SAS - tricks and tips

Carl James Schwarz
Simon Fraser University
Burnaby, BC V5A 1s6
cschwarzstat.sfu.ca

December 10, 2012

Contents

1 Introduction **2**

1.1 Where do I get *SAS* at SFU? 2

1.2 Online Resources for *SAS* 2

1.3 Why does double clicking on a *.SAS file launch the *SAS* Enterprise product? 3

2 Using the Editor **4**

2.1 Setting the directory where programs and output will be stored 4

2.2 Making nice looking code 4

2.3 Using the DM commands to clear the log and output windows 5

2.4 Deleting working datasets 6

2.5 Running *SAS* in batch mode 6

3 Reading data into *SAS*, especially dates and times **7**

4 Sending *SAS* output to various destinations **8**

4.1 Creating a PDF file of a job and/or procedure 8

4.2 Creating RTF files of a job and/or procedure 9

4.3 *SAS* output into LaTeX 9

4.3.1 PDF output into LaTeX 10

4.3.2 Selected *SAS* Tables into LaTeX - I 14

4.3.3 Selected *SAS* Tables into LaTeX - II 19

4.3.4 Creating Graphs (GPLOT, etc.) for inclusion into LaTeX 21

4.3.5 Selected *SAS* Graphs into LaTeX - II 23

4.3.6 Modifying the LaTeX tagset 27

5 Controlling the amount of output **32**

6 Should I use Proc GPLOT or Proc SGPLOT **33**

Chapter 1

Introduction

I like *SAS*! As many of my former students can attest, I find the use of *SAS* to be very efficient in the use of my time. I like its data management capabilities, its wide range of procedures, and its ability to organize output in useful ways that reduces my need to do extensive reformatting for publication etc.

This document contains an eclectic selection of tips and tricks that I've gleaned through the years in working with *SAS*. Suggestions and comments are welcome – my email address is on the title page.

All examples in this document have been tested using *SAS* 9.3 or later.

1.1 Where do I get *SAS* at SFU?

SAS is available from `http://www.sfu.ca/itservices/technical/software.html`. You will need an SFU Computing Id to obtain the software.

1.2 Online Resources for *SAS*

You can try

- `http://people.stat.sfu.ca/~cschwarz/CourseNotes/`. A collection of course notes that illustrate basic statistical analyses using *JMP*, *R*, and *SAS*.
- `http://people.stat.sfu.ca/~cschwarz/Stat-650/Notes/MyPrograms/`. A collection of sample programs in *JMP*, *R*, and *SAS*.

- `http://www.ats.ucla.edu/stat/sas/`. Collection of basic *SAS* information, tips, and tricks.
- `http://www.datavis.ca/sasmac/`. Collection basic *SAS* information, tip, and tricks along with more advanced features.
- `http://www.sascommunity.org/wiki/Main_Page`. A *SAS* Wiki.

1.3 Why does double clicking on a *.SAS file launch the *SAS* Enterprise product?

After installing the *SAS* system, if you use all of the defaults, you have a partially functional *SAS* system. For some reason, when you double-click a *.SAS file, it defaults to open with the Enterprise Guide product, So right after installation, if you want *.SAS files to open in *SAS*, do this:

> Go to Start Menu-> *SAS*-> Utilities-> Manage *SAS* File Types.
> Click on "SAS Foundation" and select all file types. Hit OK.

Chapter 2

Using the Editor

2.1 Setting the directory where programs and output will be stored

Click on the bottom right of the editor or right click on a *SAS* program will set the directory where *SAS* looks for external files and saves its output.

2.2 Making nice looking code

I find it easier to read and debug *SAS* code if a consistent indentation pattern is used. My standard format is to keep the start of each procedure and corresponding run left flush and indent the statements within the procedure. Use (several) blank lines to separate procedures. For example, I find

```
data xyx;
   input a b d e;
   datalines;
   1 2 3 4
   4 5 6 7
;;;;
run;

proc print dat=xyx;
   var a b c e;
run;
```

much easier to read and figure out than

```
data xyx;
input a b d e;
datalines;
1 2 3 4
4 5 6 7
;;;;
run;
proc print dat=xyx;
var a b c e;
run;
```

An unfortunate side effect of indenting is that the *SAS* editor tends to use tabs and/or spaces to do the indenting and may not be consistent, i.e., sometimes uses a tab and sometimes uses a space. Consequently, your nicely indented program may look less than pretty when opened at a later data as the tabs stops may change.

I find it convenient to turn off using tabs and insist that *SAS* use spaces for all white space. The following is extracted from the *SAS* help files on using the editor.

> **Using Automatic Indenting and Tabs**. When you press ENTER, you automatically indent the next line by the amount of space that the current line is indented. If you prefer not to use automatic indention:
>
> Select *Tools* → *Options* → *Enhanced Editor* → *General* and in the Indention box, select None.
>
> In addition to automatic indenting, you can indent by using the TAB key. Pressing the TAB key moves the insertion point and any text to the right of the insertion point by the amount of space that you specified in the Tab size field of the Enhanced Editor Options dialog box. Tab characters can be replaced by spaces either when you press the TAB key or when you open a file. To insert spaces instead of tab characters when you press the TAB key, select the Insert spaces for tabs check box. To replace tab characters with spaces when you open a file, select the Replace tabs with spaces on file open check box. Note: Changing the tab size modifies tab settings to the new value in all Enhanced Editor windows.

I find it useful to select both options so that Tab characters are NOT inserted into program files.

2.3 Using the DM commands to clear the log and output windows

When you run *SAS* interactively the log and output window keep track of error messages and output. However, these destinations are not cleared until you quit *SAS*.

I find it convenient to clear the log and output windows while debugging a program so that only the output and log from the most recent run are available. Consequently, I always include the following statements at the start of my *SAS* programs;

```
dm output 'clear';
dm log    'clear';
```

These command have no effect when running *SAS* in batch model.

2.4 Deleting working datasets

When debugging *SAS* programs, you will often run the same piece of code several times. Working datasets will stay around between runs. This can lead to problem where an old copy of a working dataset is used in place of a new version (that may not have been created because of a bug in the *SAS* code). Consequently, I find it useful to delete all working datasets at the start of the code each time it is run using:

```
proc datasets kill; /* delete working datasets*/
run;
```

2.5 Running *SAS* in batch mode

SAS can be run either in interactive or batch mode. In interactive mode, you select the sections of the *SAS* program (or the entire program) to run and submit the statements. You then need to wait while *SAS* processes those statements.

For long simulations, it is often more convenient to run *SAS* as a background process as a batch job. To submit a job to the batch processor, right click on the file name and select the batch mode. *SAS* will then run until the job is complete. A *.log and *.lst file will be created (along with any other file requested) to hold the output from the log and the listing destinations, respectively.

Chapter 3

Reading data into *SAS*, especially dates and times

I won't go into the many, many, many ways in which *SAS* can read in data – please consult any number of *SAS* books on this topic.

