Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. http://www.jstatsoft.org/

SASweave: Literate Programming using SAS

Russell V. Lenth
The University of Iowa

Søren Højsgaard
Aarhus University

Abstract

SASweave is a collection of awk and shell scripts that allow you to embed SAS code into a LaTeX document, and automatically incorporate the results as well. It is patterned after Sweave (Leisch 2002), which does the same thing for code written in S (or R). In fact, a document may contain both SASweave and Sweave markup. Besides the convenience of being able to easily incorporate SAS examples in a document, SASweave facilitates the concept of "literate programming": having code, documentation, and results packaged together. Among other things, this helps to ensure that the SAS output in the document is in sync with the code.

Keywords: sasweave, sweave, literate programming.

1. Introduction

SASweave is a collection of scripts that provide a similar capability for SAS that Sweave does for S or R (see Leisch 2002). That is, SASweave provides the ability to embed SAS code into a LaTeX document. By processing the document with SASweave, the code is executed and the results are included in the document. This provides a "literate programming" capability (Knuth 1992) for SAS, whereby code, output (including graphics), and documentation are all kept together, and where these elements are guaranteed to be in sync.

For those unfamiliar with literate programming and Sweave, Exhibit 1 shows just how easy this is (assuming you are already comfortable with LaTeX). The exhibit displays a SASweave source file named `demo.SAStex`. Note that it is for all practical purposes a LaTeX source file; however, it includes two `SAScode` environments that each contain SAS statements; these are called "code chunks.". (The portions that are not code chunks are called "text chunks.") The first code chunk produces printed output, and the second one produces a graph. The `\SASweaveOpts` macro in the preamble, as well as the second `SAScode` environment, specify options for how to format the results. (The data set used in this example is one of the standard data sets provided in the `sashelp` library; so it should run correctly as-is on your

demo.SAStex

```
\documentclass{article}
\usepackage{mathpazo}

\title{SASweave Demo}
\author{Russ Lenth}

\SASweaveOpts{outputsize=\footnotesize}

\begin{document}
\maketitle

This illustrates how to use \verb"SASweave" to integrate SAS code and output
in a \LaTeX{} document.
\begin{SAScode}            %%% Code chunk 1
proc univariate data = sashelp.shoes;
  var sales;
  ods select moments;
\end{SAScode}

We can also easily include graphics\ldots
\begin{SAScode}{fig=TRUE}  %%% Code chunk 2
proc gplot data=sashelp.shoes;
  plot returns * sales;
\end{SAScode}

\end{document}
```

Figure 1: Input file for a simple SASweave example.

SAS installation.)

When we run SASweave on `demo.SAStex` in Exhibit 1, it runs the SAS code, gathers the output, integrates it into a `.tex` file with the other LaTeX markup, runs `pdflatex`, and produces the document `demo.pdf` displayed (with margins cropped) in Exhibit 2. Note that the SAS code for each chunk is displayed, followed by its output in a different font. The second code chunk produces no printed output, so we see only the resulting graph.

This example illustrates most of what is needed to use SASweave effectively. There are, however, a number of options (see Section 2) that allow you to do things like exclude the listing of code or the output, change the way it is displayed, or re-use chunks of code.

SASweave (and Sweave) actually provide two different but related ways to process a source document: weaving and tangling. In weaving, the code, output, and documentation are all packaged together into a `.tex` file. In tangling, the SAS code is simply extracted from the source document and saved in a `.sas` file, thereby creating a production version of the code.

The implementation of SASweave documented here is inspired by an earlier version by Højsgaard (2006), which, like Sweave, was written in R. Both the old and the new SASweave provide for incorporating *both* SAS and S (or R) code in a document. The present version allows control (via the source filename's extension) over the order in which the SAS and S code is executed. In tangling a source file containing both SAS and S code, two code files are

SASweave Demo

Russ Lenth

May 30, 2007

This illustrates how to use `SASweave` to integrate SAS code and output in a LaTeX document.

```
SAS> proc univariate data = sashelp.shoes;
SAS>   var sales;
SAS>   ods select moments;
```

```
The UNIVARIATE Procedure
Variable:  Sales  (Total Sales)
                          Moments
N                         395    Sum Weights               395
Mean                85700.1671    Sum Observations      33851566
Std Deviation       129107.234    Variance            1.66687E10
Skewness            3.94185882    Kurtosis            24.5888987
Uncorrected SS      9.46854E12    Corrected SS        6.56746E12
Coeff Variation     150.649921    Std Error Mean      6496.08993
```

We can also easily include graphics...

```
SAS> proc gplot data=sashelp.shoes;
SAS>   plot returns * sales;
```

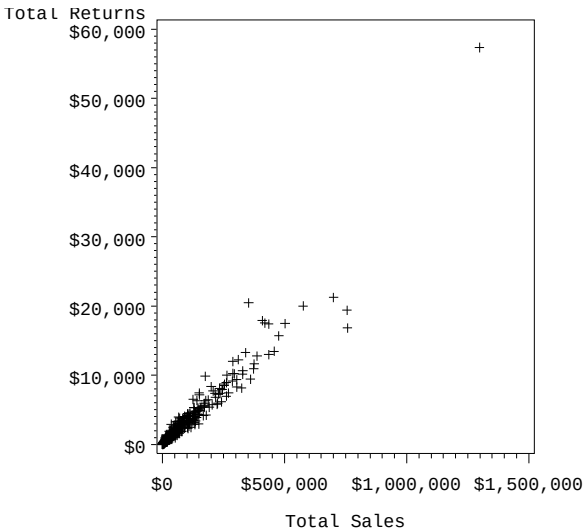


Figure 2: PDF file produced by running SASweave on the file in Exhibit 1.

