created Rnw mode from scratch, does not include a noweb file, and emacs reverts to the old, non-working nw code.

4 Incorporation into R

In my own packages, noweb source files are placed in a `noweb` directory. My own style, purely personal, is to have source code files that are fairly small, 2 to 10 pages, because I find them easier to edit. I then include a Makefile: below is an example for a project with one C program and several R functions.

⟨*makefile*⟩=

```
PARTS = main.Rnw \
        basic.Rnw \
        build.Rnw \
        formula.Rnw \
        varfun.Rnw  varfun2.Rnw \
        fit.Rnw \
        ranef.Rnw \
        lmekin.Rnw \
        bdsmatrix.Rnw

all: fun doc
fun: ../R/all.R  ../src/bds.c
doc: ../inst/doc/sourcecode.pdf
R = R --vanilla --slave

../R/all.R: all.nw
        echo "require(noweb); notangle('all.nw')" > $(R)
        echo "# Automatically created from all.nw by noweb" > temp
        cat temp all.R > $@

../src/bds.c: all.nw
        echo "/* Automatically created from all.nw by noweb */" > temp
        echo "require(noweb): notangle('all.nw', target='bds', out='bds.c')" > $(R)$
        cat temp bds.c > $@

../inst/doc/sourcecode.pdf: all.nw
        echo "require(noweb); noweave('all.nw')" > $(R)
        texi2dvi --pdf all.tex
        mv all.pdf $@

all.nw: $(PARTS)
        cat ($PARTS) > all.nw

clean:
        -rm all.nw all.log all.aux all.toc all.tex all.pdf
        -rm temp all.R bds.c
```

The first file "main" contains the definition of `<<*>>=`

```
<<*>>=
```

```
<<bdsmatrix>>
<<ghol>>
<<print.bdsmatrix>>
...
```

listing the top level chuncks defined in each of my sub-files, which in turn contain all the R function definitions. For a more sophisticated Makefile that creates each function as a separate .R file look at the source code for the coxme package on Rforge.

One can add a configure script to the top level directory of the package to cause this Makefile to be run automatically when the package is built. See the source code for the coxme library for an example which had input from several CRAN gurus. I found out that it is very hard to write a Makefile that works across all the platforms that R runs on, and this one is not yet perfected for that task — though it does work on the Rforge servers. When submitting to CRAN my current strategy is to run `make all` locally to create the documentation and functions from the noweb files, and *not* include a `configure` file. I then do a standard submission process: `R CMD build` to make the tar.gz file for submission, `R CMD check` to check it thoroughly, and then submit the tar file.

5 Documentation

This document is written in noweb, and found in the vignettes directory of my source code. Using a noweb file as a vignette is very unusual — this may be the only case that ever arises — since the goal of noweb is to document source code and the goal of vignettes is to document usage of the function. We made use of the vignetteEngines facility available in R version 3 in order to use noweb instead of Sweave as the default engine for the document. The noweb function itself is written (not surprisingly) in noweb, and the pdf form of the code can be found in the inst/doc directory.

References

[1] Johnson, Andrew L. and Johnson, Brad C. (1997). Literate Programming using noweb, *Linux Journal*, 42:64–69.

[2] Donald Knuth (1984). Literate Programming. *The Computer Journal* (British Programming Society), 27(2):97–111.

[3] Norman Ramsay (1994). Literate programming simplified. *IEEE Software*, 11(5):97–105.