A very short introduction is available in the *ReadingData.sas* file available at `http://people.stat.sfu.ca/~cschwarz/Stat-650/Notes/MyPrograms/SASTricks/SampleSASPrograms` directory.

Chapter 4

Sending *SAS* output to various destinations

SAS generated much output and getting the output into various formats is very useful when creating reports.

4.1 Creating a PDF file of a job and/or procedure

To create a PDF file of a job and/or a procedure, bracket the section of code for which you want to capture the output (tables and graphs) with:

```
options orientation=landscape;
ods pdf file=’myfilename.pdf’;
goptions device=pdf colors=(black) rotate=landscape;
  ...
  \SAS\ procedures
   ...
ods pdf close;
```

For example, the code fragment:

```
options orientation=landscape;
ods pdf file=’output-to-LaTeX-genmod.pdf’;
goptions device=pdf colors=(black) rotate=landscape;
ods graphics on;
```

```
proc genmod data=seeds plots=all;
   title2 'overdispersion model';
   class cult soil;
   model r/n = cult soil cult*soil / dist=binomial link=logit scale=deviance type3;
   lsmeans cult soil cult*soil / diff;
run;
ods graphics off;
ods pdf close;
```

send the entire output of the procedure to a pdf file.

4.2 Creating RTF files of a job and/or procedure

RTF (Rich Text Files) are useful if you want to use the *SAS* output in in Microsoft Word (or equivalent) document. Use the ODS RTF destination with:

```
options orientation=landscape;
ods rtf file='myfilename.rtf';
goptions device=emf colors=(black) rotate=landscape;
  ...
  \SAS\ procedures
   ...
ods rtf close;
```

Don't forget to change the graphics option device to EMF (Extended Meta Format) which is a recommended way to extract material for RTF files – refer to `http://support.sas.com/techsup/technote/ts674/ts674.html` for more details.

The above code fragment will create a rtf file that can then be opened in your word processor and the graphs and tables extracted.

4.3 *SAS* output into LaTeX

Getting *SAS* output into LaTeX is relatively easy, but can be surprising frustrating in getting the last "10% correct!". The key to success is using the Output Deliver System (ODS) feature of *SAS*.

The output from a *SAS* procedure can be divided into two types – tables and graphs. Details on including both types of output are provided in the sections below.

A worked example is the *output-to-LaTeX.sas* program available in the `http://www.stat.sfu.ca/~cschwarz/Stat-650/Notes/MyPrograms`. The sample program looks at different ways to analyze a generalized linear mixed model. The actual analysis is not that important, and we concentrate here on showing how to integrate the various pieces of output into *SAS*.

One of the analyses in the program uses *Proc Genmod* to fit a generalized linear model allowing for overdispersion.

4.3.1 PDF output into LaTeX

At the coarsest level, the entire output from a procedure (or several procedures) can be sent to a pdf file and then this pdf file can included into your document directly.

The code to send the entire output to a pdf file is:

```
options orientation=landscape;
ods pdf file=’output-to-LaTeX-genmod.pdf’;
goptions device=pdf colors=(black) rotate=landscape;
ods graphics on;
proc genmod data=seeds plots=all;
   title2 ’overdispersion model’;
   class cult soil;
   model r/n = cult soil cult*soil / dist=binomial link=logit scale=deviance type3;
   lsmeans cult soil cult*soil / diff;
run;
ods graphics off;
ods pdf close;
```

This creates the file *output-toLaTeX-genmod.pdf* which is several pages long and can be viewed in the directory that contains the sample program.

The package *pdfpages* enable whole pages of pdf documents to be inserted verbatim into the document. Include the command

```
\usepackage{pdfpages}
```

into the preamble of the document, and use the \includepdf command to insert all or selected pages into the document. For example, the command

```
\includepdf[landscape=true,page=1-2]{output-to-LaTeX-genmod.pdf}
```

includes the first two pages of the pdf output in the document as shown on the next page. If you want all pages to be includes, use *pages*=-. The file name used for the pdf file should NOT contain any other periods other than before the suffix.

Example of overdispersion with random effects in logistic regression
overdispersion model

The GENMOD Procedure

Model Information	
Data Set	WORK.SEEDS
Distribution	Binomial
Link Function	Logit
Response Variable (Events)	r
Response Variable (Trials)	n

Number of Observations Read	20
Number of Observations Used	20
Number of Events	437
Number of Trials	906

Class Level Information		
Class	**Levels**	**Values**
cult	2	0 1
soil	2	0 1

Response Profile		
Ordered Value	**Binary Outcome**	**Total Frequency**
1	Event	437
2	Nonevent	469

The GENMOD Procedure

Parameter Information			
Parameter	**Effect**	**cult**	**soil**
Prm1	Intercept		
Prm2	cult	0	
Prm3	cult	1	
Prm4	soil		0
Prm5	soil		1
Prm6	cult*soil	0	0
Prm7	cult*soil	0	1
Prm8	cult*soil	1	0
Prm9	cult*soil	1	1

Criteria For Assessing Goodness Of Fit			
Criterion	**DF**	**Value**	**Value/DF**
Deviance	16	68.3465	4.2717
Scaled Deviance	16	16.0000	1.0000
Pearson Chi-Square	16	66.7619	4.1726
Scaled Pearson X2	16	15.6291	0.9768
Log Likelihood		-141.0465	
Full Log Likelihood		-74.2666	
AIC (smaller is better)		156.5333	
AICC (smaller is better)		159.2000	
BIC (smaller is better)		160.5162	