Table 1: Filename extensions for use by SASweave.

Extension(s)	Description
`.SAStex`	SAS code only
`.Rtex` or `.Stex`	S code only (LaTeX syntax)
`.nw` or `.Rnw` or `.Snw`	S code only (noweb syntax)
`.SRtex` or `.SASRtex`	Both: SAS code first, then S (LaTeX syntax)
`.RStex` or `.RSAStex`	Both: S code (LaTeX syntax) first, then SAS
`.SASnw`	Both: SAS code first, then S (noweb syntax)
`.nwSAS`	Both: S code (noweb syntax) first, then SAS
`.tex`	Pass file to `pdflatex`

created.

SASweave code-chunk specifications are patterned after Sweave's LaTeX-like syntax for delimiting code chunks, rather than its noweb syntax. When a document contains both SAS and R code chunks, either the noweb or LaTeX syntax may be used for the R code. We did not attempt to produce an exact equivalent of Sweave. There are some extensions, some things that work differently, and some missing capabilities (e.g., in-text evaluation of expressions).

The present version of SASweave is designed to provide shell scripts `sasweave` and `sastangle` (for Unix/Linux or Windows) rather than R functions. Actually, the shell scripts are very minimal and the bulk of the code is a set of awk scripts (the Gnu awk (gawk) or nawk language is required). An open-source version of gawk is available for Windows, and it is provided in the Windows installer for SASweave.

This article is organized as follows. Section 2 details how to prepare the source file, and the various options for controlling how (and whether) code chunks, output, and graphics are displayed. Section 3 Describes how to run the shell scripts for SASweave. Section 4 provides some examples to illustrate how to handle certain situations. Finally, a description of each of the shell scripts and awk scripts is provided in Section 5.

2. Preparing the source file

To use SASweave, prepare a text file (hereafter called the "source file") containing standard LaTeX markup and one or more `SAScode` environments. SASweave supports source files containing SAS code, R code, both, or neither. When both are present, it can matter whether SAS or R is run first. For that reason, we have defined standard filename extensions that determine how a file is processed; those extensions are detailed in Table 1. Of course, all standard Sweave extensions are supported; files having those extensions are passed directly to Sweave. Also, a file with a `.tex` extension is passed straight to `pdflatex`. This makes it possible to use the same command to process a very wide variety of LaTeX-based documents.

The source file may contain option specifications that control how code chunks are processed. A `\SASweaveOpts{}` command, which changes the defaults for all subsequent code chunks,

may appear (alone on a line) anywhere in the source file. One-time options for a given code chunk may be given in braces following a `\begin{SAScode}` statement.

For example, to change the prompt for all code-chunk listings and put them in a box, we could include this statement in the source file:

```
\SASweaveOpts{prompt=Example: , codefmt += frame=single}
```

To embed a code chunk that is executed but completely invisible in the document, we would use

```
\begin{SAScode}{echo=FALSE,hide}
  ... SAS statements ...
\end{SAScode}
```

In order to be interpreted correctly, all `\begin{SAScode}`, `\end{SAScode}`, and `\SASweaveOpts` statements must start at the beginning of a line of the source file.

Option details

Options are enclosed in braces at the end of a `\begin{SAScode}` or `\SASweaveOpts` statement, and specified as a list of *keyword=value* pairs, separated by commas. Any whitespace in the options list is ignored, except in a `prompt` option (see below). Generally, options will appear on the same line with `\begin{SAScode}` or `\SASweaveOpts`; but if you want to extend them to additional lines, put an ampersand (`&`) at the end of the line. Anything after the closing brace is ignored.

Many options are boolean; these may be specified as `TRUE` or `FALSE`, or simply as `T` or `F`; if a boolean option is specified but not given a value, it is taken as `TRUE`. Thus, `\begin{SAScode}{fig}` is equivalent to `\begin{SAScode}{fig=TRUE}`. All keywords and values are case-sensitive. The following five characters are used in parsing options, and hence cannot be used in other ways:

```
{   }   ,   =   &
```

2.1. Options for code and output listings

`echo` (*Type:* boolean *Default value:* `TRUE`)
: Determines whether the code chunk is displayed in the document. If `TRUE`, each line is displayed, preceded by the current `prompt` string.

`hide` (*Type:* boolean *Default value:* `FALSE`)
: If `TRUE`, the listing output from SAS is not shown.

