PhUSE 2008

Paper TS06

SAS and LaTeX – a Perfect Match?

Juha-Pekka Perttola, F. Hoffmann-La Roche AG, Basel, Switzerland

ABSTRACT

Improved flexibility in creating PDF documents from SAS® can be obtained by making use of LaTeX. In this paper I will discuss a way to author PDF files through LaTeX by using SAS macros to write the LaTeX source code. This setup allows combining SAS code, log, output and graphs with freely written textual content. All LaTeX functionality is available to create complex PDF documents directly from the SAS Program Editor. In addition, any update in the underlying data and possible changes to results thereof will be directly reflected in the document upon recompilation.
LaTeX is a widely used and freely available document preparation system. It can typeset a wide variety of documents including presentation slides. LaTeX is highly customizable and especially powerful in writing technical documentation and mathematical text.
Using this framework I will also demonstrate that it is possible to use SAS macros to typeset SAS/IML® matrices directly to the PDF document, create LaTeX PSTricks graphics from SAS, construct compendiums to store content from various applications and perform image transformations on the basis of SAS data.

INTRODUCTION

WHAT IS LaTeX?

Wikipedia describes LaTeX in the following way:
LaTeX is a document preparation system for the TeX typesetting program. It offers programmable desktop publishing features and extensive facilities for automating most aspects of typesetting and desktop publishing, including numbering and cross-referencing, tables and figures, page layout, bibliographies, and much more. LaTeX was originally written in 1984 by Leslie Lamport and has become the dominant method for using TeX; few people write in plain TeX anymore. The current version is LaTeX 2$_\varepsilon$.

LaTeX is essentially a collection of macros to ease the writing of TeX commands (Lamport 1994). The first version of TeX was developed by Donald Knuth in 1978 (Knuth 1986). TeX and LaTeX are freely available, usually as a distribution like MiKTeX which was used to compile this document. LaTeX distributions typically contain a very large amount of files and MiKTeX provides an easy way to keep them all up to date. There is a huge number of customizable LaTeX packages, examples and documentation available on the Internet.

For a good introduction to the current version of LaTeX, please see for example "The Not So Short Introduction to LaTeX2$_\varepsilon$" (Oetiker et al. 2008).

SOME HISTORY OF THIS DEVELOPMENT

All the presented SAS macros have been programmed as part of my study related activities some years ago. Back then I created this way of working to help myself in compiling a couple of lengthy PDF documents.

This paper and the presentation slides that accompany it have been written using the described setup.

The finalized PDF file is produced by submitting the main program in SAS once. This follows the main idea of being always one key press away from the end result. The second principle has been to minimize the switching between applications. Therefore all the programming is done in the SAS Program Editor.

SAS MACROS

FOLDER STRUCTURE

To keep the programs and outputs organized, they are stored in a simple folder structure looking like this:

```
C:\SAS2LATEX\PHUSE_PAPER
+---Data
+---LaTeX
\---SAS
    \---Include
```

LaTeX folder will contain all the produced outputs and LaTeX related files while the resulting PDF is in the end of the execution created on the top level. The Include subfolder contains the necessary SAS macros to write this document:

```
endprint.sas
matops.sas
r.sas
s.sas
setprint.sas
shortcuts.sas
t.sas
```

Some examples of how the different macros are used can be found later in this paper.

SAS MACRO TO WRITE THE LaTeX COMMAND FILE

The central macro in authoring the LaTeX document is called %t. The name comes from the fact that everything put as the first parameter of this macro will be written to a .tex file. The code is quite short and presented as the code example 1. The macro is not perfect but it works sufficiently well in most situations.

Code example 1: t macro

```
**********************;
* j-pp 16.3.2006    *;
**********************;
%MACRO t(_text,_opt);
OPTIONS NOSOURCE NONOTES;
%IF &_opt = n %THEN %LET _opt = new;
DATA _NULL_;
  LENGTH _text $5000.;
  _text = SYMGET('_text') ;
  _text = COMPRESS(_text,"'");
  _text = TRANSLATE(_text,"'",&quoterep);
  CALL SYMPUT('_text',_text);
RUN;
PROC PRINTTO LOG = """&outpath\&texfile..tex""" &_opt;
RUN;
%PUT &_text;
PROC PRINTTO;RUN;
OPTIONS SOURCE NOTES;
%MEND;
```

The t macro enables us now to write to a LaTeX command file from SAS in the following manner:

```
%t('Hello World!');
```

The .tex command file is stored in the LaTeX subdirectory and it will be continuously appended in the background as the main SAS program executes. In the end of the execution, the .tex file will be compiled using LaTeX – in this example case, to produce the following text in the PDF:

Hello World!