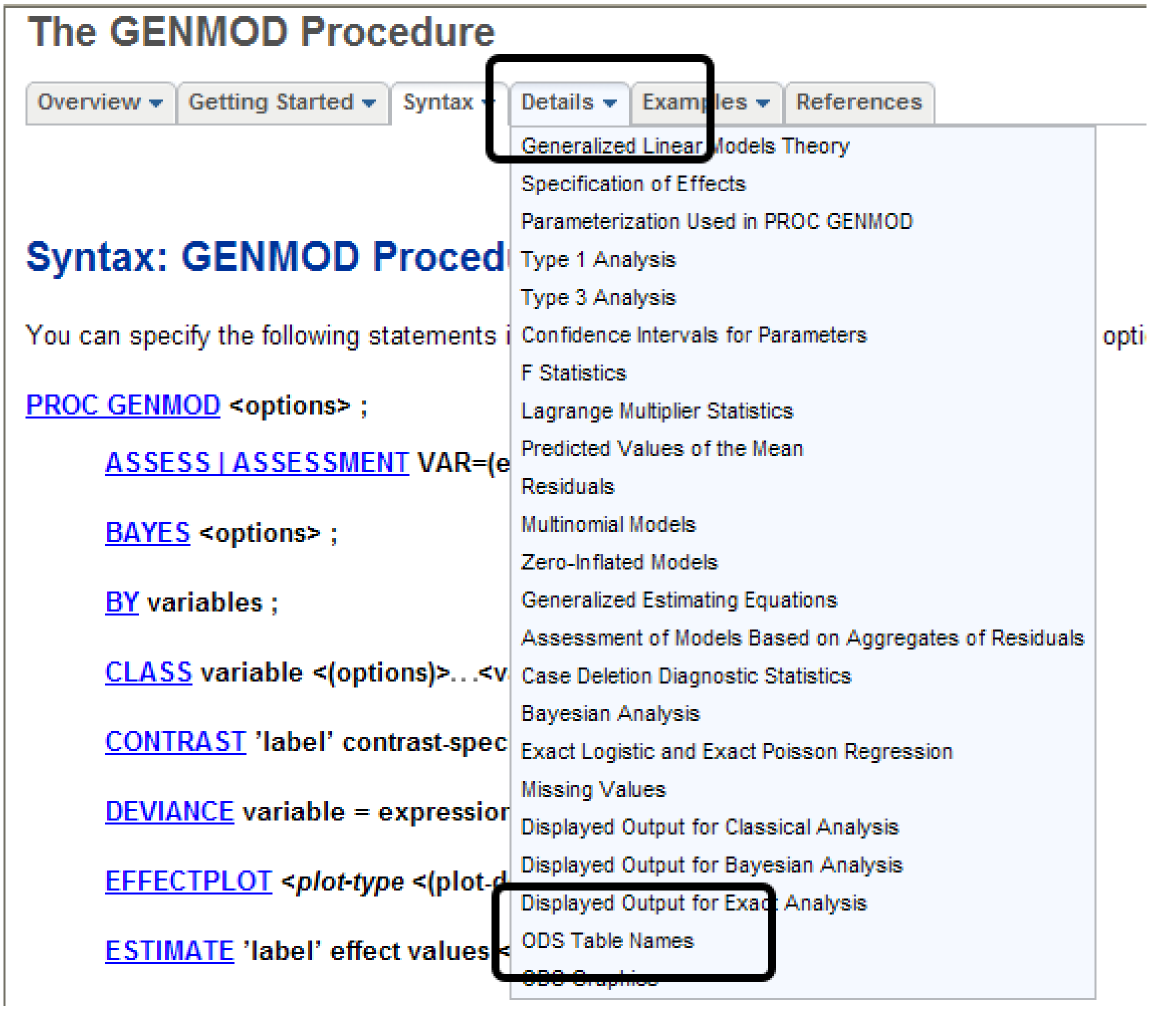
4.3.2 Selected *SAS* Tables into LaTeX - I

The previous section enables a dump of *SAS* output into your document, but does not provide much control on which output can be displayed. You could use the *ods select* statement to select which tables should be displayed in the pdf file, but you still must include a full page of a pdf document.

Because all tabular output in *SAS* is actually a table, a method to incorporate selected tables into your document is to send the relevant table to a *SAS* dataset, and then print the *SAS* dataset, but “ask” *SAS* to create a LaTeX file that can be included directly into your document.

For example, suppose we want the last table on the second page of the pdf file (the *Criteria for Assessing Goodness of Fit* table).

You first need to find the name of the table that is created by *Proc Genmod*. You can do this by looking the help documentation for Genmod :



The GENMOD Procedure

Overview
Getting Started
Syntax
Details
Examples
References
Generalized Linear Models Theory
Specification of Effects
Parameterization Used in PROC GENMOD
Type 1 Analysis
Type 3 Analysis
Confidence Intervals for Parameters
F Statistics
Lagrange Multiplier Statistics
Predicted Values of the Mean
Residuals
Multinomial Models
Zero-Inflated Models
Generalized Estimating Equations
Assessment of Models Based on Aggregates of Residuals
Case Deletion Diagnostic Statistics
Bayesian Analysis
Exact Logistic and Exact Poisson Regression
Missing Values
Displayed Output for Classical Analysis
Displayed Output for Bayesian Analysis
Displayed Output for Exact Analysis
ODS Table Names
ODS Graphics

Syntax: GENMOD Proced

You can specify the following statements i opti

PROC GENMOD <options> ;

ASSESS | ASSESSMENT VAR=(e

BAYES <options> ;

BY variables ;

CLASS variable <(options)>. . .<v

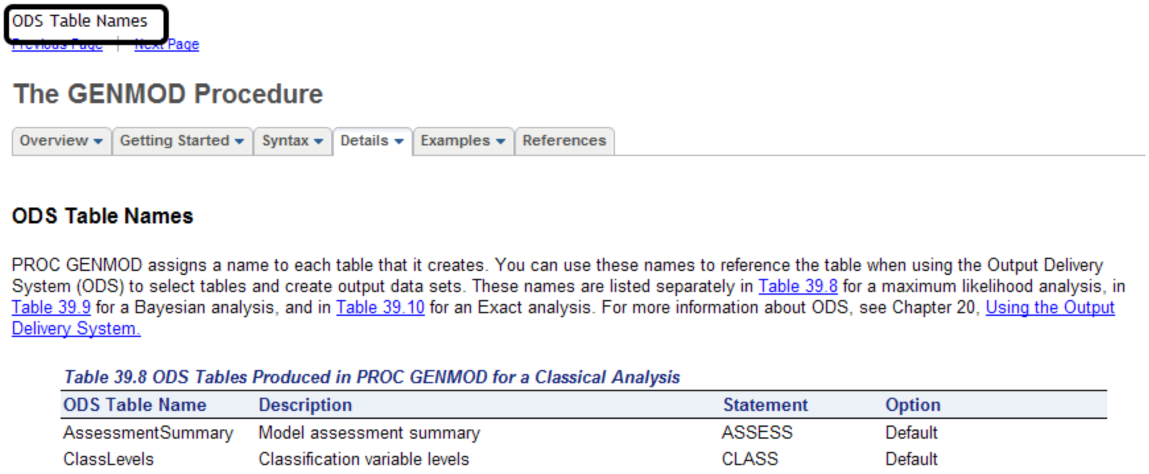
CONTRAST 'label' contrast-spec

DEVIANCE variable = expressior

EFFECTPLOT <*plot-type* <(plot-d

ESTIMATE 'label' effect values <

Select the *Details Tab* and look for the table that contains the the table of interest.



The GENMOD Procedure

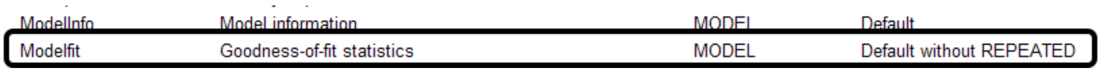
ODS Table Names

PROC GENMOD assigns a name to each table that it creates. You can use these names to reference the table when using the Output Delivery System (ODS) to select tables and create output data sets. These names are listed separately in Table 39.8 for a maximum likelihood analysis, in Table 39.9 for a Bayesian analysis, and in Table 39.10 for an Exact analysis. For more information about ODS, see Chapter 20, Using the Output Delivery System.

Table 39.8 ODS Tables Produced in PROC GENMOD for a Classical Analysis

ODS Table Name	Description	Statement	Option
AssessmentSummary	Model assessment summary	ASSESS	Default
ClassLevels	Classification variable levels	CLASS	Default

We find that the table name for the goodness-of-fit is called *ModelFit*. Note that in some cases, you must specify some additional options in the procedure (see right most column) in order to get the ods table created.



