Jeff Y.Yuan
May 1998

Introduction to SAS

Table of Contents

1. Using SAS on a UNIX System

1.1 The UNIX Operation System

Our SAS runs under the UNIX OS operation system. So - even if you intend to use SAS only for data analysis - you need to know some fundamental UNIX commands. You will always be faced with such standard housekeeping duties as the printing, renaming, or deletion of files, either ones you've created directly or ones that SAS has produced for you.

An SAS program is a file that store all your commends. Therefore, you must be familiar with the use of files on your computer system. Basically, a file is like a piece of paper in your notebook or a chapter in a book; a directory is like the notebook or book itself. Here are a few examples of program calls on UNIX (the prompt % is typed by the computer):

% *cp* file1 file2	make a copy of file "file1" named "file2"
% *mv* file1 file2	Rename "file1 to "file2"
% *more* file	look at "file" a page at a time (use long space bar to page forward)
% *ls*	list names of your files in your directory
% *ls -l*	long version of "ls" (has dates of last change)
% *textedit* file	edit "file" using text editor "textedit" (you might set DISPLAY by *setenv* first)
% *vi* file	edit "file" using full screen editor "vi"
% *rm* file	remove "file" from your directory
% *enscript* file	print a text "file" to a postscript printer (you may check what is your printer by % *echo $PRINTER*)

There are on-line manual pages for UNIX commands. For example, type *man rm* to find out more about the *rm* command.

1.2 SAS Windows

When you run an X application/client, including the SAS systems, the client attempts to connect to the display specified by the environment variable. You might **first** create an environment variable called DISPLAY by entering this commend in the C Shell if your computer has IP address 123.4.567.89:

% setenv DISPLAY 123.4.567.89:0.0

Then you can enter the SAS command at the UNIX prompt to begin your "interactive mode" SAS session:

% sas &

You will get three windows. PROGRAM EDITOR window is where you enter your SAS statements. The LOG window contains initialization information about SAS. When you submit the SAS statements, the statements and related messages appear in the LOG window. The OUTPUT window contains the results from submitting SAS statements.

The following program is used to illustrate the use of SAS windows:

```
options;
data process;
    input wafer $ yield date $;
    cards;
1 0.9 12/4/97
2 0.95 12/4/97
3 0.82 12/4/97
;
proc print data = process;
run;
```

After typing above SAS codes into the PROGRAM EDITOR window, you can submit (execute) this program by clicking on the *Locals* menu and then on the *submit*. You will get messages in both OUTPUT window and LOG window.

You can save your SAS codes into a file by clicking on *File* and then choosing *Save* or *Save as* option. It is a good idea to get in the habit of using the .sas suffix to identify a file of SAS commands. Other standard suffix conventions include:

myfile.sas	SAS command file (may have data also)
myfile.dat	data file (if separate from "myfile.sas")
myfile.log	log of what SAS did with your commands
myfile.lst	output listing of your successful commands

The myfile name is up to you; use 8 or more characters in the name, starting with a letter (a-z). Numbers (0-9) or underscore (_) can be used, but avoid other symbols. This naming convention applies to SAS variables and datasets, except that only 8 or fewer characters are allowed.

You can invoke the SAS system in noninteractive mode. All you have to do is to type the following in UNIX command line:

% sas myfile.sas

where *myfile.sas* is the file containing the SAS statements. (You can edit it by *vi* or *textedit*.) The SAS program automatically produces the *myfile.log* and *myfile.lst* for you and saves them in your home directory. Usually it takes a little while to run.

It's good to always check the log file before looking at the output listing. It is usually short. Lines with *NOTE:* are fine and can be helpful in tracing transformations and creation of new variables. Lines with *ERROR* or *WARNING* should be heeded: look for the problem somewhere ABOVE the identified line.

2. SAS Language

2.1 Options

An SAS program usually contains three kinds of SAS "paragraphs" or "steps", namely:

options	printer options
data	data input, transformation and manipulation
proc	procedures for plotting, regression, etc.