`results` (*Type:* text *Default value:* `verbatim`)
: A value of `verbatim` is equivalent to `hide=FALSE`, and a value of `hide` is equivalent to `hide=TRUE`. There is no `results=tex` option like there is in Sweave.

`eval` (*Type:* boolean *Default value:* `TRUE`)
If `FALSE`, the code chunk is not actually evaluated; it is simply displayed. This is useful when you want to display the commands only, and show the results elsewhere in the document rather than immediately following the code listing. When evaluation is suppressed, then obviously there will be no output, and thus `hide` is automatically set to `TRUE` when `eval=FALSE`.

`codefmt` (*Type:* Text *Default value:* (null))
Here you can specify any options for formatting the listing of a code chunk. Code chunks are put into a verbatim-like environment named `SASinput` derived from the LaTeX package `fancyvrb` (Van Zandt 1998). The value of `codefmt` may be any of the customization commands available for that package. However, you must separate the commands with semicolons instead of commas. Also, remember that braces are illegal within SASweave options, so you may need to work around them by defining macros. Here is an example:

```
\newcommand{\red}{\color{red}}
\begin{SAScode}{codefmt += formatcom=\red;frame=bottomline;fontfamily=courier}
  . . .
\end{SAScode}
```

The "`+=`" operator (available only here and for `outfmt`) causes the given commands to be *appended* to any formats already in existence (specified in a `\SASweaveOpts` line). (You may use the `fancyvrb` command `\RecustomVerbatimEnvironment` to change the defaults for `SASinput`.)

`outfmt` (*Type:* Text *Default value:* (null))
This is the same as `codefmt`, only it sets the format of the output listing environment `SASoutput`.

`prompt` (*Type:* Text *Default value:* `SAS>`)
The string specified here is added to the beginning of each line of a code chunk. Do not put it in quotation marks. Unlike other options, all whitespace between the "`=`" and the next "`,`" or closing "`}`" is kept as part of the prompt string.

`ls` (*Type:* integer *Default value:* `80`)
This specifies the limit on the number of characters in each line of SAS output (as in the SAS statement `options ls = 80;`). The line size is set to this value before evaluating each code chunk. (For technical reasons, SASweave must manage the line size; thus, any `options ls` statement within a code chunk has no effect on subsequent code chunks.)

2.2. Graphics options

`fig` (*Type:* boolean or integer *Default value:* `FALSE`)
If `TRUE` or positive, SASweave sets up a PDF file to receive graphical output, and the graph(s) are included in the document. `Fig=TRUE` implies that one graph will be created. If, say, `fig=3`, then SASweave expects 3 graphs to be generated. The code *must* produce at least the number of graphs specified, or you will get an error condition. Moreover,

use of `fig` requires graphics to be generated by SAS/GRAPH; the newer experimental ODS graphics capabilities are not supported.

The remaining options in this section have an effect only if `fig` is not `FALSE`.

`width` (*Type:* number *Default value:* `.6`)
This specifies the actual width of the included graph, as a multiple of `\linewidth`, similar to what is done using `\setkeys{Gin}` in Sweave. (This is completely different from the `width` option in Sweave.)

`hsize` (*Type:* number *Default value:* `4.0`)

`vsize` (*Type:* number *Default value:* `4.0`)
These options specify the `hsize` and `vsize` values in the `goptions` statement generated by SASweave. It sets the width and height, in inches, of the plot in the PDF output file. It does not affect the displayed width of the graph in the document (use the `width` option to change that). Changing `hsize` and/or `vsize` will affect the shape of the plot and the apparent font size of labels and symbols.

`striptitle` (*Type:* boolean *Default value:* `TRUE`)
When `TRUE`, the top 30 points of the plot (relative to `vsize`) are clipped off. SAS tends to put extra space at the top of plots, even when no title is given, and this tightens-up the spacing around the plot.

`plotname` (*Type:* LaTeX macro name *Default value:* (null))
If this is null, plots are displayed just below the SAS code and/or output listing. If a LaTeX macro name is provided here, the plots are not automatically included; instead, macros are defined to be the appropriate `\includegraphics` commands, and you can use them later to manually include the graphs wherever you like. The given macro name as-is will produce the first graph. If multiple graphs are created, you may call them up by appending the macro name with letters `A`, `B`, `C`, etc. For example, the options `fig=3` and `plotname=\myplot` will create the macros `\myplot`, `\myplotA`, `\myplotB`, and `\myplotC`; `\myplot` and `\myplotA` refer to the same graph (the first one).

Note that `plotname` creates LaTeX macros. To control the name of the `.pdf` file where the plot is saved, use the `label` option (see Subsection 2.4). Manual graphics inclusion of such a file will prove frustrating, however, because SAS does not set the PDF page size to be the same as that of the graph.

`figdir` (*Type:* string *Default value:* `./`)
This specifies the directory where graphics files are to be stored and retrieved. The directory must already exist; it is not created.

