Some SAS macros for BUGS data

by Rodney A. Sparapani, MS
mailto:rsparapa@mcw.edu

July 29, 2008

BUGS/WinBUGS/OpenBUGS are great tools for performing Bayesian analysis. However, their data manipulation capabilities are limited. Along with Matthew Hayat, I have developed some SAS macros that will allow you to create BUGS input data files from SAS datasets and to create SAS datasets from CODA output files. They are available on the web with a brief discussion, including some notes and documentation, at `http://www.amstat.org/chapters/milwaukee/sasmacro/sas4bugs.html`. These SAS macros are a subset of a larger project called RASmacro that creates a user-friendly, free software, open source, SAS macro programming environment and I'm encouraging users to get the superset at `http://www.amstat.org/chapters/milwaukee/sasmacro/rasmacro.zip` However, this documentation, is focused on the SAS macros for BUGS only. It is important to note that new features are continually being added and bugs, lowercase, are constantly being fixed. I encourage you to look at the latest release before requesting a new feature or reporting a bug.

I have been working with SAS for almost 20 years and with SAS macros, in particular, for over a decade. SAS macros are very powerful, but often suffer from poor documentation. My own SAS macros are no exception. However, this is an opportunity to document my work and share it with others. I originally wrote this article for the ISBA Bulletin, but the macros have evolved over time and I keep the documentation as up-to-date as possible here. I'm going to describe a set of SAS macros that were written to allow a seemless movement of data between SAS datasets to BUGS input files and CODA output files back to SAS datasets.

There are two SAS macros to make BUGS input files. `_LEXPORT` takes a list of SAS dataset variables and creates an input data file of scalars referred to as a "list" data file. `_SEXPORT` takes a list of SAS dataset variables and creates an input data file of vectors referred to as a "structure" data file. Here's a code snippet:

```
%_lexport(data=mta, file=mta.txt, var=med beh com, close=0);
%_sexport(data=mta, append=mta.txt, var=pscale0-pscale3 tscale0-tscale3);
```

This creates one file, `mta.txt`, with "list" input for scalars (`med`, `beh` and `com`) and also with "structure" input for vectors (`pscale` and `tscale`). Each of these SAS macros requires only two parameters: either `FILE=` or `APPEND=`, and `VAR=`. `DATA=` is optional and it defaults to the last SAS dataset created. Other optional settings are `LS=` for line length and `FORMAT=` for variable length which you can change to make your input files more readable (e.g. for dummy variables,

LS=78, FORMAT=1.). Often, you will want to center a continuous variable and/or generate summary statistics for discrete variables with PROB= or continuous variables with MEAN= and PREC= to include in the data file, by default, or the init file, if the INIT= option is specified. Here's a code snippet:

```
%_lexport(data=mta, file=mta.txt, init=mta.in, var=med beh com age,
    close=0, initclose=0, center=age, prob=med beh com);
%_sexport(data=mta, append=mta.txt, initappend=mta.in,
    var=pscale0-pscale3 tscale0-tscale3, prob=pscale0-pscale3 tscale0-tscale3);
```

After generating posterior samples with BUGS, we want to create SAS datasets from our CODA files. Here's a code snippet:

```
*select device and graphics file name;
goptions device=psl gaccess=gsasfile;
filename gsasfile 'chains.ps';

*import CODA files and generate univariate statistics;
*NOT recommended: for comparison and completeness only;

*one chain at a time: chain 1;
%coda2sas(out=post1, infile=mta.ind, chain=mta1.out, stats=1);
*one chain at a time: chain 2;
%coda2sas(out=post2, infile=mta.ind, chain=mta2.out, stats=1, gsfmode=append);

data post;
    set post1 post2;
run;
```

The CODA2SAS macro was originally written with BUGS in mind. It can process CODA files, but doesn't handle multiple chains automatically. You have to import each of the chains manually. Set INFILE= to the name of your index file and CHAIN= to the name of your chain file. So, I imported the first chain into the SAS dataset post1. This SAS dataset contains three variables c_1-c_3 which correspond to the monitored array c[] where c[1] was translated as c_1, etc. But first, I set my graphics output file with a FILENAME GSASFILE statement and my graphics device with a GOPTIONS DEVICE= statement. This is necessary with a summary request (STATS=1); statistics and kernel density plots with histograms are generated. Without FILENAME/GOPTIONS statements, the plots will be displayed on your graphics display device. However, if you choose a device that is a graphics file type supporting multiple images (like PostScript or PDF), then a graphics file is generated. If you want more graphics output appended to the same graphics file in a subsequent request, specify GSFMODE=APPEND. If you choose a device that is a graphics file type that supports only single images (like Encapsulated PS or JPEG), then you need to specify TYPE=. For example, if you selected the JPEG device type and specified TYPE=jpg, then the following graphics files are generated: c_1.jpg, c_2.jpg and c_3.jpg.

```
*select device and graphics file name;
goptions device=psl gaccess=gsasfile;
filename gsasfile 'chains.ps';

*import CODA files and generate univariate statistics;
*this is recommended the way;

*all chains at once;
%_decoda(out=post3, infile=mta.ind, chains=2, var=c, mu0=1);
```