The options, data steps and some procs (numerical and graphical summaries, plots) are described in the SAS/BASICS book. The other procs (regression, analysis of variance and other fancy stuff) are found in SAS/STAT. SAS reads your file in "free" format -- use as many spaces or tabs as you like -- but requires a semicolon (;) at the end of each "phrase" or "sentence". It is good practice to indent phrases in data or proc "paragraphs" for ease of reading.

Comments can appear anywhere in your program EXCEPT in the middle of data. Comment lines can begin with an "asterisk (*). Alternatively, a comment paragraph can be surrounded by "slash asterisk" (/*) and "asterisk slash" (*/).

```
 * this is a comment line
 /* this is also a comment line */
```

OPTIONS can **only appear as the first line** of myfile.sas. Here is a setup for looking on the screen:

```
options nocenter linesize=80 pagesize=24;
```

Nice size printer plots do better with the option pagesize=50. Common options include:

nocenter	do not center output (flush right instead)
linesize=80	set width of page to 80
ls=80	same as linesize
pagesize=50	set length of page to 50
ps=50	same as pagesize

2.2 Data Input

Input may be done **directly** in your myfile.sas or may come from another file. For direct input, here is an example:

```
data firstset;
  input x y;
  cards;
1      17.5
3      20.5
;
```

Here is an example of data input from **external file**:

```
data second;
  infile '~/datadir/mapping.dat' missover;
  input x y;
```

The names of directory *datadir* and data file *mapping.dat* are arbitrary, but can be used later in your SAS program to identify this particular data set. Details of input phrases (use either infile or cards, but not both):

```
data a;                          create new data set named "a".
  input x y z;                   input 3 numbers at a time as variables x,y,z.
  input trt $ x y;               input treatment "trt" as a character string and
                                  x,y as numerical variables. Note the dollar sign ($).
  infile 'trend.dat' missover;   use file "trend.dat" for the input data,
                                  "missover" skips over missing data
                                  rather than going to a new line.
                                  (must appear BEFORE the input phrase)
  infile 'trend.dat' firstobs=2;  skip first observation (first line),
                                    handy way to document column names.
  infile 'trend.dat' lrecl=2000;  allow for really wide data table or long records.
  cards;                         read data from following lines
                                 (must appear AFTER the input phrase).
  ;                              end of data entry for "cards" phrase.
```

Data values must have spaces between them (tabs can cause problems on some systems). All values must be on the same line if using the *missover* option. Missing data is represented by a period (.) as place holder. This can also be useful for estimation and prediction at new values using *proc* reg.

If you have a **SAS dataset** called stg12345.ssd001 in your home directory on SUN, then you can use following SAS codes to read the data:

```
Libname mylib '~/';                /* point to the right location */
Proc print data = mylib.stg12345;
Run;
```

2.3 Data Transfer from PC and SAS SQL with Database

If your data set is stored in some software on PC, You need to use FTP to transfer the data from your PC to the UNIX system on which SAS is running. The steps for the transfer will be (assume your data file is in Lotus 1-2-3):

a. In Lotus 1-2-3, click on File -> Save as -> file type: Text -> ok. You will get a new text format file.
b. In the Window bar, click on Start -> Programs -> Reflection -> FTP Client. Then the FTP window will pull up. You should use ASCII mode to transfer the data file.

All data from processing, parametric test and probe test are in IDEA databases. You can use SAS to access these databases automatically and use SAS/SQL to get the right data. To do this, you need some knowledge about the structure of our databases. The database schema and design of database tables published in our Web page will help you in writing SAS/SQL codes.

2.4 Data Transformations

There is no need to transform your raw data outside of SAS. In fact, it is good practice to leave your data file alone once it is debugged. Transforms are usually done in a separate data paragraph after data input. For example,

```
data newset;
  set firstset;
  logy = log(y);
```

This creates a new data set *newset* from the set *firstset* from data input above. The variable *logy* is created as the natural log of the variable *y*. Here are details of the first line and some transformations:

```
data a; set b;          create data set "a" using existing set "b"
  z = log(y);           create variable z as natural log of variable y
  z = log10(y);         log base 10
  z = sqrt(y);          square root
  z = x*y;              multiplication
                        (+ addition) (- subtraction) (/ division)
  z = y**2;             exponent: "y squared" or "y to the 2nd power"
  z = y**0.5;           "y to the 1/2 power" (same as sqrt(y))
  z = x**-2;            negative exponent: "1 over (x squared)"
  z = sin(x);           trigonometric sine function of x
                        (also cos(x), tan(x), ...)
```