Utilizing SAS keyboard macros the

```
%t(' ');
```

macro calls can be added around multiple command rows in one go, so it is not necessary to always keep typing them.

SOME FURTHER EXAMPLES

It is possible to define SAS macro variables and use them in the typesetting string. The code example 2 illustrates this and typesets the first 15 rows of the Pascal triangle.

Code example 2: Pascal triangle

```
%MACRO pt(n);
%t('\begin{center}');
%DO i=0 %TO %EVAL(&n-1);
  %DO j=0 %TO &i;
    %t('$%SYSEVALF(%SYSFUNC(FACT(&i))/(%SYSFUNC(FACT(&j))*%SYSFUNC(FACT(&i-&j))))$');
    %IF &i=&j %THEN %DO;
      %t('\\');
    %END;
  %END;
%END;
%t('\end{center}');
%MEND;
%pt(15);
```

Calling the pt macro both calculates and prints the triangle:

$$
\begin{array}{c}
1 \\
1\ 1 \\
1\ 2\ 1 \\
1\ 3\ 3\ 1 \\
1\ 4\ 6\ 4\ 1 \\
1\ 5\ 10\ 10\ 5\ 1 \\
1\ 6\ 15\ 20\ 15\ 6\ 1 \\
1\ 7\ 21\ 35\ 35\ 21\ 7\ 1 \\
1\ 8\ 28\ 56\ 70\ 56\ 28\ 8\ 1 \\
1\ 9\ 36\ 84\ 126\ 126\ 84\ 36\ 9\ 1 \\
1\ 10\ 45\ 120\ 210\ 252\ 210\ 120\ 45\ 10\ 1 \\
1\ 11\ 55\ 165\ 330\ 462\ 462\ 330\ 165\ 55\ 11\ 1 \\
1\ 12\ 66\ 220\ 495\ 792\ 924\ 792\ 495\ 220\ 66\ 12\ 1 \\
1\ 13\ 78\ 286\ 715\ 1287\ 1716\ 1716\ 1287\ 715\ 286\ 78\ 13\ 1 \\
1\ 14\ 91\ 364\ 1001\ 2002\ 3003\ 3432\ 3003\ 2002\ 1001\ 364\ 91\ 14\ 1
\end{array}
$$

This example also shows how to run the LaTeX commands within a SAS loop. This has proven to be very powerful in the creation of different kinds of content.

As all LaTeX code is plain text we can also write any complex mathematical formula directly from SAS, equations (1) and (2) offer two examples of this.

Code example 3: Equation (1)

```
%t('\begin{equation}');
%t('F(x;\mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x}');
%t('e^{-\frac{(u-\mu)^2}{2\sigma^2}} \, du');
%t('\end{equation}');
```

Result in the PDF:

$$F(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(u-\mu)^2}{2\sigma^2}} \, du \tag{1}$$

Code example 4: Equation (2)

```
%LET b2=1+\beta^2; %LET b1=-\beta;
%LET ld=\ldots; %LET vd=\vdots; %LET dd=\ddots;
%LET invS=\begin{bmatrix}
&b2  & &b1   & 0    & &ld  & 0    & 0   \\
&b1  & &b2   & &b1  & &ld  & 0    & 0   \\
0    & &b1   & &b2  & &ld  & 0    & 0   \\
&vd  & &vd   & &vd  & &dd  & &vd  & &vd \\
0    & 0    & 0    & &ld  & &b2  & &b1 \\
0    & 0    & 0    & &ld  & &b1  & 1   \\
\end{bmatrix};
%t('\begin{align}');
%t('\lefteqn{ f_Y(y)=\left( \frac{1}{2\pi\sigma^2} \right)^{\frac{n}{2}}');
%t('exp \bigg\{{-\frac{1}{2\sigma^2} y \Sigma^{-1} y^{&mt}} \bigg\}} \\');
%t('\lefteqn{f_Y(y)=\left( \frac{1}{2\pi\sigma^2} \right)^{\frac{n}{2}}');
%t('exp \bigg\{{-\frac{1}{2\sigma^2} y &invS y^{&mt}} \bigg\}}');
%t('\end{align}');
```

