Adapted from P. Smith

9 Introduction to SAS

Assume that you have either done the telnetting dialogue to log onto the cluster machines, and then issued the command ”sas &” from the unix prompt on a cluster machine, *or* that you have started SAS on a PC. At this point SAS will open a variety of windows on your mathnet screen, of which we describe the most important.

9.1 The SAS environment

This section describes the three most important SAS windows as they appear on a mathnet machine. Users from other machine platforms will see essentially the same environment, with only minor differences.

The three key windows are titled `SAS: Program Editor`, `SAS Log`, and `SAS: Output`. Each window has a title bar and a pull-down menu bar, as indicated below.

1. The `SAS: Program Editor` window, which you use to write your SAS code.

```
                 SAS: Program Editor-Untitled
  File    Edit    View    Tools    Run    Solutions    Help
00001
00002
00003
00004
00005
```

2. The `LOG` window, in which SAS shows you how the program is running and also gives you error messages if there are any mistakes in your program.

```
                SAS: LOG-Untitled
  File    Edit    View    Tools    Solutions   Help
```

3. The OUTPUT window, in which the results of your SAS program are displayed.

```
                SAS: OUTPUT-Untitled
File     Edit    View     Tools     Solutions     Help
```

9.2 Pull-down menus in the Program Editor window

The entries in the File and Edit menus mostly work just like those on any other unix window. Note that Run:Exit... terminates the SAS run. The View menu lists SAS windows and enables you to focus on the window of your choice.

The Run menu is critical: this menu enables you to execute your program. Program statements are typed in the body of the window. Clicking Run:Submit causes your program statements to be compiled and executed by SAS. After clicking Run:Submit, your program statements vanish. Clicking Run:Recall Last Submit brings them back for further editing as desired. The menu item Run:Submit N Lines causes the first *N* lines of the Program Editor window to be submitted (and to vanish, as described above). If one highlights a block of commands and uses Edit:Copy to copy them onto the Clipboard, then Run:Submit Clipboard submits the selected commands for execution. In this case, the commands do not disappear from the Program Editor window.

9.3 Editing tips for the Program Editor window

- To delete a line, type d in the margin (number line) and hit Enter.
- To delete several lines (a block), you may either:

  1. type dd at the beginning line of the block and dd at the ending line of the block, and then type Enter;
  2. type d5 at the beginning line of the block if you wish to delete, say, the next five lines.

  Note that once you delete a line or block of lines, it cannot be recovered.

- To insert a blank line, type i in the margin, then hit Enter. To insert several lines (say 3), type i3.
- To copy a line:

1. Type `c` on the margin of the line you want to copy.
2. Go to the margin of the line where you want to insert the copy and type `a` if you want the copy inserted after this line, or `b` if you want the copy inserted before this line.

- To copy several lines, use `cc` ... `cc` or `c#` to mark the lines you want to copy, and then use `a` or `b` to indicate the location of the insertion.
- Moving one or several lines is done in the same way as copying lines except that one uses `m`, `mm` ... `mm`, or `m#` to mark the area to be moved.

9.4 Saving the Contents of a Window

The `Save As` option under the `File` menu enables you to save your program file on the disk. This causes a dialog box to open, enabling you to choose a file name and a directory to store the text. (It is helpful to save SAS programs with names like `myprogram.sas`.) At a future session you can use `File:Open` to recall and rerun your program.

Alternatively, use any other editing tool you are familiar with to write your SAS code, and save it as a plain text file. Then use the `Open` option under the `File` menu in the `Program Editor` window to retrieve it.

The contents of the `Output` or `Log` windows can also be saved or recalled as described above. Note that the contents of the `Output` window will be saved as a text file.

9.5 Running SAS programs in batch mode

1. Create the SAS code using any editing tool. Save it as a plain text file, with `.sas` as the extension filename: `filename.sas`. If necessary, transfer it to your WAM account.
2. Telnet to the WAM system. Log on to your account.
3. Use `sas filename` to execute your file. The SAS system will, after the run, create a `LOG` file with the name `filename.log`. If the program has no bugs and there is some output, the resulting file `filename.lst` is also created.