2.5 Data Manipulations

You can add or drop variables/columns and observations/rows from a dataset. For instance, if you only wanted to consider the data with x greater than 10, you could have:

```
data other; set big;          /* create other from big */
  if x > 10;                  /* only use these cases  */
```

Suppose you had data set field with 3 wafers called 3, 4, 5 and you wanted to delete all data for wafer 3 for some procedures, the following will do it:

```
data subwafer; set wafer;     /* create subwafer from wafer */
  if wafer = 3 then delete;   /* delete data for wafer 3    */
```

Here is some more detail on the *if* phrase:

```
  g = 0;                          /* g=0 for large x */
  if x < 10 then g = 1;           /* g=1 for small x */

  if y = 99 then y = .;           /* recode 99 as missing data */
  if y = . then y = 0;            /* recode missing data as 0 */

  if z < 10 or y > 10 then x = 5;       /* examples of union (or) */
  if z < 10 and y > 10 then x = 6;      /* and intersection (and) */

  if x <= 10;                     /* keep only x at most 10 */
  if x >= 10;                     /* keep only x at least 10 */
  if not (x = 10);                /* keep only if x is not 10 */
```

You already saw how to **add** variables in transformations above. You can **drop** variables:

```
data a; set b;
  z = log(y);                 /* create new variable z */
  drop y;                     /* drop old variable y   */
```

Usually dropping is not necessary because the cost of carrying the unused variables is very small (unless you have a lot of data!). However, this is sometimes useful if the data need to be presented in a different way. For instance,

```
data abc;
   input p0 p1 p2 p3 p4 p5;
   cards;
1.4    1.5    1.2    2.1    2.1    2.8
1.7    1.4    1.0    1.4    1.7    2.1
1.1    1.9    2.5    2.6    2.1    2.2
1.7    1.3    1.1    1.0    2.0    1.8
1.0    1.8    1.5    1.4    2.2    2.3
;
data respons; set abc;
   resp = p0; site = 1; output;
   resp = p1; site = 2; output;
   resp = p2; site = 3; output;
   resp = p3; site = 4; output;
   resp = p4; site = 5; output;
   resp = p5; site = 6; output;
   drop p0 - p5;
proc print data=resps; run;
```

Basically, the output phrase produces a new observation after we create the variables *resp* and *site*.

After you submit the above SAS codes, you will get a new data set called *respons*. Part of it will be:

obs	resp	site
1	1.4	1
2	1.5	2
3	1.2	3
4	2.1	4
5	2.1	5
6	2.8	6
7	1.7	1
8	1.4	2
9	1.0	3
10	1.4	4
.	.	.
.	.	.
.	.	.

2.6 Random Numbers

Random numbers are available for a wide variety of distributions. These can also be used to generate experimental designs. It is best to use the functions with names beginning with *ran* - the uniform function *ranuni* appears to be better behaved than the function uniform using standard tests. But remember, computer generated random numbers are never truly random - caution and some checking on your own are always a good idea. Random numbers can be generated in a data paragraph:

```
data rn;
  do i=1 to 10;
    uni=ranuni(0);          /* an argument of 0 uses the clock as a seed */
                            /* otherwise, use a 5 to 7 digit odd number */
    output;
  end;
```

Note the use of a *do* loop, which is ended by an *end;* phrase. The output forces creation of a new case for each uniform number. Each case in set *rn* will have the variables *uni* and *i*. Here are the random number generators:

```
  x = ranuni(seed)              /* uniform between 0 & 1 */
  x = a+(b-a)*ranuni(seed);     /* uniform between a & b */
  x = ranbin(seed,n,p);         /* binomial size n prob p */
  x = ranexp(seed);             /* exponential with scale 1 */
  x = ranexp(seed) / a;         /* exponential with scale a */
  x = a-b*log(ranexp(seed));    /* extreme value loc a & scale b */
  x = rangam(seed,a);           /* gamma with shape a */
  x = b*rangam(seed,a);         /* gamma with shape a & scale b */
  x = 2*rangam(seed,a);         /* chi-square with d.f. = 2*a */
  x = rannor(seed);             /* normal with mean 0 & SD 1 */
  x = a+b*rannor(seed);         /* normal with mean a & SD b */
```

The seed above is either 0 (use clock to randomly start sequence); positive (used as initial seed - it should be odd and less than 2**31-1); or negative (use the clock to restart the sequence every time). The seed is only examined on the first encounter with a random number generator in your program, so you cannot change the process once you begin.