`infigdir` (*Type:* string *Default value:* *`figdir`*)
This allows the figures to be retrieved from a different directory from where they are stored. This seems contradictory, but it becomes useful when the source file is to be woven into a `.tex` file (using `sasweave -t`), for later inclusion into a main `.tex` document in a different directory. You then want to make `infigdir` match what it needs to be relative to the location of the main document.

2.3. Options for file handling

`split` (*Type:* boolean *Default value:* `FALSE`)
: If `FALSE`, the results of weaving the code chunks are all incorporated in the main `.tex` file; if `TRUE`, these results are written to separate `.tex` files and read-in to the main file with an `\input` statement.

`prefix.string` (*Type:* string *Default value:* `base`)
: This sets the beginnings of the names of all graphics files, as well as of the `.tex` files generated if `split` is `TRUE`. It may include a directory path, delimited by slashes. A hyphen, a code-chunk label, and the appropriate extension are appended to the prefix string. For example, suppose that `prefix.string` is set to `chunks/myprefix`. If code chunk #3 produces graphics, the associated graphics file is named `chunks/myprefix-swv-003.pdf` (it may have several pages if there are multiple figures); in addition, if `split=TRUE`, the verbatim output for the chunk will be written to `chunks/myprefix-swv-003.tex`. If no `prefix.string` is given and the source file is named `myfile.SAStex`, the defaults are `myfile-swv-003.pdf` and `myfile-swv-003.tex`, respectively. If a `label` (see Subsection 2.4) is also specified, it is used in place of `swv-003` weherever it appears in these illustrations.

2.4. Options for code reuse

`label` (*Type:* name *Default value:* `lastchunk`)
: This specifies a name under which the current code chunk is saved. The same code may then be reused as part of a later code chunk, using the `\SAScoderef` command. Unlike Sweave, the `label` keyword is required. The default label of `lastchunk` is a handy way to reuse the previous code chunk.

  If specified, the label is also used in lieu of the chunk number in naming any files created by that chunk. For example, if the third code chunk in the source file `mysource.SAStex` produces a graph, the graph will be saved to a file named `mysource-swv-003.pdf`. However, if it is given a label of `foo`, then the file name will be `mysource-foo.pdf`.

`showref` (*Type:* boolean *Default value:* `FALSE`)
: If `TRUE`, any SAS code recalled using `\SAScoderef` will be displayed in the code listing (as long as `hide` is `FALSE`). If `FALSE`, reused code will be excluded from the listing. This makes it possible to hide pieces of SAS code (perhaps `ods` statements) that you don't want to display.

  The `\SAScoderef` command has a starred version `\SAScoderef*` that will force the reused chunk to be displayed regardless of the value of `showref`. This allows you to display some reused code while hiding other code within the same chunk.

2.5. Argument substitution

It is possible to define reusable chunks of SAS code that accept arguments to be provided later in a `\SAScoderef` statement. This is done in much the same ways as a LaTeX macro

definition: set up a code chunk that contains the symbols `#1`, `#2`, etc. as placeholders. You need to assign this chunk a label, and use options of `eval=FALSE` and (probably) `echo=FALSE`. Then incorporate this chunk in later code chunks using

```
\SAScoderef{label}{arg1}{arg2} ···
```

(or the same with `\SAScoderef*`), where *`label`* is the label given the previously defined code chunk. The contents of *`arg1`* will be substituted for any appearances of `#1` , *`arg2`* will be substituted for any appearances of `#2`, and so forth. No careful checking is done by SASweave; if you provide too many arguments, they'll just have no effect, and if you provide too few, the code passed to SAS will contain things like "`#1`," likely producing an error.

3. Running SASweave

The shell commands for tangling and weaving are as follows:

```
sastangle options filename[.SAStex]
sasweave options filename[.SAStex]
```

where *`filename`* is the name of the source file (if an extension is not given, `.SAStex` is assumed). The possible *`options`* can include flags from the list below.

Option for `sastangle`

`-s` Run SAS after the `.sas` file is created.

Treatment of intermediate files in `sasweave` (default is `-g`)

`-c` Clean up the intermediate files that are generated. If errors occur, intermediate files are left anyway. If the creation of the `.tex` file is successful, the `.sas` and `.lst` files are deleted, and the SAS `.log` file is renamed with an extension of `.saslog`. If `pdflatex` processing is requested and it is successful, the `.log` and `.saslog` files and any intermediate graphics files are deleted. (However, only files with standard names are deleted; so the `label`, `figdir`, or `prefix.string` options—see Section 2—may prevent graphics files from being deleted.) The `.tex` and `.aux` files are not deleted.

`-g` Clean up intermediate files, but do not delete the graphics files.

`-l` Leave all the intermediate files in place.

Flags to specify the target of `sasweave` (default is `-p`)

`-p` Run `pdflatex` on the resulting `.tex` file.

`-t` Terminate after the `.tex` file is produced.

`-n` Rename the `.tex` file to an extension of `.nw`. This is useful for subsequent Sweave processing when the source file also includes Sweave content in the noweb syntax (with S code delimited by `<<>>` and `@`).

`-r` Rename the `.tex` file to an extension of `.Rtex`. This is useful for subsequent processing when the source file also includes Sweave content.