Result in the PDF:

$$f_Y(y) = \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} exp\left\{-\frac{1}{2\sigma^2} y \Sigma^{-1} y^T\right\}$$

$$f_Y(y) = \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} exp\left\{-\frac{1}{2\sigma^2} y \begin{bmatrix} 1+\beta^2 & -\beta & 0 & \dots & 0 & 0 \\ -\beta & 1+\beta^2 & -\beta & \dots & 0 & 0 \\ 0 & -\beta & 1+\beta^2 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1+\beta^2 & -\beta \\ 0 & 0 & 0 & \dots & -\beta & 1 \end{bmatrix} y^T\right\} \tag{2}$$

Defining parts of the formula as SAS macro variables, as is done in the equation (2), makes it easy to write repeated elements. The Σ^{-1} related parts have been written in blue color for easier identification. The ampersand is used in the LaTeX syntax to separate the elements within the matrix. Quite remarkably no problems arise even as the ampersand is of course also used by SAS for special purposes.

SIMPLE EXAMPLE DOCUMENT

The code example 5 creates a simple one page PDF document with title and table of contents. Submitting the code within the folder structure presented previously, will automatically create the PDF document and open it in Adobe® Reader®. The program consist of three parts:

1. Creation of the proper running environment, mainly using SAS macro variables
2. Writing the LaTeX code
3. Compiling the created .tex file to a PDF and opening it

Code example 5: Simple example document

```
*************************************************************************;
* Simple example document, jpp 10.8.2008                                ;
*************************************************************************;
***Set environment*******************************************************;
%LET basepath=C:\sas2latex\simple;
%LET filepath=&basepath\SAS;
%LET outpath=&basepath\Latex;
%LET acrorpath=C:\Program Files\Adobe\Reader 9.0\Reader;
X CD &outpath;
%INCLUDE "&filepath\include\t.sas";
%LET texfile=Simple Example Document;
%LET quoterep='¢';
%LET SC=SYSTASK COMMAND;
***LaTeX code************************************************************;
%t('\documentclass{article}',n);
%t('\title{&texfile}');
%t('\author{jpp}');
%t('\begin{document}');
%t('\maketitle');
%t('\tableofcontents');
%t('\section{First Section}');
%t('Some text here.');
%t('\end{document}');
***Compile***************************************************************;
&SC "latex ""&outpath\&texfile..tex""" SHELL WAIT;
&SC "latex ""&outpath\&texfile..tex""" SHELL WAIT;
&SC "dvips ""&outpath\&texfile..dvi""" SHELL WAIT;
&SC "ps2pdf14 ""&outpath\&texfile..ps"" ""&basepath\&texfile..pdf""" SHELL WAIT;
&SC """&acrorpath\AcroRd32.exe"" ""&basepath\&texfile..pdf""";
&SC """&acrorpath\AcroRd32.exe""";
***End*******************************************************************;
```

Please notice that LaTeX needs to be invoked at least twice, as done in this example. This is necessary to create all the cross-references within the document. LaTeX creates a .dvi file which is converted to PostScript® with Dvips. ps2pdf script is then used to convert the .ps file to PDF. There is a TeX version called pdfTeX which can write the PDF directly but in some cases the outcome seems to be nicer when performing these additional conversions.

Usually it is helpful to divide the chapters of longer documents into separate SAS files and include them from the main program. In more complicated documents it is necessary to define a more complex LaTeX preamble than done here. The preamble is a collection of commands which tell LaTeX how the document should be compiled.
It is also a good idea to keep the folder structure as clean as possible. To achieve this I usually use SAS to wipe out all the pre-existing content of the LaTeX output directory in the beginning of every run. This way it is always guaranteed that all the result files are updated.

CREATING PRESENTATIONS

Among the available LaTeX packages there are several that are intended to write presentations. These packages make it possible to create PDF slides instead of text documents. The package I usually use is called powerdot (Adriaens & Ellison 2005) but probably some of the other packages are as useful. powerdot offers some premade slide layouts but it is also possible to customize ones own graphical look.