ModelInfo	Model information	MODEL	Default
Modelfit	Goodness-of-fit statistics	MODEL	Default without REPEATED

We use the *ods output TableName=YourDataSetName;* command to send the selected table to a *SAS* dataset. For example:

```
proc genmod data=seeds;
   title2 'overdispersion model';
   class cult soil;
   model r/n = cult soil cult*soil / dist=binomial link=logit scale=deviance type3;
   lsmeans cult soil cult*soil / diff;
   ods output ModelFit=GenmodModelFit;
run;
```

sends the goodness-of-fit table to the *SAS* dataset called *GenmodModelFit*. [Note that in *SAS* dataset names are not case sensitive.] You can have as many of the *ods output* statements as you want in each procedure and each sends a selected table to a dfferent *SAS* dataset.

Once you have the table in a *SAS* dataset, you can use *Proc Print* to print the dataset and send the output to a variety of destinations. Here we want to create LaTeX output that can be included into our document.

This is done using *tagsets* – special files (created by *SAS*) that describe how files should be "marked up" when output. There are 4 LaTeX tagsets provided by *SAS* as outlined at `http://support.sas.com/rnd/base/ods/odsmarkup/latex.html`

For example, the code fragment:

```
ods tagsets.latex file="output-to-LaTeX-fragment01.tex" (notop nobot) stylesheet="sas.s
proc print data=GenmodModelFit ;
title ;
run;
ods tagsets.latex  close;
```

creates the following LaTeX code fragment (only the first 20 lines are shown)

```
\sascontents[1]{The Print Procedure}
\sascontents[2]{Data Set WORK.GENMODMODELFIT}


\begin{sastable}[c]{rlrrr}\hline
   \multicolumn{1}{|S{header}{r}|}{Obs} &
   \multicolumn{1}{|S{header}{l}|}{Criterion} &
   \multicolumn{1}{|S{header}{r}|}{DF} &
   \multicolumn{1}{|S{header}{r}|}{Value} &
   \multicolumn{1}{|S{header}{r}|}{ValueDF}
\\\hline
\endhead
   \multicolumn{1}{|S{rowheader}{r}|}{1} &
   \multicolumn{1}{|S{data}{l}|}{Deviance} &
   \multicolumn{1}{|S{data}{r}|}{  16} &
   \multicolumn{1}{|S{data}{r}|}{     68.3465} &
   \multicolumn{1}{|S{data}{r}|}{      4.2717}
\\\hline
   \multicolumn{1}{|S{rowheader}{r}|}{2} &
   \multicolumn{1}{|S{data}{l}|}{Scaled Deviance} &
```

The *notop* and *nobot* options on the *tagsets* command informs *SAS* not to generate a preamble or following material before or after the code fragment.

This generated LaTeX code contains many special macros that are defined in the *SAS* style file which was generated using the style option on the *tagsets* command. This style sheet should be included in your document using the *\usepackage{SAS}* command. Then when you use the

```
\input{output-to-LaTeX-fragment01.tex}
```

command, the following table output is produced:

Obs	Criterion	DF	Value	ValueDF
1	Deviance	16	68.3465	4.2717
2	Scaled Deviance	16	16.0000	1.0000
3	Pearson Chi−Square	16	66.7619	4.1726
4	Scaled Pearson X2	16	15.6291	0.9768
5	Log Likelihood	_	−141.0465	_
6	Full Log Likelihood	_	−74.2666	_
7	AIC (smaller is better)	_	156.5333	_
8	AICC (smaller is better)	_	159.2000	_
9	BIC (smaller is better)	_	160.5162	_

Note that you may have to run LaTeX multiple times (as many as 4 times!) to get the table to format properly.

The entire output from a procedure can be sent to LaTeX by placing the *ods tagset* commands before and after the procedure, but I find it much more useful to select the parts of the procedure tables using the *ods output* command to be printed.

Now that the table has been extracted, you can improve the appearance of the output generated. For example, I only want to see the two chi-squared statistics, I don't need the observation number printed on the left side (remove using the *noobs* argument of *Proc Print*), and I want the default labels for each variable to be printed.

The code fragment

```
ods tagsets.latex file="output-to-LaTeX-fragment02.tex" (notop nobot) ;
proc print data=GenmodModelFit noobs label split=’ ’ ;
where df ^= . and df ^= ._  and index(criterion,’Scaled’)=0;
run;
ods tagsets.latex  close;
```

does this, with the result:

Criterion	DF	Value	Value/DF
Deviance	16	68.3465	4.2717
Pearson Chi−Square	16	66.7619	4.1726

The different tagsets produce different types of LaTeX code. For example, the *simplelatex* and *tablesonly-*

latex produce pure LaTeX code that does not require the *SAS* style file, but the output looks “undistinguished”. For example, the code fragment:

```
ods tagsets.simplelatex file="output-to-LaTeX-fragment03.tex" (notop nobot) ;
proc print data=GenmodModelFit noobs label split=’ ’ ;
where df ^= . and df ^= ._  and index(criterion,’Scaled’)=0;
run;
ods tagsets.simplelatex  close;
```

produces

Criterion	DF	Value	Value/DF
Deviance	16	68.3465	4.2717
Pearson Chi-Square	16	66.7619	4.1726

4.3.3 Selected *SAS* Tables into LaTeX - II

A second method used the *ODS DOCUMENT* feature. Here, the entire procedure is stored in a document store, and is replayed later, again using the appropriate tagset. The key difference is that you can construct much more elaborate output objects (which is beyond the scope of this document). For simple inclusion of LaTeX output this method is likely too cumbersome. However, it is the only way to access some of the graphics that are automatically generated by some procedures.