9.6 Transferring and Printing Files

From any mathnet machine, either use **ftp** or **stfp** to move files back and forth between mathnet and the cluster muchines, or copy from SAS windows or saved text-files in a cluster-machine window into text-editor windows on your mathnet machine.

9.7 Getting help

- The `Help:SAS System Help` menu selection, which is available on any SAS window, will open a Netscape window with help on all components of SAS. You will most likely want help with Base SAS Software or SAS/STAT Software at the beginning.

- The SAS manuals are available on line at

  `https://www.statlab.umd.edu/sasdoc/sashtml/onldoc.htm`

  You do not have to be running SAS to view the online documentation.

- If you have any questions regarding the WAM system, you can call the Consulting Lab, (301) 405-1500, Rm. CSS 1400, for help.

- If your question has more to do with statistical software than the cluster system, you can also call the Statistics Lab, (301) 405-6030, Rm. CSS 1425, for further assistance. However, the lab will not help students to solve homework assignments.

9.8 SAS: STATISTICAL ANALYSIS SYSTEM

SAS is:

- a collection of programmed statistical procedures,

- a data management / file manipulation package,

- a compiler for a high level programming language,

- a system for data description and report writing.

SAS programs consist of `DATA` steps and `PROC` steps (procedures). `DATA` steps: Create and manipulate data files, one observation at a time. `PROC` steps summarize or analyze SAS data sets, and can produce printed results or new data sets.

SAS statements:

- Must end in a semicolon (;)
- Generally are free format — spaces and position on a line do not matter. A statement can wrap onto several lines. This means that you can and should use spacing, indentation, and other elements of style to create a program that is visually easy to read.

Names in SAS are:

- Up to 8 characters long. First character must be a letter or underscore _ . Other characters may be letters, digits, or underscore. Special characters, like `$ @ #` etc. are **not** allowed.
- These rules apply to names of variables, data sets, procedures, arrays, etc.

9.9 The DATA Step

- The `DATA` step interprets your raw data and puts it into a standard form which SAS can deal with.
- SAS `PROC`'s can not work with raw data. SAS must first create its own version of your data, called a SAS dataset.

SAS datasets are organized by:

- *Observations:* the data values associated with a single entity. Typically each line of data corresponds to one observation.
- *Variables:* each observation may consist of several variables.

Schematic representation of a SAS data set:

```
                     Variables

Obs         DOSE       RESPONSE      GENDER

 1           125          3.1           M
 2           135          9.0           M
 3           173         15.4           F
 .            .            .            .
 .            .            .            .
 .            .            .            .
 12          183         20.7           M
```

Sample SAS code for a `DATA` Step:

```
Data   one ;
   input   dose  response  gender  $  ;
   lndose  =  log( dose ) ;
 datalines ;   /* indicates start of data lines */
    125        3.1          M
    135        9.0          M
    173       15.4          F
     .  .  .  .  .  .  .  .  .  .
    183       20.7          M
     ;            /* indicates end of data lines */
 run ;
```

The names, such as `one` for the dataset, or the variable names, such as `dose, response, gender, lndose` are user-defined.

The **input** statement tells SAS how to read the lines of datathat you are entering. The default (shown above) is to simply list the variable names. SAS assumes that at least one space will separate the data entries. Missing values should be written with a dot (.) .

The `$` above specifies that the variable `gender` is a character-valued variable.

Another form of the input statement specifies column positions:

```
  input   A   1-5   B  6-9   C  12   E  $ 15-20  ;
```

The input statement can be very powerful; many possible formats and positioning options are available. Using the pointer controls, it is possible to read very complicated raw data, for example, multiple records per observation.

A sample SAS program which reads data, transforms variables and produces simple summaries is given below.

```
data one ;
  input dose response gender $ ;
  lndose = log(dose) ;
datalines ;
125   3.1   M
135   9.0   M
173   15.4  F
154   3.2   F
136   2.0   M
153   .     F
165   20.2  M
164   17.1  M
144   14.9  F
183   20.7  M
155   13.9  F
130   8.9   M
;
run;

proc sort data=one ;
  by gender dose ;
proc print ;

proc means;
  var dose response;
proc means;
  by gender;
  var dose response;
proc univariate;
  var dose lndose response ;
run;
```

9.10 Permanent Files in SAS

It is important to distinguish between ASCII data sets and SAS data sets. An ASCII data set contains data coded in standard format: letters, digits and special characters. An ASCII data set can be viewed using an ordinary editor. On the other hand, a SAS data set contains coded information identifying the contents and variables to SAS; the data values are coded according to an internal SAS format which is not viewable by a computer editor. SAS can handle both types of data files in a `DATA` step, but with somewhat different commands.