After the powerdot package is installed, it has to be set as the documentclass in the LaTeX preamble:
`%t('\documentclass[paper=screen,mode=present,style=tycja,fleqn]{powerdot}');`

Each slide is then defined within the slide environment as shown in the code example 6.

Code example 6: Presentation slide

```
%t('\begin{slide}{Slide title}');
%t('\pause');
%t('\begin{itemize}');
%t('\item First bullet \pause');
%t('\item Second bullet');
%t('\end{itemize}');
%t('\end{slide}');
```

The pause keywords animate the slide to add each bullet when a key is pressed. Within the slide environment, most of the normal LaTeX code is accepted. This makes it straightforward to build a presentation from the same elements as written to a LaTeX document.

In presentations it is particularly useful to define different slide sections to be included from the main program. This way it is possible to customize the presentation for different audiences by simply commenting out some of the sections and recompiling. It is also possible to write the slides concurrently in different languages and then set up a selection of the language in the main program. Same kind of selection can be done with the different slide styles.

CAPTURING RESULTS CREATED WITH SAS

To get the full benefit of using SAS to write the document, there should be an easy way to transfer the results calculated with SAS to the document. The easiest method as discussed earlier is to use macro variables but some additional macros make it possible to capture also other kinds of content.

CAPTURING TEXTUAL OUTPUT

The code example 7 calculates results to a text file with PROC CORR and saves the program that was run from the log window. After the files are saved, the t macro is used to issue a VerbatimInput command which typesets the created files to this document.

Code example 7: Capturing text from SAS

```
%setprint(progam.log,output.lst);
PROC CORR DATA=sashelp.class NOSIMPLE NOPROB
VAR height weight;
RUN;
%endprint;
%t('\VerbatimInput{program.log}');
%t('\VerbatimInput{output.lst}');
```

Running this will add the following text to the PDF:

```
_______________SAS_PROGRAM_BEGINS_________
188414  +PROC CORR DATA=sashelp.class NOSIMPLE NOPROB;
188415  +VAR height weight;
188416  +RUN;
188417  +%endprint;
_______________SAS_PROGRAM_ENDS___________

                     The CORR Procedure

               2  Variables:    Height   Weight


            Pearson Correlation Coefficients, N = 19
```

	Height	Weight
Height	1.00000	0.87779
Weight	0.87779	1.00000

It is also possible to save the programs into separate files and add them to the document using the LaTeX verbatim environments which are designed to typeset all text as it is.

CAPTURING FIGURES

Capturing a figure file created with SAS can be done as shown in the code example 8.

Code example 8: Capturing a graph from SAS

```
%t('\begin{figure}[ht]');
%t('\caption[SAS figure 1]{SAS figure 1}');
%t('\label{fig:sasfig}');
%t('\includegraphics{SAS_figure_1.eps}');
%t('\end{figure}');
```

Different versions of LaTeX/TeX usually accept only certain types of graph file formats.

CAPTURING RESULTS FROM SAS/IML

LaTeX typesets matrices in a very nice manner. The only downside is that the matrix syntax is quite cumbersome to write. Therefore I have programmed two SAS macros to tie the SAS/IML results and LaTeX typesetting closer together. The macros are called %matsave and %matconv.
In the following example, a linear model normal equation is typesetted with all the interim results based on imaginary X and y matrices. The SAS macro variable &mt is the matrix transpose symbol defined in the main program to make it possible to change the symbol easily.

Code example 9: Capturing matrices from SAS/IML, Equation (3)

```
PROC IML;
X={1 1 2 2, 1 3 1 4, 1 1 3 2, 1 3 4 4, 1 2 3 4};
y={4, 2, 3, 3, 4};
*Calculate all the interim results;
tX=t(X);
tXX=tX*X;
invtXX=inv(tXX);
invtXXtX=invtXX*t(X);
b=invtXXtX*y;
*Save all the calculated matrices;
%matsave(X y tX tXX invtXX invtXXtX b);
*Convert all the matrices to SAS macro variables;
%matconv(X y tX tXX invtXX invtXXtX b);
*Write all the matrices using the created macro variables;
%t('\begin{align}');
%t('(X^{&mt} X)^{-1} X^{&mt}y = ');
%t('\left(&X^{&mt} X \right)^{-1} X^{&mt}y = \\ ');
%t('\left( &tX &X \right)^{-1} X^{&mt}y = \\ ');
%t('&tXX ^{-1} &X^{&mt}y = \\ ');
%t('&invtXX &tX y = \\ ');
%t('&invtXXtX &y = &b = \hat b ');
%t('\end{align}');
```