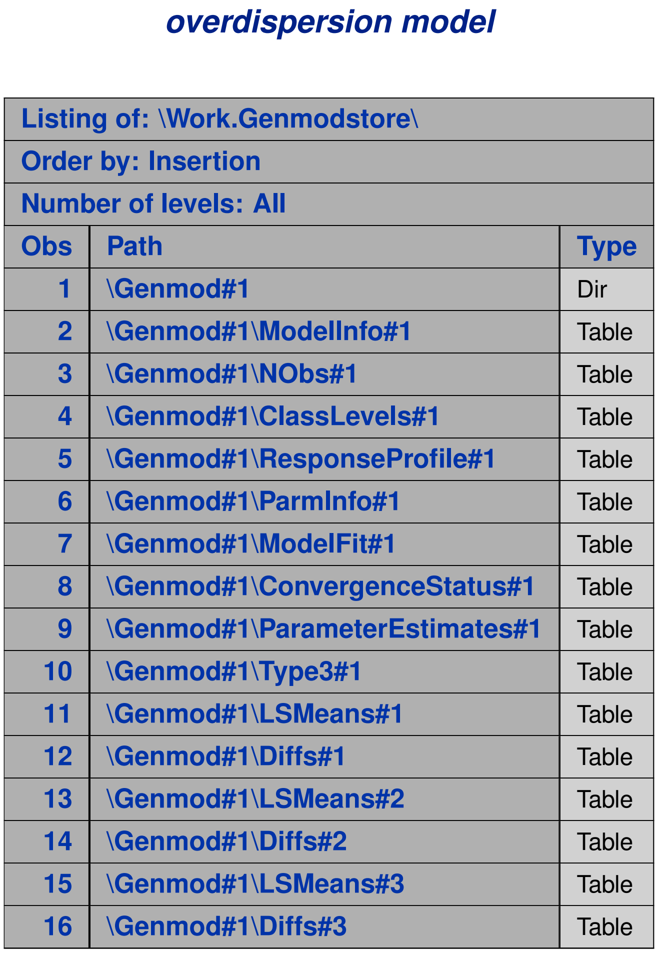
We start by creating a document store of the procedure. I find it easiest to create a separate document store for each procedure, but you can using the same document store for many procedures (indeed for the entire job).

```
ods document name=GenModStore(write);
proc genmod data=seeds plots=all;
   title2 ’overdispersion model’;
   class cult soil;
   model r/n = cult soil cult*soil / dist=binomial link=logit scale=deviance type3;
   lsmeans cult soil cult*soil / diff;
   ods output ModelFit=GenmodModelFit;
   ods output LSmeans =GenmodLSmeans;
run;
ods document close;
```

This creates a document store (*GenModStore*) in which all the output from the procedure is stored. We need to again find the name of the element in the document store that contains the output we want. This is done using:

```
proc document name=GenModStore;
   list / levels=all;
run;
```

which generates a complete listing of all the objects created by the procedure:



overdispersion model

Listing of: \Work.Genmodstore\		
Order by: Insertion		
Number of levels: All		
Obs	Path	Type
1	\Genmod#1	Dir
2	\Genmod#1\ModelInfo#1	Table
3	\Genmod#1\NObs#1	Table
4	\Genmod#1\ClassLevels#1	Table
5	\Genmod#1\ResponseProfile#1	Table
6	\Genmod#1\ParmInfo#1	Table
7	\Genmod#1\ModelFit#1	Table
8	\Genmod#1\ConvergenceStatus#1	Table
9	\Genmod#1\ParameterEstimates#1	Table
10	\Genmod#1\Type3#1	Table
11	\Genmod#1\LSMeans#1	Table
12	\Genmod#1\Diffs#1	Table
13	\Genmod#1\LSMeans#2	Table
14	\Genmod#1\Diffs#2	Table
15	\Genmod#1\LSMeans#3	Table
16	\Genmod#1\Diffs#3	Table

Looking though the table list, we see that the table of interest is the *Genmod#1\ModelFit#1* table. We use the replay feature of the document feature, along with the appropriate tagsets to select and replay the table.

```
ods tagsets.latex file="output-to-LaTeX-fragment32.tex" (notop nobot) ;
proc document name=GenModStore;
   replay \Genmod#1\ModelFit#1 ;
run;
ods tagsets.latex close;
run;
```

which gives

overdispersion model

The GENMOD Procedure

Criteria For Assessing Goodness Of Fit			
Criterion	**DF**	**Value**	**Value/DF**
Deviance	16	68.3465	4.2717
Scaled Deviance	16	16.0000	1.0000
Pearson Chi-Square	16	66.7619	4.1726
Scaled Pearson X2	16	15.6291	0.9768
Log Likelihood		−141.0465	
Full Log Likelihood		−74.2666	
AIC (smaller is better)		156.5333	
AICC (smaller is better)		159.2000	
BIC (smaller is better)		160.5162	

It is possible to select which lines of the table to print, but this is clumsy and the method in the previous table is likely easier.

4.3.4 Creating Graphs (GPLOT, etc.) for inclusion into LaTeX

Plain LaTeX only allows a limited number of graphic formats to be imported (usually postscript (*.ps) or encapsulated postscript (*.eps)). If you are using *pdflatex* to generate the pdf files directly, you can also include *.png files. I find these most useful for LaTeX.

The following code-fragment creates a png file from Proc Gplot:

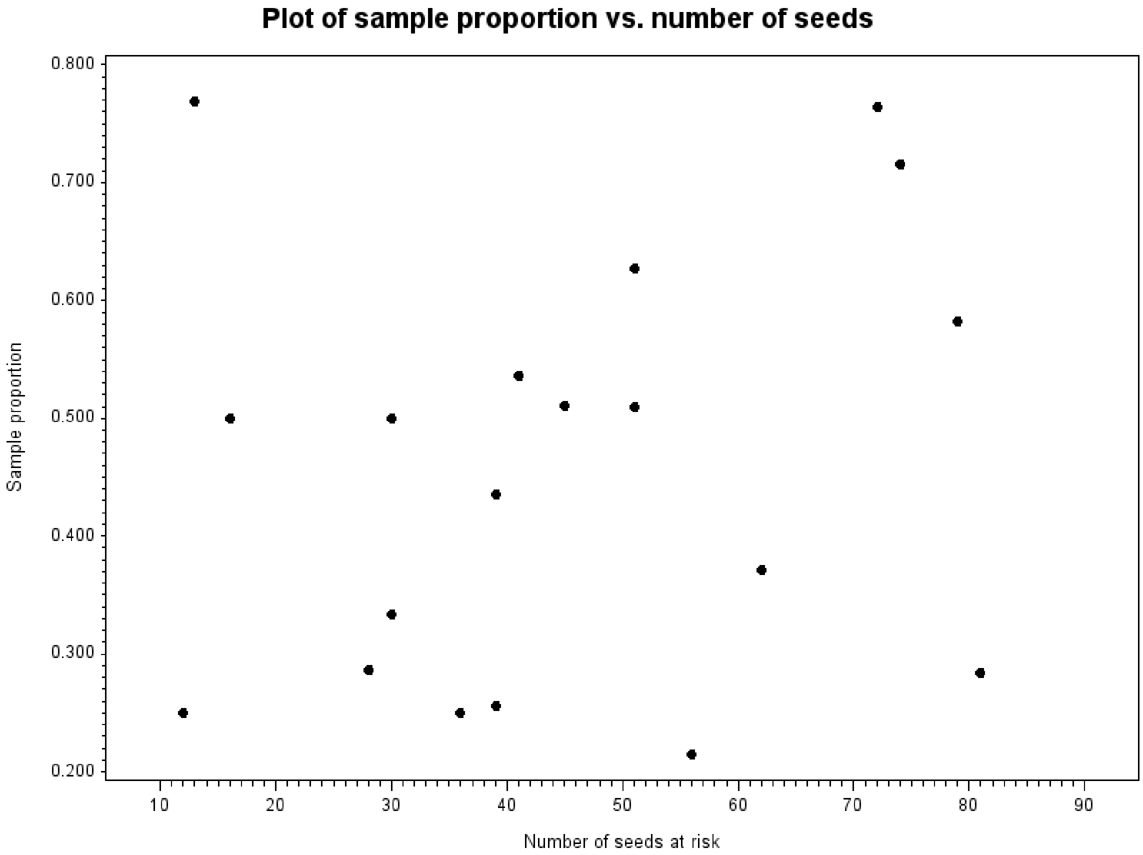
```
ods listing;
filename graphout "output-to-LaTeX-graph02.png";
goptions device=png gsfname=graphout;
goptions reset=symbol;
proc gplot data=seeds;
   title ’Plot of sample proportion vs. number of seeds’;
   axis1 label=(a=90 r=0 ’Sample proportion’);
   axis2 label=(         ’Number of seeds at risk’) offset=(1 cm, 1 cm);
   plot phat*n=1 / vaxis=axis1 haxis=axis2;
   symbol1 v=dot;
run;
ods listing close;
```

Notice that the *gsfname=* options points to a filename (called graphout) that, in turn, points to the actual file that will hold the graphics output. You cannot specify the file name directly in the gsfname= option.