Data input from an ASCII file. A `DATA` step which reads data from an existing ascii file is very similar to the previous sample `DATA` steps which read data from the input stream. An input file is identified to SAS by an `INFILE` statement. The following elementary example illustrates the syntax.

```
data fromfile ;
  infile ’/home/pjs/info/my.stuff’ ;
  input name $ rank $ ser_no ;
```

The file is specified by the `INFILE` statement. Note that the full Unix pathname must be given between the quotation marks. Alternatively, one can provide a `FILENAME` statement (which may appear outside the `DATA` step). This allows multiple `DATA` steps involving the same file and also allows one to specify information about the physical layout of the file (necessary for instance if the file exists on tape or CD-ROM). The example above could have been programmed as follows.

```
filename myinfo ’/home/pjs/info/my.stuff’ ;
data fromfile ;
  infile myinfo ;
  input name $ rank $ ser_no ;
```

In this program `myinfo` is what SAS calls a *fileref*: a label by which the SAS job references a certain file stored permanently on the computer system. Depending on the platform, filerefs may also be specified by platform-specific statements or through operating system commands.

Writing data to an ASCII file. To create an external file containing output data, a SAS programmer must use `PUT` statements in a `DATA` step. Note that by default, a `DATA` step creates an output line in a *SAS data set.* This default is overridden by the `PUT` statement. One must also identify the file to be created, either by using a FILE statement in the DATA step or by a FILENAME statement. A simple example using the FILE statement follows.

```
data makefile ;
  infile '/home/pjs/info/my.stuff' ;
  file '/home/pjs/info/month.pay' ;
  input name $ rank $ ssno annpay ;
  monpay = annpay/12 ;
  put name $ 1-10 @12 ss_no 9.0 ann_pay 6.0 mon_pay 8.2 ;
```

This program creates a new variable and writes a new ascii file. It is necessary to tell SAS how the output is to be arranged in the output records. In this example, NAME will be a 10 character numeric field. The numerical variables begin in the 12th character of the output line. The 9.0 indicates that SSNO is to be coded as a nine digit field with no digits to the right of the decimal point; on the other hand, MONPAY will be coded in an 8 character field with two characters to the right of the decimal point. These are examples of SAS FORMATs. There are corresponding INFORMATs for reading input data.

SAS libraries. To cause SAS data sets to be saved permanently, it is necessary to create a SAS Library. Essentially, this means that one must specify a directory to which a SAS data set is to be written. By default, all SAS data sets are recorded in the library WORK, but this library vanishes when the program is terminated. A LIBNAME statement is needed to cause SAS data sets to be recorded permanently. The library identification is then part of the data set name and must be used explicitly in later SAS steps. Consider the following example.

```
libname drugs '/home2/pjs/drugdata' ;
data drugs.rawdata ;
  input drug $ dose response gender $ weight ;
  datalines ;
ASPIRIN 100 1 M 183
BUFFERIN 150 0 F 107
 . . .
;
```

The LIBNAME statement identifies a directory on the system. Its *libref* is drugs. Any SAS data set with a name of the form DRUGS.*xxx* will be stored permanently in that directory as a SAS data set. Such a data set is available for analysis or updating in later SAS sessions.

Using SAS data sets. A SAS data set can be used as input to a later SAS DATA step very easily. Instead of an INPUT statement, one uses a SET statement. For instance the data set created in the previous example can be accessed by the following program.

```
data patients ;
  set drugs.rawdata ;
  . . .
```

Any of the variables in the data set specified by SET are available for analysis. Note also that the data set PATIENTS is temporary and will vanish at the end of the SAS job.

To find out about the variables in a SAS data set, use PROC CONTENTS. This will list all variables in the data set, whether they are numeric or character, information about how they are stored, the number of records, etc.