The code example 9 typesets the equation (3) to this document.

$$
\begin{aligned}
(X^TX)^{-1}X^Ty = &\left(\begin{bmatrix}1 & 1 & 2 & 2\\ 1 & 3 & 1 & 4\\ 1 & 1 & 3 & 2\\ 1 & 3 & 4 & 4\\ 1 & 2 & 3 & 4\end{bmatrix}^T X\right)^{-1} X^Ty = \\
&\left(\begin{bmatrix}1 & 1 & 1 & 1 & 1\\ 1 & 3 & 1 & 3 & 2\\ 2 & 1 & 3 & 4 & 3\\ 2 & 4 & 2 & 4 & 4\end{bmatrix}\begin{bmatrix}1 & 1 & 2 & 2\\ 1 & 3 & 1 & 4\\ 1 & 1 & 3 & 2\\ 1 & 3 & 4 & 4\\ 1 & 2 & 3 & 4\end{bmatrix}\right)^{-1} X^Ty = \\
&\begin{bmatrix}5 & 10 & 13 & 16\\ 10 & 24 & 26 & 36\\ 13 & 26 & 39 & 42\\ 16 & 36 & 42 & 56\end{bmatrix}^{-1}\begin{bmatrix}1 & 1 & 2 & 2\\ 1 & 3 & 1 & 4\\ 1 & 1 & 3 & 2\\ 1 & 3 & 4 & 4\\ 1 & 2 & 3 & 4\end{bmatrix}^T y = \\
&\begin{bmatrix}3.8 & 0.8 & -0.4 & -1.3\\ 0.8 & 1.55 & 0.1 & -1.3\\ -0.4 & 0.1 & 0.2 & -0.1\\ -1.3 & -1.3 & -0.1 & 1.3\end{bmatrix}\begin{bmatrix}1 & 1 & 1 & 1 & 1\\ 1 & 3 & 1 & 3 & 2\\ 2 & 1 & 3 & 4 & 3\\ 2 & 4 & 2 & 4 & 4\end{bmatrix} y = \\
&\begin{bmatrix}1.2 & 0.6 & 0.8 & -0.6 & -1\\ -0.05 & 0.35 & 0.05 & 0.65 & -1\\ -0.1 & -0.3 & 0.1 & 0.3 & 0\\ -0.2 & -0.1 & -0.3 & -0.4 & 1\end{bmatrix}\begin{bmatrix}4\\ 2\\ 3\\ 3\\ 4\end{bmatrix} = \begin{bmatrix}2.6\\ -1.4\\ 0.2\\ 0.9\end{bmatrix} = \hat{b}
\end{aligned}
\tag{3}
$$

What happens within code example 9 is that the matsave macro first creates SAS datasets from the calculated IML matrices. The matconv macro is then used to convert these datasets to SAS macro variables containing each matrix with LaTeX syntax. After this, these macro variables can be used anywhere in the document to write out the whole matrix.

When written in the LaTeX syntax the content of the X matrix will be as follows:

```
\begin{bmatrix}1 & 1 & 2 & 2 \\1 & 3 & 1 & 4 \\1 & 1 & 3 & 2
\\1 & 3 & 4 & 4 \\1 & 2 & 3 & 4\end{bmatrix}
```

This string is stored as the value of macro variable &X after performing the matconv call. It is worth noting that X and y matrices are defined only once in the beginning of the IML procedure. Any update there will be reflected through the whole chain of results to the PDF.

CONDITIONAL TYPESETTING

Parts of the document can be conditionally typeset if the results calculated with SAS meet certain criteria. For example if a p-value is below the predefined α level, the next chapter which calculates the subgroup results can be automatically included into the document – and otherwise it will be left out. This is easy to achieve by storing the relevant SAS results to macro variables and executing the LaTeX code if the desired condition is met.

SOME GRAPH EXPERIMENTS USING PSTricks THROUGH SAS

LaTeX contains packages to allow very flexible creation of different kinds of plots. One of these is called PSTricks. With some interim steps it is for example possible to use SAS data to plot curves like done in figure 1. Combining any image with PSTricks drawing is also possible as seen in figure 2.