4. Examples

The examples in this section illustrate how to use some of SASweave's capabilities.

4.1. Basic use of SASweave

The example in Section 1 illustrates the most basic use of SASweave when no options (other than font size) are specified. The first code chunk in Exhibit 1 is a simple SAS program that produces only listing output.

The second code chunk shows the simplest way to incorporate a figure. The default shape of the plotting region is square, but SAS's formatting of labels causes it to be rather tall because the label for the vertical axis is so long.

In the remaining examples, the SASweave source is displayed in a box, and the results are shown below it.

4.2. R and SAS together

Here is a simple example where both R and SAS code are incorporated in the same source file. One of the standard datasets in R is summarized, and it is written to a text file, imported into SAS, and summarized there as well. In this example, it is very important that the R code be run first, as it creates the data needed by SAS; hence the filename extension used is `.RSAStex`. By default, SAS statements are formatted in `\small` font. Font sizing is not provided among the options in Sweave, so we do it manually.

```
{\small
\begin{Scode}
table(chickwts$feed)
write.table(chickwts, file="chickwts.txt", row.names=FALSE, quote=FALSE)
\end{Scode}
}
\begin{SAScode}
data sasuser.chickwts;
  infile "chickwts.txt" firstobs = 2;
  input weight feed $;
proc freq data=sasuser.chickwts;
  table feed;
\end{SAScode}
```

```
R> table(chickwts$feed)

   casein horsebean   linseed  meatmeal   soybean sunflower
       12        10        12        11        14        12

R> write.table(chickwts, file = "chickwts.txt", row.names = FALSE,
R+     quote = FALSE)
```

```
SAS> data sasuser.chickwts;
SAS>   infile "chickwts.txt" firstobs = 2;
SAS>   input weight feed $;
SAS> proc freq data=sasuser.chickwts;
SAS>   table feed;

The FREQ Procedure
                                        Cumulative    Cumulative
feed        Frequency     Percent     Frequency      Percent
-----------------------------------------------------------------
casein             12       16.90            12        16.90
horsebea           10       14.08            22        30.99
linseed            12       16.90            34        47.89
meatmeal           11       15.49            45        63.38
soybean            14       19.72            59        83.10
sunflowe           12       16.90            71       100.00
```

4.3. Multiple figures in a float

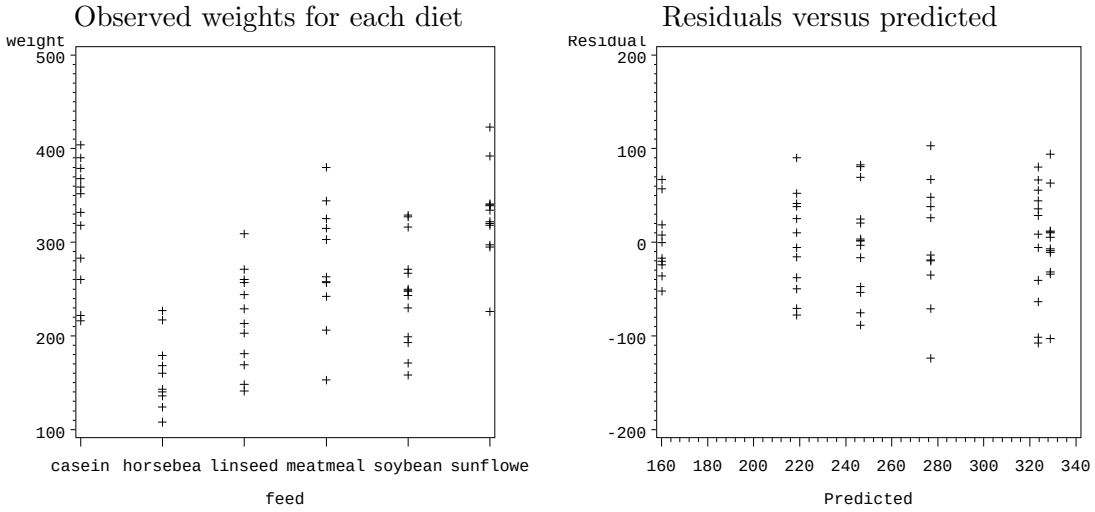
The following code segment illustrates the use of several options. First, we suppress the code listing (`echo=FALSE`). We ask for two plots (`fig=2`) of reduced width (`width=.45`). Rather than the default placement of plots, we specify that that they be saved as LaTeX macros (`plotname=\chickPlot`) for later inclusion in a `figure` environment. Subsequently, the macros `\chickPlotA` and `\chickPlotB` call up the two plots.

```
\begin{SAScode}{echo=FALSE, fig=2, width=.45, plotname=\chickPlot, &
  outfmt = fontsize=\footnotesize}
proc glm data=sasuser.chickwts;
  class feed;
  model weight = feed / ss1;
  output out=chickfit p=Predicted r=Residual;
  means feed;
  ods exclude NObs ClassLevels;
proc gplot data=chickfit;
  plot weight * feed
       Residual * Predicted;
\end{SAScode}
See Figure~\ref{chickfig} for some supplementary displays.
\begin{figure}
  \caption{Plots of the \texttt{chickwts} data.}\label{chickfig}
  \begin{center}
    \begin{tabular}{cc}
    Observed weights for each diet & Residuals versus predicted\\
    \chickPlotA & \chickPlotB
    \end{tabular}
  \end{center}
\end{figure}
```

```
The GLM Procedure

Dependent Variable: weight
                                      Sum of
```