2.7 Experimental Design Using SAS

Experimental designs can be laid out using SAS. Here is an example of a design with 4 treatments/levels and 5 replicates per treatment. The following SAS program assigns trt the values 1,2,3,4, each with 5 replicates.

```
data uniform;
   do run = 1 to 20;
      x = ranui(0);
      output;
   end;
proc print data = uniform;

proc sort data = uniform; by x;
data c;
    set uniform;                  /* _N_ is the observation number      */
    trt = ceil (_N_ / 5);         /* ceil returns the next highest integer */
proc print data = c; var run trt;
run;
```

The SAS output looks like, but change by time from the below:

run	trt
1	1
17	1
13	1
4	1
16	1
2	2
7	2
15	2
6	2
14	2
12	3
18	3
11	3
8	3
10	3

20	4
5	4
3	4
9	4
19	4

Here is a randomized complete design, with 3 blocks (3 lots, for example) and 4 treatments per block. We assign the treatments 1,2,3,4 at random to the 4 sites within a block.

```
data aa;
  do block = 1 to 3;
     do site = 1 to 4;
       x = ranuni(0);
       output;
     end;
  end;
proc sort; by block x;
data cc; set aa;
    trt = 1 + mod(_N_ - 1, 4);       /* mod = remainder of _N_ /4 */
proc sort; by block site;
proc print data=cc;
  var block site trt;
run;
```

The output will be:

block	site	trt
1	1	4
1	2	1
1	3	2
1	4	3
2	1	2
2	2	4
2	3	1
2	4	3
3	1	3
3	2	1

3	3	4
3	4	2

which can be used as the plan for your experiment.

SAS has modules for DOE analysis. The details are included in SAS/QC and SAS/STAT manuals.

3. Introduction to Procedures

3.1 Procedures

Procedures come in many forms. They consist of the ***proc*** phrase followed by a set of sub-phrases particular to the procedure invoked. The ***proc*** phrase in its simplest form is simply

proc print;

This automatically uses the data set from the previous *proc* or data step. The form

proc print data=abc;

explicitly uses the data set *abc* rather than the previously or created one. Procedures usually do not create a new or add to existing data sets unless this is made explicit with an output phrase. For instance,

```
proc means;
  . . .
  output out = newname . . . ;
```

This explicitly creates the data set *newname*. Each *proc* has its own sub-phrases (the first ". . ." above) and their own set of variables that can be added to the new data set. The general form of the output phrase is:

```
  output out=d1 a=a1 b=b1 c=c1;
```

with *out*= being the keyword for the data set name d1 and a=, b=, and c= being any number of optional keywords for new variables. The names after the equals -

a1, b1 and c1, respectively - are up to you. They are the names of these variables that you can later use. The details can be found in the SAS/STAT book.

3.2. Sorting and Running a *proc* by Subgroups

Sometimes it is very helpful to run each of several subgroups through some summary or analysis procedure. This can be done with the sort procedure and use of the *by* phrase:

```
proc sort; by trt;
proc means; by trt;
```

will first sort the data by variable *trt* and then run the means procedure separately for each group. This is much cleaner than running SAS several times, each time retaining only a group under study. However, it does produce a lot more output! The *by* phrase is on all procedures. BUT you MUST sort before you use it. You can sort by several things at once:

```
proc sort; by site trt;
proc means; by site trt;
```

It is a good idea to always run *proc* sort before using *by* with other procedures, even if you think you did it earlier in your program.