Figure 1: a Plot of Confusion

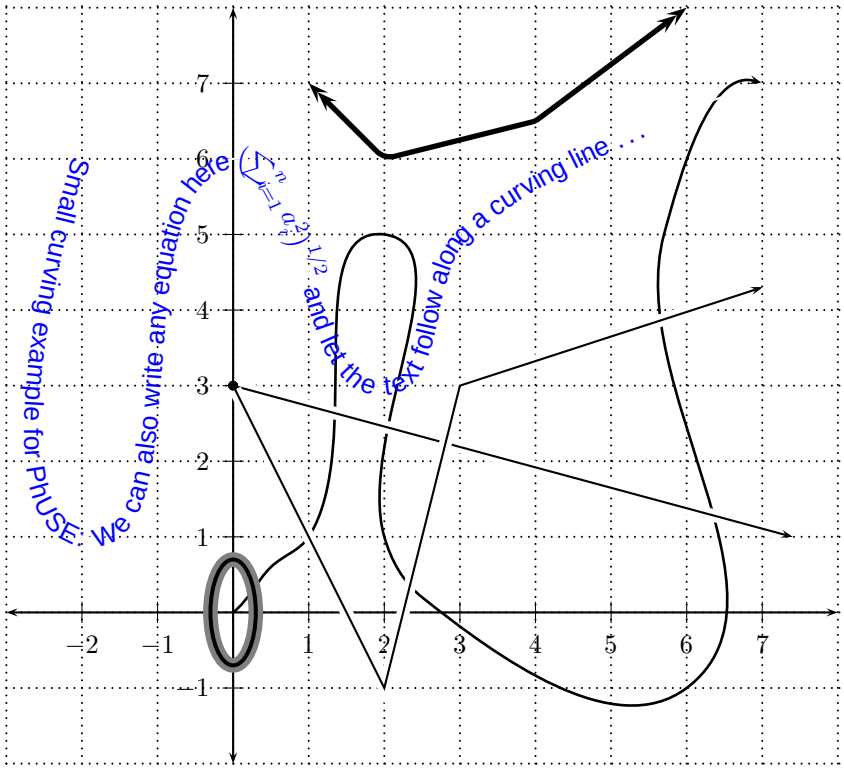
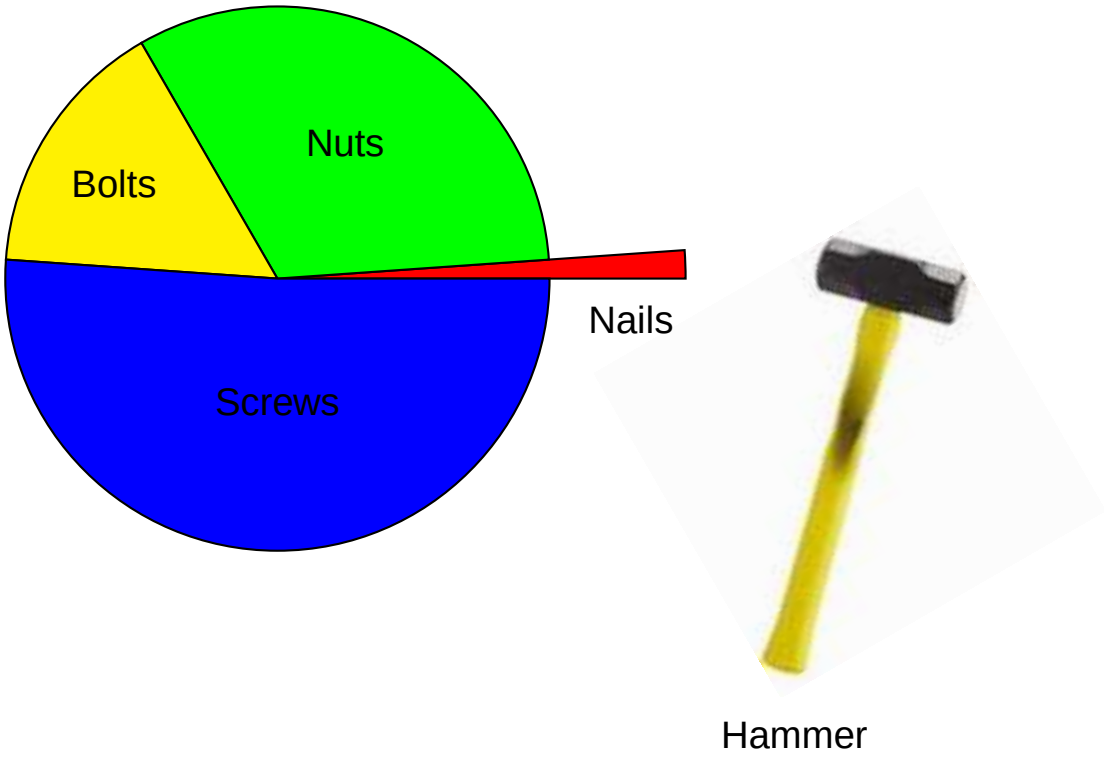