Figure 3: Plots of the `chickwts` data.




```
Source                     DF       Squares    Mean Square   F Value   Pr > F
Model                       5    231129.1621     46225.8324     15.36   <.0001
Error                      65    195556.0210      3008.5542
Corrected Total            70    426685.1831

R-Square     Coeff Var      Root MSE    weight Mean
0.541685      20.99052      54.85029       261.3099

Source                     DF      Type I SS    Mean Square   F Value   Pr > F
feed                        5    231129.1621     46225.8324     15.36   <.0001
The GLM Procedure
Level of            ------------weight-----------
feed          N             Mean          Std Dev
casein       12       323.583333       64.4338397
horsebea     10       160.200000       38.6258405
linseed      12       218.750000       52.2356983
meatmeal     11       276.909091       64.9006233
soybean      14       246.428571       54.1290684
sunflowe     12       328.916667       48.8363842
```

See Figure 3 for some supplementary displays.

4.4. Separating code and output; hiding code

Sometimes we want to put the results in a separate place from the code listing; for example, in a float. The best way to do this is to reuse the same code, via labels. This example shows two code chunks. Chunk 1 contains the code we want to run; but it is only listed, not evaluated (`eval=FALSE`). Code chunk 2 recalls chunk 1 using its default label of `lastchunk`, and adds an `ODS` statement to restrict the output; this time it is executed, but the code listing is suppressed (`echo=FALSE`).

Figure 4: Results of PROC ROBUSTREG.

The ROBUSTREG Procedure

Parameter Estimates

Parameter		DF	Estimate	Standard Error	95% Confidence Limits		Chi-Square	Pr > ChiSq
Intercept		1	329.0835	16.8600	296.0384	362.1286	380.97	<.0001
feed	casein	1	-0.0836	23.8437	-46.8163	46.6492	0.00	0.9972
feed	horsebea	1	-169.863	25.0075	-218.877	-120.850	46.14	<.0001
feed	linseed	1	-110.343	23.8437	-157.076	-63.6106	21.42	<.0001
feed	meatmeal	1	-49.5349	24.3796	-97.3180	-1.7518	4.13	0.0422
feed	soybean	1	-82.8402	22.9764	-127.873	-37.8074	13.00	0.0003
feed	sunflowe	0	0.0000	.	.	.	.	.
Scale		1	52.5603					

Robust Linear Tests

FEED_OVERALL

Test	Test Statistic	Lambda	DF	Chi-Square	Pr > ChiSq
Rho	12.4552	0.7977	5	15.61	0.0080
Rn2	71.1819		5	71.18	<.0001

```
Here is the SAS code to perform a robust analysis of the chick-weights data.
The output is displayed in Figure~\ref{robust-out}.
% Chunk 1
\begin{SAScode}{prompt=, eval=FALSE}
proc robustreg data = sasuser.chickwts  method = M (wf = bisquare);
  class feed;
  model weight = feed;
  Feed_overall: test feed;
\end{SAScode}
\begin{figure}
\caption{Results of \texttt{PROC ROBUSTREG}.}\label{robust-out}
% Chunk 2
\begin{SAScode}{echo=FALSE}
\SAScoderef{lastchunk}
ODS select ParameterEstimates TestsProfile;
\end{SAScode}
\end{figure}
```

Here is the SAS code to perform a robust analysis of the chick-weights data. The output is displayed in Figure 4.

```
proc robustreg data = sasuser.chickwts  method = M (wf = bisquare);
  class feed;
  model weight = feed;
  Feed_overall: test feed;
```

4.5. Argument substitution; hiding code

In this example, we set up (but do not evaluate or echo) a code chunk named `import`; it contains the strings `#1`, `#2`, and `#3`, which serve as placeholders for arguments to be supplied later. In the second code chunk, we read a data file and run `PROC REG`; the file-reading part is done by re-using the `import` chunk with appropriate arguments supplied. That part of the code is not displayed in the listing, however, because `showrefs` is `FALSE` by default.

```
\SASweaveOpts{eval=FALSE} %%% suppress all evaluation for this example

\begin{SAScode}{echo=FALSE, eval=FALSE, label=import}
proc import
  datafile = "#1"
  out = #2
  dbms = #3
  replace ;
\end{SAScode}
%
% Secretly read-in a file before an analysis ...
\begin{SAScode}{fig=2}
\SAScoderef{import}{c:\BPSdata\ta05-03.dat}{reactTime}{TAB}
proc reg data=reactTime;
  model Time = Distance;
  plot Time*Distance Residual.*Predicted.;
\end{SAScode}
```

```
SAS> proc reg data=reactTime;
SAS>   model Time = Distance;
SAS>   plot Time*Distance Residual.*Predicted.;
```