After the *.png file is created, you include in LaTeX in the usual way:

```
\includegraphics[width=\textwidth]{MyPrograms/SASTricks/output-to-LaTeX-graph02}
```

which gives:



You can add options to the goptions statement to control the text size on the axes, the resolution and other attributes. More details are available at: `http://support.sas.com/techsup/technote/ts674/ts674.html`.

4.3.5 Selected *SAS* Graphs into LaTeX - II

Many of the *SAS* procedures produce automatic graphs when the *ODS GRAPHICS* feature is turned on. For example,

```
options orientation=landscape;
ods pdf file='output-to-LaTeX-genmod.pdf';
goptions device=pdf colors=(black) rotate=landscape;
ods graphics on;
proc genmod data=seeds plots=all;
   title2 'overdispersion model';
   class cult soil;
   model r/n = cult soil cult*soil / dist=binomial link=logit scale=deviance type3;
   lsmeans cult soil cult*soil / diff;
run;
ods graphics off;
ods pdf close;
```

creates many diagnostic graphs and sends them to the pdf file. We would like to extract some of the graphs and include in our document. Notice that unlike the previous example, we don't have code to generate the plots (i.e. there is no *gplot* statement).

We again used the *ODS DOCUMENT* storage feature.

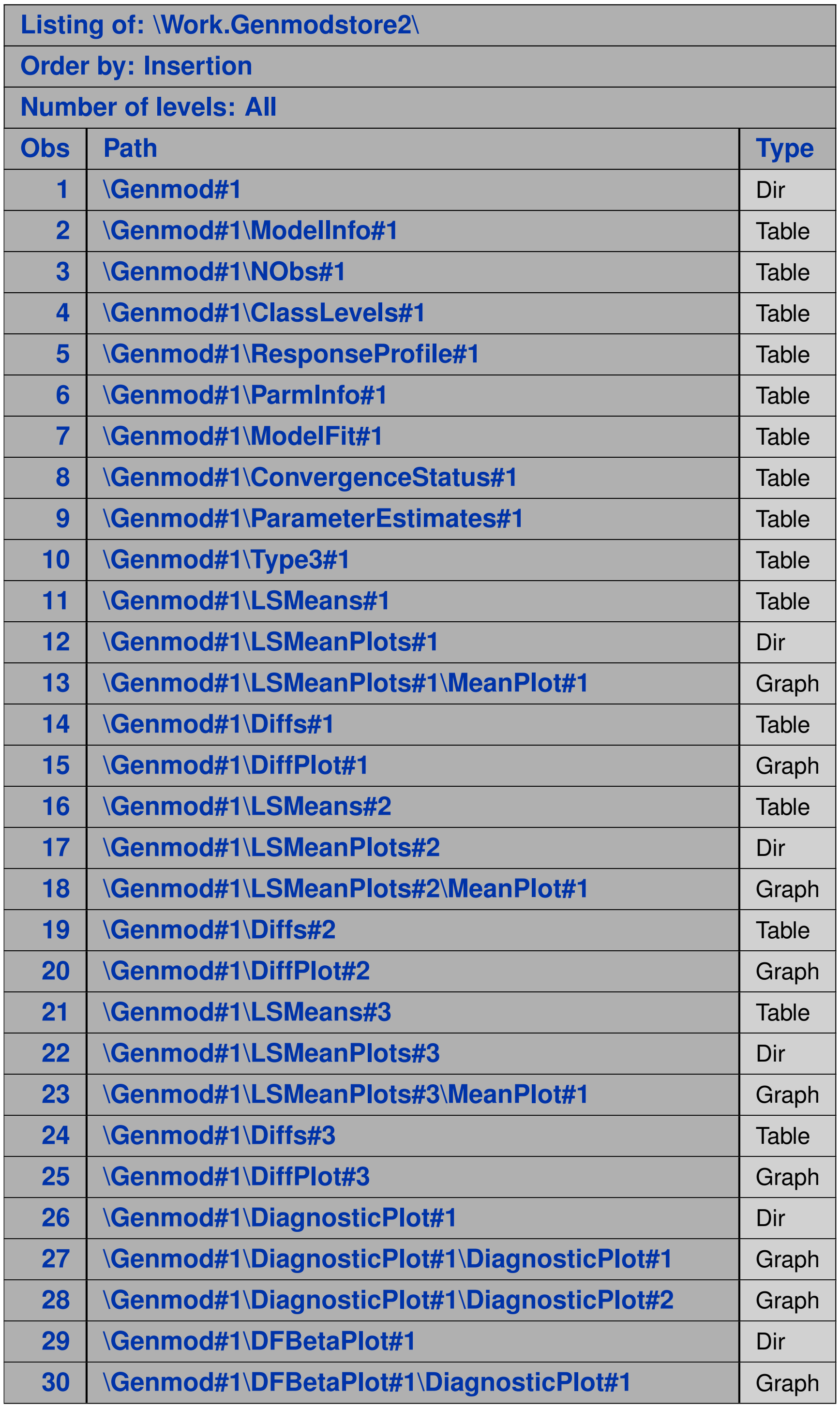
```
ods document name=GenModStore2(write);
ods graphics on;
proc genmod data=seeds plots=all;
   title2 'overdispersion model';
   class cult soil;
   model r/n = cult soil cult*soil / dist=binomial link=logit scale=deviance type3;
   lsmeans cult soil cult*soil / diff;
   ods output ModelFit=GenmodModelFit;
run;
ods graphics off;
ods document close;
```