Next, I read in both chains with the `_DECODA` macro. Name your index file either `NAME.ind`, `NAMEIndex.txt` or `NAME.ind.txt` and each of your chains `NAME#.out`, `NAME#.txt` or `NAME#.out.txt` respectively, where `#` is the number of your chains (1-2 in this example). If you follow this naming convention, then an unlimited number of chains are supported by specifying only `INFILE=` for the index file and `CHAINS=` for the number of chains. If the chain files follow some other naming convention, then you can specify each of them manually as `CHAIN1=` up to `CHAIN10=`. The `INFILE=` and `OUT=` parameters are required as well as either `CHAINS=` or `CHAIN1=`, etc. The posterior samples from both chains are contained in the SAS dataset `post3`.

The same `CODA2SAS` comments apply to `_DECODA` with respect to `GFSMODE=` and `TYPE=` (although the syntax is the same for `TYPE=`, if the variable of interest is a monitored array, stick with `CODA2SAS`). Although `STATS=` is still accepted, `VAR=` is the new recommended name of the option which is more SAS-ish. If one or more SAS variable names contained in the SAS dataset created is/are provided as arguments via `VAR=`, then only those SAS variables are summarized. If you want all SAS variables summarized, then you specify that the SAS way as `VAR=_all_` instead of `STATS=1`. And, note that the syntax has changed. Instead of `VAR=c_1-c_3`, you specify `VAR=c`. This is more intuitive since that was the name of the monitored array, `c[]`. But, this requires the introduction of the SAS variable `OBS` into the `OUT=` SAS dataset that represents the element of the array, i.e. `1-3`. If the monitored variable is not an array, then `OBS=0`. In addition, more summaries are available. Tests and tables for location are performed and the default location of 0 can be changed with `MU0=`. If you specify `AUTOCORR=1` or set `NLAG=` to something other than 25, auto-correlations are generated (SAS/ETS is required for this option). By default, trace plots are generated. Other useful options, not shown, are `EXP=` for exponentiating parameters base e (useful for odds/risk ratios), `PCTLPTS=` for specifying percentiles other than 2.5% and 97.5% to summarize and `THIN=` for thinning the posterior (useful when substantial auto-correlation remains for many lags); see the documentation in the `_DECODA` macro for more details.

```
*transform data so that each record contains all 3 variables;
proc transpose data=post3 out=post prefix=c;
    where 1<=obs<=3;
    by chain iter;
    id obs;
    var c;
run;

*produce simultaneous confidence intervals;
%bayesintervals(data=post, vars=c1-c3, tail=U);
```

Lastly, I used the SAS macro BAYESINTERVALS by RD Wolfinger which is available at http://ftp.sas.com/samples/A56648. It constructs simultaneous intervals of the posterior for c1-c3. Also available at the same URL is BAYESTESTS by PH Westfall which is for Bayesian multiple hypothesis testing (BAYESTESTS requires an ESTIMATE macro which you can construct by hand or which can be created by MAKEGLMSTATS by RD Tobias which is also provided).

There are four other SAS macros that you might find useful: _DEBUGS, SAS2CODA, _LIMPORT and _CEXPORT. _DEBUGS is similar to _DECODA except that it only provides the summaries, it does not read CODA files so you have to specify DATA=. Also, _DEBUGS allows you to create a subset of the data with OUT= and THIN= and/or WHERE=. SAS2CODA reads a SAS dataset and creates CODA files. _LIMPORT creates a SAS dataset from a "list" file with two required options, OUT= for the new SAS dataset and INFILE= for the "list" file. However, importing can be tricky so a temporary SAS/IML program (which you can name with FILE=) is created and run automatically. If the importing fails, then you should be able to make corrections to the program and run it manually. _CEXPORT (with similar syntax to _LEXPORT) reads a SAS dataset and creates a Comma Separated Values (CSV) file that can be used with spreadsheet applications and can also be imported into Stata.

CODA2SAS, SAS2CODA, BAYESINTERVALS, BAYESTESTS and MAKEGLMSTATS are self-contained and do not require any other SAS macros. However, _DECODA, _DEBUGS, _LEXPORT, _SEXPORT, _LIMPORT and _CEXPORT require other SAS macros which are also included in decoda.zip and rasmacro.zip. Here are the instructions to install all SAS macros contained in decoda.zip or rasmacro.zip (similar instructions would apply to other SAS macros):

1. unzip decoda.zip or unzip rasmacro.zip in a directory I'll generically call SASMACRO (you can call it anything you want)

2. edit your SAS configuration file (sasv8.cfg, sasv9.cfg or sasv9_local.cfg)

2a. you probably have a line something like the following:

```
-sasautos '!SASROOT/sasautos'      /* for Unix/Linux */
-sasautos '!SASROOT\core\sasmacro' /* for Windows    */
```

2b. edit or add the corresponding line as follows:

```
-sasautos ( '!SASROOT/sasautos' 'SASMACRO' )      /* Unix/Linux */
-sasautos ( '!SASROOT\core\sasmacro' 'SASMACRO' ) /* Windows */
```