Note that `import` is effectively a macro for SASweave; and we can actually define new macros based on it. The first code chunk below simply calls up `import` and substitutes appropriate arguments so that it becomes a simplified macro suitable for importing comma-delimited files. It is then used and displayed.

```
% Create a new "macro"
\begin{SAScode}{echo=FALSE, eval=FALSE, label=importCSV}
\SAScoderef{import}{#1.csv}{#1}{CSV}
\end{SAScode}
%
% Test run...
\begin{SAScode}{showref}
\SAScoderef{importCSV}{newFile}
\end{SAScode}
```

```
SAS> proc import
SAS>   datafile = "newFile.csv"
SAS>   out = newFile
SAS>   dbms = CSV
SAS>   replace ;
```

5. Implementation

This section gives an overview of how the SASweave software is structured, and a description of the main tasks of each body of code.

The basic approach in this SASweave implementation is rather brute-force in nature: a single SAS program is created that contains everything needed for the final `.tex` file—both code and text chunks. The text chunks and code listings are simply printed in the right places in the SAS output. The output file is then post-processed and saved as a `.tex` file, which, optionally, is passed to `pdflatex` to produce a `.pdf` file with the formatted document.

For the verbatim listing of code and output, we provide a LaTeX package named `SasWeave.sty` that defines verbatim-like environments `SASinput` and `SASoutput`; these are based on the standard LaTeX package `fancyvrb`. `SasWeave.sty` is similar to the package `Sweave.sty` that is part of Sweave. (Originally, it was named `SASweave.sty`, but this had the effect of tricking Sweave into thinking that `Sweave.sty` was already loaded.)

The pre- and post-SAS operations are done as much as possible by means of awk scripts. awk is an ideal scripting language for this purpose, because its design focuses on pattern-matching, and there is an implied loop where we go through a file line-by-line. That is exactly what is needed here. Moreover, awk is quite forgiving (we leave error-checking to SAS and LaTeX), and an implementation of awk is available for virtually any platform.

The main workhorse among the awk scripts is the one named `saswv1.awk` (henceforth called just `saswv1`), which reads the source file and writes the `.sas` file. This script looks for five main conditions: lines that start with "`\begin{SASweaveOpts}`," "`\begin{SAScode}`," and "`\end{SAScode}`", and processing of cases where a flag named `sas` is zero (meaning the current source-file line is in a text chunk) or 1 (it is in a code chunk). By doing appropriate things in response to these five conditions, the script arranges things so that if we are weaving the source file, the output `.sas` file will be organized as follows (and in the order described).

1. Text chunks go into `put` statements within `PROC IML`. (This includes inserting judicious linefeeds to keep these statements from exceeding the line-width limit. For this reason, SASweave must control SAS's `LS` option.)

2. If code is to be echoed, the appropriate verbatim environment `SASinput` is set up and included at the end of the preceding text chunk.

3. If output is to be displayed, a `\begin{SASoutput}` statement is added to the text chunk.

4. Appropriate setup code is added to the SAS program. These include setting up the desired line size at `ls`, and if a figure is to be saved, some `goptions` statements to setup an output `pdf` file.

5. The SAS code itself is added to the SAS program.

6. At the end of a code chunk, we start a new text chunk (`PROC IML` again), and start it with `\end{SASoutput}`.

7. If there are any figures, the needed `\includegraphics` statements are generated. If there is no `plotname`, these are added to the text chunk; otherwise, they are wrapped in LaTeXmacro definitions before adding them to the text chunk.

8. We are now ready for more text from the source file (step 1).

(One can see exactly how the `.sas` file is structured by weaving a file with the `-l` option.)

The `saswv1` script also contains some startup and ending code and a few functions to ease in processing options. It also calls other functions defined in a different awk script that is loaded at the same time. These externally-supplied functions determine the actions taken at the beginning of the run, at the beginning and end of a text chunk, setting up a graph, and outputting the lines of a text chunk. There are two versions of these functions. The ones in the file `saswsetup.awk` are used for weaving the source file (for eventual creation of a `.tex` file). The alternative functions in `sastsetup.awk` are suitable for tangling. Thus, the `sastsetup.awk` function for outputting text chunks does nothing at all, and the others there do very little (for example, graphics are set up with the dimensions specified in the SASweave options, but they go to the default device rather than a `.pdf` file). The design decision to provide different output routines for tangling and weaving, while keeping the same basic `saswv1` script, helps with maintainability and consistency; a change made to `saswv1.awk` will appropriately affect both tangling and weaving operations.

The script `saswv2.awk` handles post-processing of the `.lst` file generated by SAS (in weaving operations only) to create a `.tex` file. It is shorter and simpler than `saswv1`, but there are more patterns that need handling. What complexity exists there is due to looking for empty `SASinput` and `SASoutput` environments so that they are not added to the `.tex` file. Beyond that, the main operations are stripping off the top two lines of each page, outputting only one blank line whenever two consecutive blank lines are encountered, and diverting chunks to other files when `split` is true (this is done by checking for certain signal lines that `saswv1` outputs).