This creates a document store (*GenModStore2*) in which all the tabular and graphical output from the procedure is stored. We need to again find the name of the element in the document store that contains the output we want. This is done using:

```
proc document name=GenModStore2;
   list / levels=all;
run;
```

which generates a complete listing of all the objects created by the procedure:

overdispersion model



Listing of: \Work.Genmodstore2\

Order by: Insertion

Number of levels: All

Obs	Path	Type
1	\Genmod#1	Dir
2	\Genmod#1\ModelInfo#1	Table
3	\Genmod#1\NObs#1	Table
4	\Genmod#1\ClassLevels#1	Table
5	\Genmod#1\ResponseProfile#1	Table
6	\Genmod#1\ParmInfo#1	Table
7	\Genmod#1\ModelFit#1	Table
8	\Genmod#1\ConvergenceStatus#1	Table
9	\Genmod#1\ParameterEstimates#1	Table
10	\Genmod#1\Type3#1	Table
11	\Genmod#1\LSMeans#1	Table
12	\Genmod#1\LSMeanPlots#1	Dir
13	\Genmod#1\LSMeanPlots#1\MeanPlot#1	Graph
14	\Genmod#1\Diffs#1	Table
15	\Genmod#1\DiffPlot#1	Graph
16	\Genmod#1\LSMeans#2	Table
17	\Genmod#1\LSMeanPlots#2	Dir
18	\Genmod#1\LSMeanPlots#2\MeanPlot#1	Graph
19	\Genmod#1\Diffs#2	Table
20	\Genmod#1\DiffPlot#2	Graph
21	\Genmod#1\LSMeans#3	Table
22	\Genmod#1\LSMeanPlots#3	Dir
23	\Genmod#1\LSMeanPlots#3\MeanPlot#1	Graph
24	\Genmod#1\Diffs#3	Table
25	\Genmod#1\DiffPlot#3	Graph
26	\Genmod#1\DiagnosticPlot#1	Dir
27	\Genmod#1\DiagnosticPlot#1\DiagnosticPlot#1	Graph
28	\Genmod#1\DiagnosticPlot#1\DiagnosticPlot#2	Graph
29	\Genmod#1\DFBetaPlot#1	Dir
30	\Genmod#1\DFBetaPlot#1\DiagnosticPlot#1	Graph

Listing of: \Work.Genmodstore2\		
Order by: Insertion		
Number of levels: All		
Obs	**Path**	**Type**
31	**\Genmod#1\StdDFBetaPlot#1**	Dir
32	**\Genmod#1\StdDFBetaPlot#1\DiagnosticPlot#1**	Graph
33	**\Genmod#1\ResidualXBetaPlot#1**	Dir
34	**\Genmod#1\ResidualXBetaPlot#1\DiagnosticPlot#1**	Graph

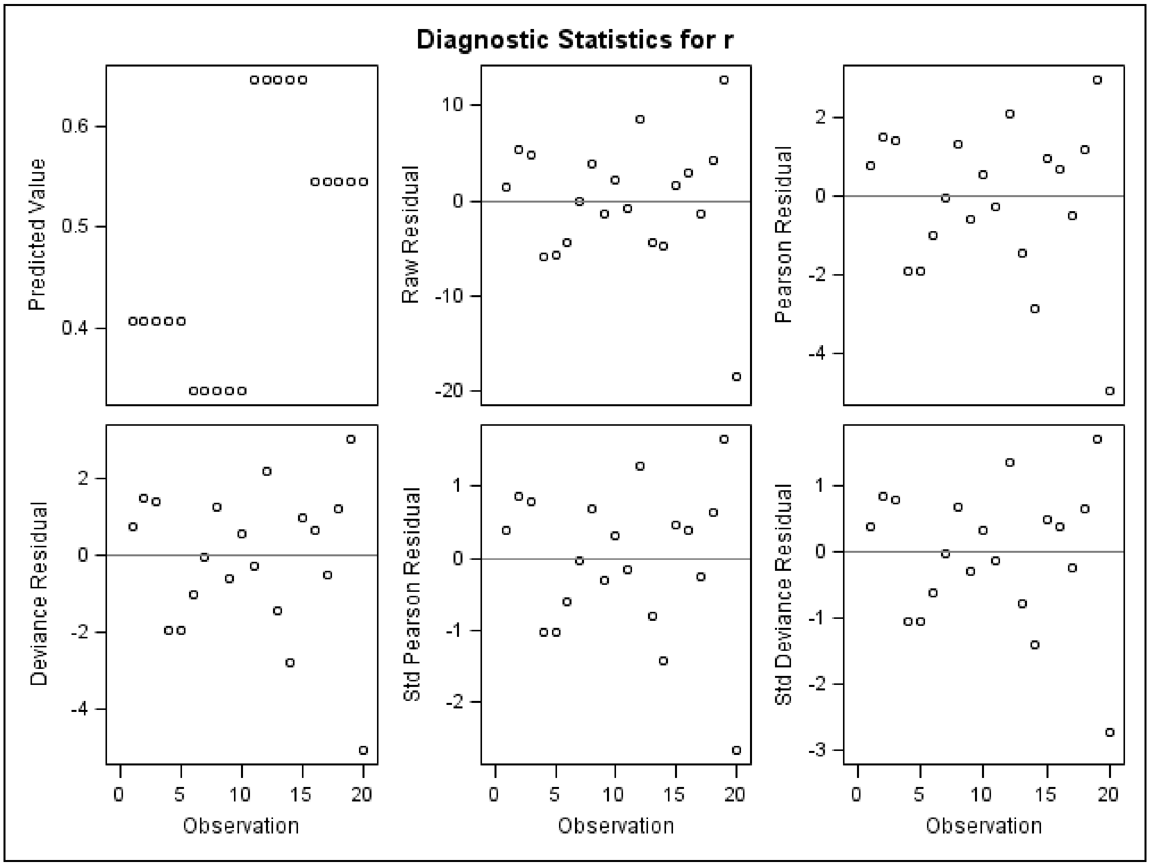
Looking though the element list, we see that a graph of interest is the *Genmod#1\DiagnosticPlot#1\DiagnosticPlot#1* graph. We use the replay feature of the document feature, to create a png file. Notice the use of the *listing* destination and how the file name is specified:

```
ods listing;
goptions device=png colors=(black) rotate=landscape;
ods graphics on / imagefmt=png imagename='output-to-LaTeX-graph01' reset=index;
proc document name=GenModStore2;
   replay \Genmod#1\DiagnosticPlot#1\DiagnosticPlot#1 / dest=listing;
run;
ods graphics off;
ods listing close;
run;
```

which gives a *png* file. We use

```
\includegraphics[width=\textwidth]{output-to-LaTeX-graph01}
```

to include the plot in the document:



4.3.6 Modifying the LaTeX tagset

The above tricks for creating LaTeX output work reasonably well. However, additional tweaks are sometimes necessary. For example consider extracting and generating the code for the population marginal means:

```
ods tagsets.latex file="output-to-LaTeX-fragment60.tex" (notop nobot) ;
proc print data=GenmodLSmeans noobs label split=' ' ;
run;
ods tagsets.latex  close;
```

This gives the following table:

Plot of sample proportion vs. number of seeds

Statement Number	Effect	cult	soil	Estimate	Standard Error	z Value	Pr > |z|
1	cult	0		0.1103	0.2199	0.50	0.6161
1	cult	1		−0.2473	0.1864	−1.33	0.1846
1	soil		0	−0.5266	0.2087	−2.52	0.0116
1	soil		1	0.3896	0.1989	1.96	0.0501
1	cult*soil	0	0	−0.3788	0.3077	−1.23	0.2183
1	cult*soil	0	1	0.5993	0.3143	1.91	0.0565
1	cult*soil	1	0	−0.6745	0.2821	−2.39	0.0168
1	cult*soil	1	1	0.1798	0.2437	0.74	0.4607

Notice that the column titles for *Statement Number* and *Standard Error* are very wide. If you look at the output in the pdf file generated earlier, you will see that these column headers are stacked vertically and if you look at the actual latex code generated you will see that the tagset tried to stack the column headers:

```
\multicolumn{1}{|S{header}{r}|}{Standard {\newline} Error} &
```

through the use of the \newline command. However, this is in a table where such commands do not work. Refer to a discussion at: `http://tex.stackexchange.com/questions/2441/how-to-add-a-forced-line-brea` about this issue.