3.3 Numerical Summaries

Below are samples that generate summary reports for your data sets:

```
proc means;                                  /* means, SDs, min, max for x and y*/
  var x y;                                   /* for variables x and y */
  output out=b mean=mx my std=sx sy;     /* output means and SD for x and y */

proc means noprint;                          /* Without printout */
  var x y;
  output out=b mean=mx my std=sx sy;         /* output means and SD for x,y */

proc univariate;                             /* detailed univariate summaries */
```

```
  var x y;                                  /* for variables x and y */
  output out=b mean=mx std=sx;     /* create set b with mean and SD for x only */
```

4. Graphical Reports

4.1 *proc* plot

The main character-based graphic routines are *proc* plot and *proc* univariate plot. There is a system of fancy graphics routines (beginning with letter g) in SAS/GRAPH. In addition, SAS has a module called INSIGHT which is very nice for graphics and general user interface.

Proc plot is very popular in SAS data analysis. It allows you to visualize the data in a quick and convenient way.

```
proc plot;                          /* scatter plot  */
  plot y*x;                         /* plot y vertical and x horizontal      */
  plot y*x='*';                     /* use "*" as plotting symbol           */
  plot y*z=trt;                     /* use value of trt as plotting symbol */
  plot y*x='*'  y*z / overlay;      /* overlay two plots on same page    */
```

Here is a way to construct Interaction Plots. It gives you a plot of the average values of y for each site and *trt*.

```
proc sort; by site trt;
proc means noprint; by site trt;
  var y;
  output out=means mean=my;
proc plot;
  plot my*site=trt;
```

The *noprint* option used in *proc* means is available for many procedures. Sometimes it can be very handy in shortening output. You can do plots by another variable.

Here is a fancier way to construct Interaction Plots and some Diagnostic Plots, which allows you to use the information from statistical modeling.

```
proc glm;
   class a b;
   model y = a | b;
   lsmeans a*b / out=lsm;
   output out=diag  p=py  r=ry;
proc plot data=lsm;                    /* Interaction Plot */
   plot lsmean*a=b;                    /* cell mean vs. a by b */
   plot lsmean*b=a;                    /* cell mean vs. b by a */
proc plot data=diag;                   /* Diagnostic Plots */
   plot y*py py*py='*' / overlay;  /* observed vs. predicted */
   plot ry*py;                         /* residual vs. predicted */
```

You can set several plot features:

```
   plot y*x / vaxis=10 to 100 by 5;    /* vertical axis ticks */
   plot y*x / haxis=10 to 20 by 2;     /* horizontal axis ticks */
   plot y*x / vzero hzero;             /* include origin on plot */
   plot y*x / href=0;                  /* horizontal reference line */
   plot y*x y1*x='*' / overlay;        /* overlay two plots */
```

4.2 A Sample of *proc* gplot

SAS/GRAPH is a very powerful tool for data presentation. If you copy the below SAS codes into your SAS editor window and submit it, you will get a very nice output.

```
/************************************************************/
/*  PART 1: GOPTIONS always in the beginning of SAS codes          */
/************************************************************/
goptions reset=all gunit=pct colors=(white) cback=blue
        ftext=swiss htext=2.5;
/************************************************************/
/*  PRAT 2: Data steps to define the input for the graph                */
/************************************************************/
DATA MAP;
length function color $8;
retain xsys ysys zsys '2' when 'b';
DO ANGLE=0 TO 2*3.1416 BY .05;
```

```
  if angle=0 then function = 'poly';
  else function='polycont';
  style='solid';
  color='yellow';
  RADIUS=20;
  X=RADIUS*COS(ANGLE);
  Y=RADIUS*SIN(ANGLE);
  ID='CIRCLE';
  z=0;
  OUTPUT;
END;
DO ANGLE=0 TO 2*3.1416 BY .05;
  if angle=0 then function = 'poly';
  else function='polycont';
  style='solid';
  color='green';
  RADIUS=20;
  X=RADIUS*COS(ANGLE);
  Y=RADIUS*SIN(ANGLE);
  ID='CIRCLE';
  z=50;
  OUTPUT;
END;
/*************************************************************/
/*  PART 3: SAS/GRAPH procedures for generating the graph    */
/*************************************************************/
PROC G3d data=map anno=map;
scatter y*x=z/shape='point' noneedle zmin=0 zmax=50;
run;
quit;
```