Figure 2: Home Improvement Sales Data



COMPENDIUM CONCEPT

Some time after programming the first versions of these macros I was introduced to a very nice article by two of the developers of R. Their paper gives a more theoretical framework to this kind of approach. *"We introduce the concept of a compendium as both a container for the different elements that make up the document and its computations (i.e. text, code, data, ...), and as a means for distributing, managing and updating the collection"* (Gentleman & Temple Lang 2004) .

SAS and LaTeX combined with a standardized folder structure can create and host simple compendiums. These compendiums can be used as containers for the different elements that make up the PDF document.

One of the benefits of combining SAS and LaTeX is that the creation of the resulting document can be taken to a level which the article describes as *"atomic action in which the inputs are synchronized"*. This ensures maximum reproducibility of the results and also by *"providing a well-defined structure that others can use directly makes it easier for both authors and readers to work with such documents".* With SAS and LaTeX it is possible to build a structure which on each run of the main program reproduces and reports the complete set of results without any manual phases.

To achieve even more flexible creation of output objects I have also created some ways to access other programs from SAS. This is what we are going to look at next.

CAPTURING RESULTS CREATED WITH R

One of the macros in the Include directory is called %r. This macro can be used to write and submit R programs through the SAS Program Editor. The results can then be collected to the output document in the usual way. The code example 10 shows how the r macro works.

Code example 10: r macro usage example

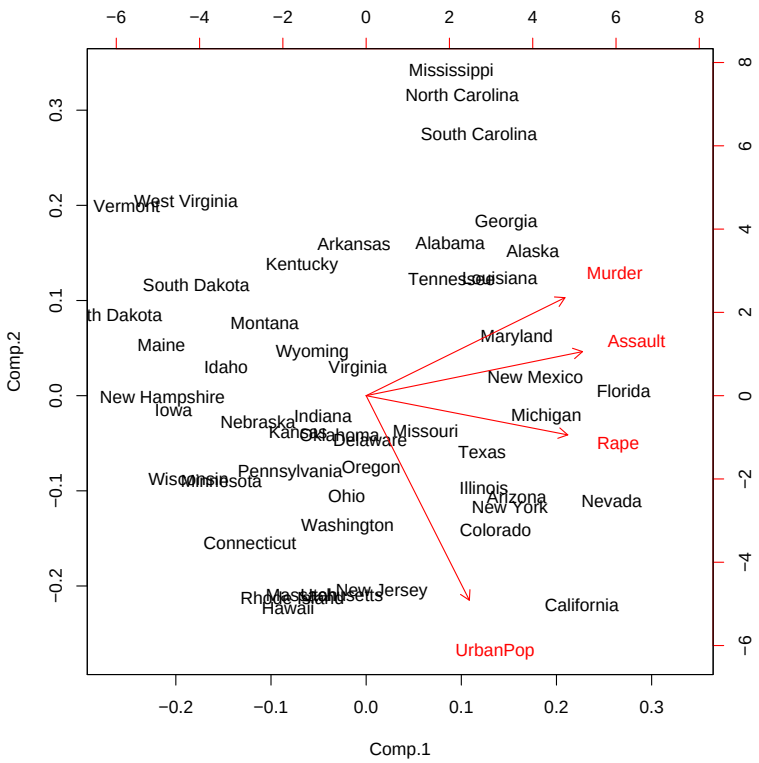
```
************************************************************;
*R program begins;
%r('data(USArrests);
pc.cr <- princomp(USArrests, cor=TRUE);
summary(pc.cr);
',rtest.lst);
*Draw plot;
%r('postscript(file = "biplot.eps");
biplot(pc.cr);
dev.off();');
*R program ends;
************************************************************;
*Include output to the document;
%t('\VerbatimInput{rtest.lst}');
*Include figure to the document;
%t('\begin{figure}[ht]');
%t('\caption[Test Plot from R]{Test Plot from R}');
%t('\label{fig:biplot}');
%t('\includegraphics[scale=0.55,angle=-90]{biplot.eps}');
%t('\end{figure}');
************************************************************;
```

This program will create the following output and figure 3, and add them to this document.

```
Importance of components:
                          Comp.1    Comp.2    Comp.3     Comp.4
Standard deviation     1.5748783 0.9948694 0.5971291 0.41644938
```

```
Proportion of Variance 0.6200604 0.2474413 0.0891408 0.04335752
Cumulative Proportion  0.6200604 0.8675017 0.9566425 1.00000000
```

Figure 3: Test Plot from R



In addition to the r macro I have created another macro called %s. The s macro can be used to write and submit Survo programs from SAS. Survo is a computing environment for statistical programming and adding a possibility to use it as well made it possible to show the results from three statistical packages side by side in the same document. Combining outputs this way can be useful for example in validation activities.

IMAGE MANIPULATION EXAMPLE BASED ON SAS DATA VALUES

ImageMagick is a command line picture manipulation program. Combined with Ghostscript it can for example transform images from vector types to bitmap types and vice versa. This is often necessary when working with LaTeX. ImageMagick can also apply many kinds of effects to images, figure 5 is an example of this.
Figure 4 is an EPS picture created with ImageMagick. Figure 5 is created from figure 4 by swirling it in five sections using ImageMagick based on SAS dataset values. Data defines the amount and direction of the swirl – a big positive value swirls a lot to the right and a negative one to the left. Many other types of manipulations are also possible. The data which the transformation is based on is generated with the SAS random number generator using a changing seed number. This causes the swirls to be different every time this document is recompiled.

CONCLUSION

Hopefully the examples given in this paper are convincing enough to show that SAS and LaTeX can be made to work nearly perfectly together. SAS is very powerful in calculating results and LaTeX at least

Figure 4: Original image

Figure 5: Swirled image

equally so in displaying them. Combining these two programs opens up new possibilities and with some additional tools also other programs can be used to create content from within SAS. Removing all manual phases from the creation of the resulting document minimizes the possibility of having out of date content and ensures the reproducibility of the results. Using SAS macro variables to define switches makes it easy to customize the PDF document for different kind of purposes.

When combining SAS and LaTeX in the way proposed in this paper, there are surprisingly few special characters which give trouble. Single quotes are ones which do and for this reason the use of special quote replacement character has been added to the t macro – but even with this addition the quoting still creates issues from time to time. One good future enhancement would be to make the user interface better. What happens now is that if an error occurs in the LaTeX execution phase it will be encountered as many times as LaTeX is called. Also sometimes it is quite difficult to debug errors and figure out the source of them as many software packages are used in parallel.

REFERENCES

Adriaens, H. & Ellison, C. (2005), 'The powerdot class', `ftp://tug.ctan.org/tex-archive/macros/latex/contrib/powerdot/doc/powerdot.pdf` .

Gentleman, R. & Temple Lang, D. (2004), 'Statistical analyses and reproducible research', *Bioconductor Project Working Papers, Working Paper 2,* `www.bepress.com/bioconductor/paper2/` .

Knuth, D. E. (1986), 'Computers & typesetting, volume A: The TeXbook', *Addison Wesley Professional, Boston* .

Lamport, L. (1994), 'LaTeX: A document preparation system, 2nd edition', *Addison Wesley Professional, Boston* .

Oetiker, T., Partl, H., Hyna, I. & Schlegl, E. (2008), 'The not so short introduction to $LaTeX2_\varepsilon$', `http://tobi.oetiker.ch/lshort/lshort.pdf` .

ACKNOWLEDGEMENTS

I would like to thank Kimmo Vehkalahti for the great discussions on developing more automated ways of working and all the contributors of TeX, LaTeX and MiKTeX for their magnificent work.

CONTACT INFO

You can contact the author using this email address: `juha-pekka.perttola@roche.com`