The solution is to put the column headers where you want multiple lines into a \pbox environment.

To do this, we need to modify the *tagset.latex* code using *Proc Template*. Even before that we need to find the tagset and see what type of code generates the actual latex. Templates are a *SAS* feature that allows you to customize your output at a very (!) fine-grained level. The LaTeX templates are automatically installed in your *SAS* installation, but the source code can be extracted by clicking on the link at the top of the page at: `http://support.sas.com/rnd/base/ods/odsmarkup/latex.html`. This link downloads the templates from `http://support.sas.com/rnd/base/ods/odsmarkup/textags.tpl`

These templates are simple text files and can be opened with any text editor. As part of the template definition, you will see the following header in the tagsets.latex definition

```
    define tagset tagsets.latex;
        notes 'This is the LaTeX definition';
        stacked_columns = no;
        output_type = 'latex';
        map = '%$&{}~^#_\';
        mapsub= '/\%/\$/\&/\{/\}/\~/\^/\#/{\textunderscore}/{\textbackslash}/'
        nobreakspace = '~';
        split=' {\newline} ';
        embedded_stylesheet=yes;
        image_formats = ’ps,epsi,png,jpeg,gif’;
        /*ext_graph_instance = yes;*/
        ...
            lines omitted
        ...
        define event data;
            start:
                put VALUE /if cmp($sascaption, ’true’);
                break /if cmp($sascaption, ’true’);
                put ’ & ’ CR / if !cmp(COLSTART, ’1’) ;
                /* Print cell formatting including class name and alignment. */
                put ’   ’;
                put ’\multicolumn{’;
                put colspan;
                put ’1’ / if !exists(colspan);
                put ’}’;
                put ’{|S{’;
                put HTMLCLASS;
                put ’}{’;
                put just;
                put ’}|}’;
                put ’{’;
                put tranwrd(VALUE, ’-’, ’$-$’) / if contains(HTMLCLASS, ’Data’);
                put VALUE / if ! contains(HTMLCLASS, ’Data’);

            finish:
                break /if cmp($sascaption, ’true’);
                put ’}’;
        end;
```

The \newline in the definition of the *split* variable in the preamble and the *data event* is what actually creates the LaTeX code.

The \pbox environment allows you put multiple lines within a LaTeX cell in a tabular environment using code of the form:

```
\pbox{Line 1 \\ Line 2 \\ Line3 }
```

So we need to modify the latex.tagset to output this type of code.

Proc Template allows you create new templates by modifying existing templates. Here is my attempt. Notice I changed the definiton of \newline in the header part of the template and inserted the \pbox command where the data is output.

```
proc template;

/* This tagset changes the existing latex tagset to allow multiple
   rows within a table cell, e.g. in table headers created
   using the split=’ ’ argument in the Proc Print command */

   define tagset tagsets.mylatex; parent=tagsets.latex;

       notes ’This is the My new LaTeX definition’;
       split=’ \\ ’;  /***** change *****/

       define event data;  /* override the existing definition to put all cell entries
           start:
               put VALUE /if cmp($sascaption, ’true’);
               break /if cmp($sascaption, ’true’);
               put ’ & ’ CR / if !cmp(COLSTART, ’1’) ;

               /* Print cell formatting including class name and alignment. */
               put ’   ’;
               put ’\multicolumn{’;
               put colspan;
               put ’1’ / if !exists(colspan);
               put ’}’;
               put ’{|S{’;
               put HTMLCLASS;
               put ’}{’;
               put just;
               put ’}|}’;
               put ’{\pbox[b]{\textwidth}{’;    /******* Change here *******/
               put tranwrd(VALUE, ’-’, ’$-$’) / if contains(HTMLCLASS, ’Data’);
               put VALUE / if ! contains(HTMLCLASS, ’Data’);

           finish:
               break /if cmp($sascaption, ’true’);
               put ’}}’;    /******* change here for closing } ******/
       end;
   end;
run;
```

To use this new tagset, include it in your *SAS* program using the %INCLUDE command and use the new latex tagset name as before:

```
%include ’NewLatexTagset.sas’;
ods tagsets.mylatex file="output-to-LaTeX-fragment61.tex" (notop nobot) ;
proc print data=GenmodLSmeans noobs label split=’ ’ ;
```

```
title ;
run;
ods tagsets.mylatex  close;
```

Then in your LaTeX document, also include the

```
\usepackage{\pbox}
```

This gives the following table:

Statement Number	Effect	cult	soil	Estimate	Standard Error	z Value	Pr > |z|
1	cult	0		0.1103	0.2199	0.50	0.6161
1	cult	1		-0.2473	0.1864	-1.33	0.1846
1	soil		0	-0.5266	0.2087	-2.52	0.0116
1	soil		1	0.3896	0.1989	1.96	0.0501
1	cult*soil	0	0	-0.3788	0.3077	-1.23	0.2183
1	cult*soil	0	1	0.5993	0.3143	1.91	0.0565
1	cult*soil	1	0	-0.6745	0.2821	-2.39	0.0168
1	cult*soil	1	1	0.1798	0.2437	0.74	0.4607

Still not perfect as the column headers for *Statement number* should be right justified, but the table is much improved.

Chapter 5

Controlling the amount of output

SAS allows you control the amount of output generated from the various procedures. The most common requirement is to turn OFF all of the procedure output from a simulation study – you don't need to see the output from 1000 replicates of an analysis!

Turning off all output from a proceudre is done using the *ODS SELECT* command as follows:

```
ODS SELECT NONE;
... \SAS\ procedures
RUN;
ODS SELECT ALL;
```

Don't forget the *Run;* statement after the last procedure whose output is to be hidden.

Chapter 6

Should I use Proc GPLOT or Proc SGPLOT

Verions of *SAS* prior to 9.2, used procedures for graphing data which were often cumbersome to use and required many tricks to get interesting plots. Starting in version 9.2, *SAS* introduced a new series of graphical procedures such as SGPLOT.

A nice comparison of the two illustrating how to transfer plots coded in GPLOT to SGPLOT is available at `http://robslink.com/SAS/papers/sg_procs/sgplot.html`.