The same maintainability and portability considerations as described for `saswv1` motivate the design of the command-line interface. For each operating system, we need a shell script that serves as a front end to the awk scripts. The unix/linux shell scripts `sastangle` and `sasweave`, and the Windows scripts `sastangle.bat` and `sasweave.bat`, are all as minimal as possible. They simply identify and change to the directory where the source file resides, and then call one of the awk scripts `saswmain.awk` (for weaving) or `sastmain.awk` (for tangling). These two scripts parse the command line for flags and determine the source file's extension. Based on the extension and flags, the source file or one of its derivatives is passed to `saswv1`, SAS, `saswv2`, Sweave, and `pdflatex` as is appropriate and in the correct sequence. The scripts call the awk `system` function with appropriate arguments to invoke a shell and run SAS, R, and `pdflatex` as needed.

A certain amount of file copying and renaming takes place when both R and SAS code needs processing. For example, with a `.RSAStex` source file, we first copy it to another file with an extension of `.Rtex`, then run Sweave; the resulting `.tex` file is renamed with a `.SAStex` extension before passing it to SASweave. This management is also done using the `system` function.

The `saswmain` and `sastmain` scripts each require a common additional script named `saswcfg.awk`, which defines certain variables with system-specific values. This configuration file gives the path where the awk scripts are installed, and the commands to run SAS, R, `pdflatex`, and Sweave. The Windows installer for SASweave creates this file. The one for unix/linux is simply copied and edited, but typically only the awk-script path needs modification.

6. Discussion

SASweave provides a simple and reliable way of presenting and documenting SAS analyses. We have used it to great benefit in consulting, research and teaching. In research and consulting, one or more SASweave source files provide structure for preparing analyses and simulation studies, etc. You can document what you did, and when the files are processed, you have a reliable record of exactly what was done, along with the results. In teaching how to use SAS, SASweave streamlines the preparation of class handouts. Also, if "live" SAS analyses are done in class, it is an easy matter for the instructor to save the `.sas` file, enclose selected portions in `SAScode` environments, add comments (or not), and use SASweave to make a documented form of the class examples with output included.

We have tried to make SASweave behave similarly to Sweave where that is appropriate and practical. One notable difference between the two arises from the fact that Sweave uses R to parse the input statements and simulate an interactive mode, while SASweave does not parse the code in any way. One code chunk in Sweave might produce several sets of code listings interspersed with output listings. In SASweave, one code chunk always produces one code listing, followed by one output listing containing all the results. (There is a small up side to this: unlike Sweave, the code listing is displayed with exactly the same spacing and line breaks as in the source file.) For the reasons described here, we recommend that each SAS procedure that produces output be put into its own code chunk.

That said, it is also necessary that *all* the SAS code for a given procedure be incorporated in the same code chunk; otherwise, errors will result. The reason for this is that SASweave creates one SAS program that handles both the code chunks and the text chunks, and each text chunk is handled by initiating a new SAS procedure. Where this becomes most noticeable is in using `PROC IML`, where it is natural to want to interleaf code and results. In the current implementation, we cannot do this; we have to put all the `IML` code in one code chunk. Other more minor shortcomings include non-availability of PostScript graphs, and no support for the emerging "ODS graphics" provisions in certain SAS procedures.

On a positive note, SASweave also offers some nice extensions (we think) of Sweave. The main ones include support for multiple figures in one code chunk, ability to assign macro names to plots, argument substitution, and the ability to hide reused code. Those go on our wish list for future releases of Sweave.

References

Højsgaard S (2006). "SASRweave: An R package for literate programming with SAS and R at the same time." http://genetics.agrsci.dk/~sorenh/misc/software/, October 30, 2006.

Knuth DE (1992). "Literate Programming." *CSLI Lecture Notes 27*, Center for the Study of Language and Information, Stanford, California.

Leisch F (2002). "Dynamic generation of statistical reports using literate data analysis." In W Härdle, B Rönz (eds.), "Compstat 2002—Proceedings in Computational Statistics," pp. 575–580. Physika Verlag, Heidelberg, Germany. ISBN 3-7908-1517-9.

Van Zandt T (1998). "The `fancyvrb` package: Fancy verbatims in LaTeX." Available in all standard LaTeX distributions.

Affiliation:

Russell V. Lenth
Department of Statistics and Actuarial Science
The University of Iowa
Iowa City, Iowa 52242, USA
E-mail: russell-lenth@uiowa.edu
URL: http://www.stat.uiowa.edu/~rlenth

Søren Højsgaard
University of Aarhus
Faculty of Agricultural Sciences
Research Centre Foulum
Institute of Genetics and Biotechnology
Blichers Allé 20, P.O. Box 50
DK-8830 Tjele
E-mail: Soren.Hojsgaard@agrsci.dk
URL: www.agrsci.dk

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume VV, Issue II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd