Introduction to SAS Programming

Jerry Alan Veeh

March 5, 1997

1 Introduction

Welcome!

- Short Tutorial
- Projects are listed here together with a brief description.
- Language Reference
- Procedures Reference
- Function Reference
- Configuration Reference

This is version 0.9, last updated on March 5, 1997.

SAS is a registered trademark of the SAS Institute. Microsoft Windows is a trademark of Microsoft Corporation.

2 Short Tutorial

2.1 Basic SAS Operating Environment

The SAS operating environment consists of three windows: the program editor window, the output window, and the log window.

The **program editor window** is where all SAS commmands are typed by the user. The commands are not processed by SAS until the LOCAL–SUBMIT option from the menu in the program editor window is selected.

The **output window** displays the results of the analysis performed by the procedures invoked in the program editor window.

The **log window** contains a transcript of all statements submitted for processing from the program editor window and also any error messages generated.

2.2 Basic Syntax

SAS has a syntax consisting of keywords which invoke procedures or select options in them, as well as variable names and operations.

The SAS language is NOT case sensitive.

The language is written in logical lines. Each logical line MUST be terminated by a semicolon. A logical line can span multiple lines on screen. Failure to terminate a logical line with a semicolon can lead to strange and difficult to trace errors. Here is an example of one logical line:

```
proc discrim
     data=jv.junk
     method=normal
     pool=yes
     posterr;
```

2.3 Comments in Programs

Program comments consist of text contained between /* and */. Comments can NOT be nested. Examples:

```
/*
This is a
multiple line comment.
*/

proc discrim      /* Use the discriminant analysis procedure */
     data=jv.junk
     method=normal /*Analysis assumes multivariate normal */
     pool=yes      /*Use pooled estimate of covariance */
     posterr;      /*Estimate classification errors */
```

Comments make the purpose of statements clear to any reader (even yourself one week later). Use plenty of comments in your programming.

2.4 Data Step Basics

ALL data MUST be put into a **SAS data set** before that data can be analyzed by SAS. The **data step** is used to construct a SAS data set which will be further analyzed using SAS procedures (procs).

Here is an example of a simple data step. These lines would be typed in the program editor window.

```
data first;     /* SAS data set is called "first" */
input name $ age; /* Data set has 2 variables "name" and "age" */
cards;     /* The actual data follows */
```

```
john 12 /* Here is the actual data.*/
jane 11 /* The data lines do NOT end with */
jack 13 /* semicolons!!! */
amy  10
run; /* Execute these statements now. */
```

Each line of data is referred to as a **data card** or a **data record**.
Here (conceptually) is the data set `first` as SAS sees it:

```
OBS NAME  AGE
1   john  12
2   jane  11
3   jack  13
4   amy   10
```

The SAS internal variable `OBS` is not user accessable.

2.5 Running a SAS Program

Type the following lines in the program editor window.

```
data first;
input name $ age;
cards;
john 12
jane 11
jack 13
amy  10
run;

proc print data=first;
run;
```

Lines typed in the program editor window are not processed by SAS until the program is submitted for processing by selecting the LOCAL–SUBMIT option from the menu in the program editor window. Select LOCAL–SUBMIT from the menu in the program editor window now.

The data should appear in the output window. If not, return to the program editor window and select LOCAL–RECALL from the menu. Check that the program was typed exactly as above. Make any necessary corrections and re-submit the program. You may also check the log window for error messages.

2.6 Data Step Details

Here are the hidden details of how SAS processed the data cards in the previous example. Each data step program contains two **implicit** pieces.

The SAS data step contains an implicit loop. The programming statements in a data step are executed once for each data card. This is how the successive data cards are read. The values of all variables defined in the data step are erased immediately before each pass through this implicit loop. (The retain statement can be used to circumvent this behavior.)

The SAS data step also contains an implicit output statement. After the final program line of the data step the current values of **all** variables are written to the SAS data set. This is how the SAS data set is created.

The data step program operates conceptually as though it were written like this.

```
data first;
do while there are data cards;
input name $ age;
output;
loop;
cards;
john 12
jane 11
jack 13
amy  10
run;
```

The output statement can be used by the programmer to control when variables are written to the data set. The implicit loop is always present. Keep these facts in mind as you write your data step programs.

2.7 SAS Variables

There are two types of variables in SAS: numeric variables and character variables.

Numeric variables have values which are numbers. This is the default data type in the input statement.

Character variables have values which are character strings. The names of character variables are **followed** by a $ in the input statement.

The names of all SAS variables must be 8 characters or less with no embedded blanks.

2.8 Defining New Variables

Raw data can be manipulated in the datastep. This is often done prior to the analysis of the data using SAS procedures. Here's an example.

In a math course, the course grade is based on the scores on 3 midterms and a final exam. The midterms are worth 20% each; the final worth 40%. On each test 100 points are possible. Here's an example of a datastep which inputs data and computes the course total.

```
data score;
```

```
input last $ first $ m1-m3 final;
total=0.2*m1+0.2*m2+0.2*m3+0.4*final;
cards;
brown susan 75 82 35 88
smith james 22 92 45 97
run;

proc print data=score;
run;
```

Copy this program into the program editor window, select LOCAL–SUBIMT from the menu there, and see how it runs.

The data set `score` will contain a variable `total` which is the weighted average of the midterm and final exam scores. Note the use of the subscripted variable `m` to stand for the 3 midterm scores.

The usual trigonometric, exponential, logarithmic, and arithmetic functions are available for use in the data step. The functions topic gives a partial listing of other specialized functions which are available.

This concludes the tutorial. You may wish to begin working on a project to get experience in data step programming and the use of SAS procedures in the analysis of data.

3 Projects

Here's a list and brief description of the available projects. Everyone should do the first 4 projects.

- Project 1 An introduction to the SAS operating environment.
- Project 2 The basic SAS data step with input of data directly through the cards statement; use of labels, the sort procedure and print procedure; the means procedure.
- Project 3 Reading data from ASCII files; computing new variables in the data step; the means procedure.
- Project 4 Modifying existing SAS data sets using set; using loops in the data step; the ttest procedure.
- Project 5 Column-wise input; analysis of categorical data using chi-square tests.
- Project 6 Updating existing SAS data sets with new data.
- Project 7 Basics of presentation quality graphics with proc gplot and proc g3d.
- Project 8 Basic one factor analysis of variance using proc GLM.
- Project 9 Advanced analysis of variance, custom hypothesis tests, and other features of proc GLM.

- Project 10 Multivariate analysis of variance using proc GLM.
- Project 11 Basic Box-Jenkins modeling of univariate time series analysis using proc arima (time domain).
- Project 12 Some aspects of frequency domain analysis of time series using proc spectra.
- Project 13 Discriminant analysis with proc discrim.
- Project 14 Reading data from dBase and DIF files; using dBase and DIF files instead of actual SAS datasets.
- Project 15 Using arrays, first and last, and processing dates. Repeated measures analysis.

3.1 Project 1

The objective of this project is to familiarize you with the SAS environment.

ASSIGNMENT:

1. Here is an example of a simple data step. These lines would be typed in the program editor window.

```
data first;
input name $ age;
cards;
john 12
jane 11
jack 13
amy  10
run;
```

Use the windows clipboard to place the above lines in the SAS program editor window. After the `run;` statement type the lines

```
proc print data=first;
run;
```

also in the program editor window. Then select LOCAL–SUBMIT from the menu in the program editor window. You should see the data appear in the output window.

If your program doesn't run properly, check the log window for error messages. Then recall the program into the program editor window by selecting LOCAL–RECALL from the menu in the program editor window. Make necessary corrections and then resubmit the program for processing by SAS.

2. Enhance your program by using the title statement to put a title line with your name, project number, and date as the first line on all printouts produced by SAS. (Use LOCAL–RECALL from the menu in the program editor window to recall a copy of your program, if necessary.) Don't mix the title statement with the data. It could be

the first line of your program (before `data first;`). DO THIS ON ALL FUTURE PROJECTS TOO.

3. After you get your program working, SAVE IT to disk. First, make sure your program is visible in the program editor window by using LOCAL–RECALL from the menu in the program editor window if necessary. Using the menu in the program editor window, select FILE–SAVE AS–WRITE TO FILE and then type in a name such as `a:proj1.sas`. The program can be retrieved by using FILE–OPEN–READ FILE from the menu in the program editor window.

4. Make the output window active by clicking in it. Then use EDIT–CLEAR TEXT from the menu in the output window to clear the output window. Switch back to the program editor window and run your program one final time. Switch back to the output window and select FILE–PRINT from the menu. A hard copy of output should be printed on the printer. Turn this in.

5. With your program visible in the program editor window, use the FILE–PRINT selection from the menu in the program editor window to print a hard copy of your program. Turn this in too.

3.2 Project 2

In this project a simple data set is analyzed.

This problem is taken from Finkelstein and Levin and the data is contained in the file sleep.dat.

The Able Company claimed in TV ads that taking its product Super-Drowsy reduced the time to fall asleep by 46% over the time necessary without pills. Able based this claim on a sleep study. Persons were asked to record how long they took to fall asleep ('sleep latency') in minutes, and their average for a week was calculated. In the next week these persons received Super-Drowsy and recorded their sleep latency. The following table gives the average sleep latency for each of the 73 persons first for the week without Super Drowsy and then for the week with Super Drowsy.

1	61.07	20.36	26	16.07	7.50	51	41.79	11.79
2	46.07	37.50	27	33.21	26.79	52	22.50	22.50
3	84.64	61.07	28	51.43	20.36	53	39.64	18.21
4	31.07	9.64	29	28.93	18.21	54	78.21	26.79
5	54.64	28.93	30	139.29	37.50	55	46.07	16.07
6	26.79	11.79	31	78.21	39.64	56	46.07	28.93
7	58.93	31.07	32	43.93	35.36	57	61.07	33.21
8	13.93	9.64	33	111.43	7.50	58	39.64	28.93
9	71.79	35.36	34	56.79	31.07	59	56.79	18.21
10	82.50	33.21	35	106.07	43.93	60	43.93	24.64
11	37.50	24.64	36	98.57	77.14	61	43.93	24.64
12	35.36	46.07	37	43.93	70.00	62	31.07	24.64
13	37.50	9.64	38	114.64	26.79	63	46.06	24.64
14	114.64	16.07	39	39.64	33.21	64	22.50	9.64
15	35.36	9.64	40	91.07	31.07	65	50.36	41.79
16	73.93	35.36	41	18.21	20.36	66	41.79	18.21

17	17.50	11.79	42	63.21	18.21	67	35.36	18.21
18	94.29	20.36	43	37.50	20.36	68	65.36	78.21
19	22.50	9.64	44	41.79	22.50	69	37.50	31.50
20	58.93	30.00	45	40.00	43.93	70	48.21	9.64
21	46.07	100.71	46	50.36	41.79	71	102.86	43.93
22	31.07	9.64	47	48.21	60.00	72	86.79	43.93
23	62.14	20.36	48	63.21	50.36	73	31.07	9.64
24	20.36	13.93	49	54.64	20.36			
25	56.79	32.14	50	73.93	13.93			

ASSIGNMENT:

1. Put the data above into a SAS data set containing 3 variables and use `Patient`, `Week 1` and `Week 2` as labels. Refer to Data Step Basics , SAS Variables, and Input Statement (List) for assistance. (Use the windows clipboard to transfer the data from the help file to the SAS program window following the cards statement.) Your input statement should end in @@ here.

2. Use proc sort to arrange the data in increasing order by patient number. Print this sorted data set using proc print.

3. Use proc means to compute the mean and standard deviation of the sleep latency times for each individual week. Also find a 90% confidence interval for the weekly mean sleep latency time. Make a nice printout of this information too.

4. Briefly describe a reasonable procedure that could be used to test statistically for the effectiveness of the drug. No more than a short paragraph here!

3.3 Project 3

In this project futher analysis is done with the data from Project 2.

Since two sleep latency measurements, given in the file sleep.dat, are taken on each patient, one correct method for testing for effectiveness of the drug is to do a paired comparison t-test. To do this test, the difference of the sleep latency times for each patient must be computed. As is described in Defining New Variables this can be done easily by defining a new variable in the data step.

ASSIGNMENT:

1. Use proc means and the modified data set to test for effectiveness of the drug. Turn in a neat printout of the RESULTS of using proc means. There's no need to print out the actual data. Turn in a copy of your program too. Give a SHORT answer to the question: is the drug effective? Don't forget to use the title statement to put your name and project number on each page of the printout.

2. The file iris.dat contains the famous 'Fisher Iris Data' (from Andrews and Herzberg) consisting of sepal length and width and petal length and width measurements for 3 types of Iris: iris setosa, iris versicolor, and iris virginica. There are 50 measurements for each type of iris. Each row in the file contains 12 numbers: the four sepal and petal measurements for one plant of each species in the order specified above. Use the infile statement (not the cards statement) to create a SAS data set which contains only the sepal length measurements for each species. Turn in a nice printout of this data set, with appropriate labels. Use the pagesize option to print about 55 lines

on a page. Consult infile statement for the method of reading files. The information in Defining New Variables may also prove useful, as might that regarding drop and keep.

3. Briefly describe how you would conduct a statistical test of whether the sepal lengths for the three species are the same or different.

3.4 Project 4

Here the Fisher Iris contained in iris.dat data is examined in more detail.

To compare the sepal length data for the 3 types of iris (data given in project 3), the most appropriate analysis would be a one factor analysis of variance. Using SAS to do this analysis will be discussed in Project 8. Here two sample t-tests are done to make pairwise comparisons of the 3 types.

Recall that the two sample t-test is used to test the equality of means when one has 2 independent samples from 2 normal populations. In the standard scenario, these normal populations are assumed to have equal, but unknown, variances. Approximations can be made to handle the case in which the variances are unknown but not assumed to be equal.

In order to use proc ttest, it is necesary to have a variable in the data set that can be used in the class statement.

ASSIGNMENT:

1. Use the techniques in Defining New Variables and Do Loops to create a SAS data set containing 5 variables: `species` and `m1-m4`. These variables contain an indicator of the species of each plant and the 4 corresponding measurements. This data set should contain 150 observations.

2. Use the set statement and the where statement to construct 3 SAS data sets from the single data set of problem 1. Each of these data sets should contain the sepal length measurements for 2 of the 3 species and a variable (taking only 2 values) which distinguishes the species from which the corresponding sepal length measurement came. This new variable will be used in the class statement in proc ttest. You don't need to print out a copy of the data sets but do turn in a copy of your program.

3. Run proc ttest and do pairwise comparisons of the sepal length data for the 3 types of iris. Print out a nice copy of the proc ttest output.

4. How could you use the where= statement to avoid doing problem 2 above?

5. Briefly answer the following questions. Do the 3 species have different mean sepal lengths? Does your answer depend on whether the significance level is 0.05 or 0.01?

3.5 Project 5

This project uses column-wise input statements to construct a data set from a file. Categorical response data is analyzed using proc freq. The data is from the file prodpref.dat.

A study was used to determine whether a cost reduced product was equally preferred to the current product. 200 judges were asked their preference on the pair of samples, coded 27 and 45. One half of the group tasted sample 27 first and 45 second, the other half tasted the samples in the reverse order. Each judge was also asked to rate six qualities of these samples on a 0 to 6 scale (0=no response,1=excellent,... ,6 poor).

The preferences of each judge were recorded as: 1, prefers first sample tasted 2, prefers second sample tasted 3, no preference. The data are in the file prodpref.dat. and are coded in the form

```
1    27    45    333424    543424    1
2    27    45    443324    533555    1
3    27    45    543314    543414    1
4    27    45    552232    343333    2
```

The first entry is the judge number, the second entry is number of the sample tasted first, the third entry is the number of the sample tasted second, the fourth entry is the six ratings for the first sample tasted, the fifth entry is the six ratings of the second sample tasted, and the last entry is the preference.

1. How many of the judges preferred sample 27? You may want to use some if-then statements in constructing an appropriate data set and also the n option in proc means. You could also use proc freq. Turn in a nice printout to back up your answer. There is no need to print out the data set here.

2. Now use chi-square tests to compare the distribution of the rankings of the two products for each of the six aspects of quality. The chi-square test is easily done using proc freq once the data is in a suitable form. The information on column-wise input and do loops may be useful. Are the two products interchangable? Back up your comments with some nice printouts, including a printout of your final program. There is no need to print out the data set here.

3.6 Project 6

Here the update statement is used to update the values in a SAS data set. The data set modified is the product preferences data (prodpref.dat) of Project 5.

ASSIGNMENT:

1. Suppose it was determined that some of the data is incorrect. The following are the correct entries:

```
10     45    27    323333    133333    2
29     45    27    323323    113333    2
106    27    45    313433    433434    1
107    27    45    443333    443333    3
108    45    27    513433    413533    2
109    45    27    513433    333333    3
110    45    27    433434    234333    3
111    45    27    654544    543433    1
```

Correct the original data set and then repeat the analysis of Project 5 with the corrected data. The information about update should be helpful. Turn in a printout as in Project 5.

3.7 Project 7

Proc gplot and proc g3d are used here to produce some relatively simple graphs.

1. The file skin.dat contains data on skin cancer incidence per 100,000 and sunspot activity. The structure of each line in the file is: Year Male Total Sunspot, where Male is the incidence of skin cancer in males, Total is the incidence of skin cancer in the total population, and Sunspot is a measure of sunspot activity. Turn in a single graph showing a nice plot of the Male versus Year data and the Total versus Year data using a different line style for the male and total data. Use the /overlay option of proc gplot.

2. Make a nice line graph of Male versus Year and Sunspot versus Year on the same graph using two different vertical axes for the two measurements.

3. Make a nice graph of the Total versus Year and Sunspot versus Year data on the same graph with a single vertical axis, this time with a line graph for Total and a needle graph for Sunspot.

4. The file tomato.dat contains the yield of tomato plants grown in 5 different types of soil with 3 different types of fertilizer. Use proc g3d to make a plot of yield as a function of soil and fertilizer. Does a 2 factor additive model for yield as a function of soil and fertilizer make sense here?

3.8 Project 8

Here basic features of proc glm are studied. The focus is on the basics of the analysis of variance.

The iris data, contained in iris.dat, will again be used. In Project 4 a comparison of the means of the sepal lengths measurements was made on a pairwise basis.

A more powerful method of analysis is to test simultaneously for the equality of the 3 means. This is done using a one way analysis of variance.

ASSIGNMENT:

1. In order to do a one way ANOVA using proc glm it is necessary to have a variable to be used in the class statement. In this context the variable should have 3 values depending on the species from which the observation is taken. Create a data set containing the sepal length measurements and an appropriate new variable. The techniques of do loops will be helpful. You don't need to turn in a copy of the data set but do turn in a copy of the program used to create the data set.

2. Use proc glm to test for equality of the means for the 3 species. Also compute Bonferroni type simultaneous confidence intervals for the comparison of the 3 means at both the 90% and 99% confidence level. Turn in a nice copy of this printout.

3. Briefly, are the 3 mean lengths different? Which pairs are different at the 90% level? At the 99% level?

4. Create a SAS data set from the file pig.dat as mentioned in the proc glm discussion. The missover option of the infile statement may be useful. You don't need to turn in a copy of the data set. Then test for the presence of litter effect. Is there litter effect at the 5% level of significance? Submit a nice printout to back up your conclusions.

3.9 Project 9

Here some of the advanced features of proc glm, as discussed in proc glm (advanced) are explored. Project 8 should be completed before starting this one.

1. The main thrust of the analysis of the tomato data is to identify the combination of soil and fertilizer which produces the greatest yield. Carry out an appropriate analysis and describe your conclusion. How dependent are your results on the chosen significance level? Turn in nice print outs of the analysis (not the data set) to back up your answers. Also turn in a copy of your program.

2. The file skin.dat contains data on skin cancer incidence. An investigator is considering 2 models for cancer incidence as a function of time: linear in time and exponential in time. Use proc glm to fit both types of models. Which model is better? Turn in nice print outs of the analysis (not the data set) to back up your answers. If you have done Project7, one or two nice graphs could be useful here too.

3.10 Project 10

Here proc glm is used to perform multivariate analysis of variance (MANOVA).

The Fisher Iris Data, introduced in Project 3 and contained in the file iris.dat, is analyzed.

1. Use the manova statement in proc glm, as described in Proc GLM for MANOVA to test the hypothesis that the vector of 4 measurements of each species of iris have the same mean vector. What is your conclusion? Turn in print outs of only the relevant parts of your analysis, and also of your program. What are the implications of your analysis on the prospect of using discriminant analysis based on the 4 measurements to distinguish between species?

2. Test the hypothesis that the length measurements are twice the width measurements for both sepal and petal. Is this hypothesis accepted? Test the hypothesis that the sepal length means are the same for all 3 species. Is this the same as the univariate one way ANOVA of Project 8? Turn in suitable printouts to support your answers. Also turn in a copy of your program.

3. An experimenter believes that the sepal width is a linear function of the other 3 measurments. Test this hypothesis; describe your methods and conclusions. Turn in suitable printouts to support your answers and also a copy of your program.

3.11 Project 11

Some time domain methods (Box-Jenkins methods) for the analysis of time series using proc arima are used to analyze 2 sets of data.

The first data set, milk.dat, contains milk production data by month for the years 1962-1975 (Cryer). The second data set, airline.dat, contains the yearly millions of air passenger miles flown each month within the United States for the years 1960-1977 (Cryer).

Suggestion: If you plan to use high quality graphs in your work, read about proc gplot and do Project 7 before starting this assignment.

1. Make a SAS data set containing the milk production data for the years 1962-1974, leaving the 1975 data out of the analysis. You don't need to turn in a print out of the data set. Use proc arima to fit an ARIMA model to the time series and use this model to predict milk production for 1975. Print out a copy of the final analysis and forecasts for 1975 using the model you select. Based on the printout, how adequate is your model for the data? Turn in a copy of your program too.

2. Conduct an analysis of the airline data similar to that of problem 1, leaving the 1977 data out of the analysis. The raw airline data shows increasing variablility with trend, so the data should be transformed prior to analysis in order to obtain a stationary time series. Add a variable logmiles to the data set containing the raw airline data, and conduct the analysis using proc arima on this new variable. Remember to exponentiate the forecasts from this analysis to obtain forecasts of air passenger miles.

3. If you are using proc gplot to produce high quality graphs, make graphs of the original milk data and its sample autocorrelation function. Do the same for the raw airline data, the log of the airline data, and the autocorrelation function of the log of the airline data.

3.12 Project 12

Proc spectra is used to identify periodicities in time series data. The data sets analyzed here are the airline passenger miles data and the milk production data used in Project 11. The raw data is found in the files airline.dat and milk.dat.

Suggestion: If you plan to use high quality graphs in your work, read about proc gplot and do Project 7 before doing this assignment.

1. Use proc spectra to find the periodogram of the milk production data. Since the data is monthly, it should be no surprise to find the data is periodic with period 12. How is this seen from your analysis? What does the test for white noise show for the original data? The periodicity suggests differencing the original series at lag 12 should produce a stationary time series. Create a new data set containing the differenced series. You may want to use the dif operator. Are there any remaining periodicities in the difference series? Is the new series white noise? Choose appropriate weights and compute the spectral density estimate of the differenced series. Turn in your a nice copy of your program and submit nice print outs from SAS to back up your answers to the questions above. There is no need to turn in a printed copy of the data.

2. Repeat the analysis of problem 1 using the airline data. Recall that a logarithmic transform of the raw data is suggested before analysis. Are there periodicities? How can they be removed? Submit nice print outs of your program and analysis from SAS to back up your answers. There's no need to print out the data.

3. If you can use proc gplot, make some nice graphs of the periodogram and spectral density estimates to supplement your analysis in the preceding problems and turn them in also.

3.13 Project 13

Here the basic features of using proc discrim for discriminant analysis are studied.

The Fisher Iris data contained in iris.dat, and examined in Project 3, is used again. The objective is to determine if the species of a given iris plant can be determined simply on the basis of the four measurements.

1. What statistical test should always be done before a discriminant analysis is attempted? WHY?

2. Assume the measurements for each species are a sample from a multivariate normal population. Assume also that the covariance matrices for the 3 populations are equal. Carry out a discriminant analysis using proc discrim. Is discrimination successful? What type of discrimination rule is used? Which observations are misclassified? Submit nice print outs from SAS to back up your answers. Turn in a nice print out of your program too. There's no need to print out the data.

3. Repeat the analysis of problem 1, but this time test (alpha=0.01) for equality of covariance matrices of the 3 populations. Is the discrimination rule different? If so, what is the discrimination rule now and how successful is it? Submit nice print outs from SAS to back up your answers.

4. Instead of using the full data set to devise a discrimination rule, use only the first half of the observations for each species. For simplicity, assume normality and equality of covariance matrices. What is the discrimination rule? Use this discrimination rule on the remaining half of the observations. What are your estimated error rates? How do these error rates compare to the estimated rates in problem 1? Submit nice print outs from SAS to back up your answers. Turn in a copy of your program too–but not the data itself.

3.14 Project 14

In this project, dBase files are used both in place of SAS datasets and to create SAS datasets. The details of how this is done is in the notes about dBase and DIF Files.

This project should be done only if you are using SAS 6.08 release TS 407, or later, for the Microsoft Windows environment.

ASSIGNMENT:

1. The dBase file iris.dbf contains some of the Fisher Iris data analyzed previously. Make a working copy of this dBase file. Use the technique of reading dBase files directly (without converting the dBase file to a SAS data set) to create a SAS data set containing two variables. The two variables in the SAS data set are to be the sum of sepal length and sepal width and the sum of petal length and petal width. Print out a nice copy of this SAS data set.

2. Read iris.dbf into a SAS dataset called newiris. Use proc contents to print out a nice listing of the contents of the SAS data set.

3. Take any SAS data set you have created and turn it into a dBase file. Then make a nice 1 page printout of (part) of the contents of that dBase file by using dBase (or a similar program).

3.15 Project 15

This project does some advanced data step processing as well as a repeated measures analysis of variance.

ASSIGNMENT:

1. The file medical.dat contains the cholesterol readings of 4 patients over an eight week period. The objective is to determine whether the drug administered over this period was effective in reducing the cholesterol level of the patients. The code 999 was used in the data set to indicate a missing value. Use an array to replace 999 with the SAS missing value symbol . (period) in the data set. You should also use an appropriate informat to read in the dates correctly. Turn in a copy of your program.

2. For a preliminary analysis only the first and last cholesterol reading for each patient will be used. Create a new data set containing 3 variables: patient name, first cholesterol reading, and last cholesterol reading. The first. and last. functions should be used on a sorted version of the original data set. You may also want to use retain. Turn in a copy of your program and a print out of the new data set.

3. Since each patient contributes 2 readings a repeated measures analysis is appropriate. Test the null hypothesis that the drug has no effect using a repeated measures analysis. Is the drug effective? Turn in a copy of your program with a brief explanation of your conclusion.

4. Conduct the repeated measures analysis using all of the cholesterol data. What is your conclusion? Turn in a copy of your program with a brief explanation of your conclusion.

4 Data Sets

The following are the data sets used in the projects. For each data set, copy the data from this document and paste the data into a text file, then save the text file with the indicated name. The data should then be ready to use in the projects. Use a simple text editor, such as Notepad, so that the file is a simple text file and does not contain any special formatting commands.

4.1 Data File: airline.dat

This data, from Cryer, is the monthly U.S. air passenger miles (in millions) for the period January 1960 to December 1977 (n=216).

The data below should be placed in the text file airline.dat.

```
1960   2.42   2.14   2.28   2.50   2.44   2.72   2.71   2.74   2.55   2.49   2.13   2.28
1961   2.35   1.82   2.40   2.46   2.38   2.83   2.68   2.81   2.54   2.54   2.37   2.54
1962   2.62   2.34   2.68   2.75   2.66   2.96   2.66   2.93   2.70   2.65   2.46   2.59
1963   2.75   2.45   2.85   2.99   2.89   3.43   3.25   3.59   3.12   3.16   2.86   3.22
1964   3.24   2.95   3.32   3.29   3.32   3.91   3.80   4.02   3.53   3.61   3.22   3.67
1965   3.75   3.25   3.70   3.98   3.88   4.47   4.60   4.90   4.20   4.20   3.80   4.50
1966   4.40   4.00   4.70   5.10   4.90   5.70   3.90   4.20   5.10   5.00   4.70   5.50
1967   5.30   4.60   5.90   5.50   5.40   6.70   6.80   7.40   6.00   5.80   5.50   6.40
1968   6.20   5.70   6.40   6.70   6.30   7.80   7.60   8.60   6.60   6.50   6.00   7.60
1969   7.00   6.00   7.10   7.40   7.20   8.40   8.50   9.40   7.10   7.00   6.60   8.00
1970   10.45   8.81   10.61   9.97   10.69   12.40   13.38   14.31   10.90   9.98 9.20   10.94
1971   10.53   9.06   10.17   11.17   10.84   12.09   13.66   14.06   11.14   11.10 10.00   11.98
1972   11.74   10.27   12.05   12.27   12.03   13.95   15.10   15.65   12.47   12.29 11.52   13.08
1973   12.50   11.05   12.94   13.24   13.16   14.95   16.00   16.98   13.15   12.88 11.99   13.13
1974   12.99   11.69   13.78   13.70   13.57   15.12   15.55   16.73   12.68   12.65 11.18   13.27
1975   12.64   11.01   13.30   12.19   12.91   14.90   16.10   17.30   12.90   13.36 12.26   13.93
1976   13.94   12.75   14.19   15.67   14.66   16.21   17.72   18.15   14.19   14.33 12.99   15.19
1977   15.09   12.94   15.46   15.39   15.34   17.02   18.85   19.49   15.61   16.16 14.84   17.04
```

4.2 Data File: iris.dat

This is the famous ‘Fisher Iris Data’ (from Andrews and Herzberg).

The data below, taken from Fisher (1936), are the measurements of the sepal length and width and petal length and width in centimetres of fifty plants for each of three types of iris; Iris setosa, Iris versicolor and Iris virginica. These data are commonly referred to as the “Fisher Iris Data”. Although the data were collected by Dr. Edgar Anderson, R. A. Fisher published the data on Iris setosa and Iris versicolor to demonstrate the use of discriminant functions. The Iris virginica data are used to extend Fisher’s technique and to test Randolph’s (1934) hypothesis that Iris versicolor is a polyploid hybrid of the two other species which is related to the fact that Iris setosa is a diploid species with 38 chromosomes, Iris virginica a tetraploid and Iris versicolor having 108 chromosomes is a hexaploid.

The data below should be placed in the text file iris.dat. If you wish to do the projects using dBase files, you should import the data from the text file into a program that can output dbf files and save that database as iris.dbf.

```
5.1  3.5  1.4  0.2  7.0  3.2  4.7  1.4  6.3  3.3  6.0  2.5
4.9  3.0  1.4  0.2  6.4  3.2  4.5  1.5  5.8  2.7  5.1  1.9
4.7  3.2  1.3  0.2  6.9  3.1  4.9  1.5  7.1  3.0  5.9  2.1
4.6  3.1  1.5  0.2  5.5  2.3  4.0  1.3  6.3  2.9  5.6  1.8
5.0  3.6  1.4  0.2  6.5  2.8  4.6  1.5  6.5  3.0  5.8  2.2
5.4  3.9  1.7  0.4  5.7  2.8  4.5  1.3  7.6  3.0  6.6  2.1
4.6  3.4  1.4  0.3  6.3  3.3  4.7  1.6  4.9  2.5  4.5  1.7
5.0  3.4  1.5  0.2  4.9  2.4  3.3  1.0  7.3  2.9  6.3  1.8
4.4  2.9  1.4  0.2  6.6  2.9  4.6  1.3  6.7  2.5  5.8  1.8
4.9  3.1  1.5  0.1  5.2  2.7  3.9  1.4  7.2  3.6  6.1  2.5
5.4  3.7  1.5  0.2  5.0  2.0  3.5  1.0  6.5  3.2  5.1  2.0
4.8  3.4  1.6  0.2  5.9  3.0  4.2  1.5  6.4  2.7  5.3  1.9
4.8  3.0  1.4  0.1  6.0  2.2  4.0  1.0  6.8  3.0  5.5  2.1
4.3  3.0  1.1  0.1  6.1  2.9  4.7  1.4  5.7  2.5  5.0  2.0
5.8  4.0  1.2  0.2  5.6  2.9  3.6  1.3  5.8  2.8  5.1  2.4
5.7  4.4  1.5  0.4  6.7  3.1  4.4  1.4  6.4  3.2  5.3  2.3
5.4  3.9  1.3  0.4  5.6  3.0  4.5  1.5  6.5  3.0  5.5  1.8
5.1  3.5  1.4  0.3  5.8  2.7  4.1  1.0  7.7  3.8  6.7  2.2
5.7  3.8  1.7  0.3  6.2  2.2  4.5  1.5  7.7  2.6  6.9  2.3
5.1  3.8  1.5  0.3  5.6  2.5  3.9  1.1  6.0  2.2  5.0  1.5
5.4  3.4  1.7  0.2  5.9  3.2  4.8  1.8  6.9  3.2  5.7  2.3
5.1  3.7  1.5  0.4  6.1  2.8  4.0  1.3  5.6  2.8  4.9  2.0
4.6  3.6  1.0  0.2  6.3  2.5  4.9  1.5  7.7  2.8  6.7  2.0
5.1  3.3  1.7  0.5  6.1  2.8  4.7  1.2  6.3  2.7  4.9  1.8
4.8  3.4  1.9  0.2  6.4  2.9  4.3  1.3  6.7  3.3  5.7  2.1
5.0  3.0  1.6  0.2  6.6  3.0  4.4  1.4  7.2  3.2  6.0  1.8
5.0  3.4  1.6  0.4  6.8  2.8  4.8  1.4  6.2  2.8  4.8  1.8
5.2  3.5  1.5  0.2  6.7  3.0  5.0  1.7  6.1  3.0  4.9  1.8
5.2  3.4  1.4  0.2  6.0  2.9  4.5  1.5  6.4  2.8  5.6  2.1
4.7  3.2  1.6  0.2  5.7  2.6  3.5  1.0  7.2  3.0  5.8  1.6
4.8  3.1  1.6  0.2  5.5  2.4  3.8  1.1  7.4  2.8  6.1  1.9
5.4  3.4  1.5  0.4  5.5  2.4  3.7  1.0  7.9  3.8  6.4  2.0
5.2  4.1  1.5  0.1  5.8  2.7  3.9  1.2  6.4  2.8  5.6  2.2
5.5  4.2  1.4  0.2  6.0  2.7  5.1  1.6  6.3  2.8  5.1  1.5
4.9  3.1  1.5  0.2  5.4  3.0  4.5  1.5  6.1  2.6  5.6  1.4
5.0  3.2  1.2  0.2  6.0  3.4  4.5  1.6  7.7  3.0  6.1  2.3
5.5  3.5  1.3  0.2  6.7  3.1  4.7  1.5  6.3  3.4  5.6  2.4
4.9  3.6  1.4  0.1  6.3  2.3  4.4  1.3  6.4  3.1  5.5  1.8
4.4  3.0  1.3  0.2  5.6  3.0  4.1  1.3  6.0  3.0  4.8  1.8
5.1  3.4  1.5  0.2  5.5  2.5  4.0  1.3  6.9  3.1  5.4  2.1
5.0  3.5  1.3  0.3  5.5  2.6  4.4  1.2  6.7  3.1  5.6  2.4
4.5  2.3  1.3  0.3  6.1  3.0  4.6  1.4  6.9  3.1  5.1  2.3
4.4  3.2  1.3  0.2  5.8  2.6  4.0  1.2  5.8  2.7  5.1  1.9
5.0  3.5  1.6  0.6  5.0  2.3  3.3  1.0  6.8  3.2  5.9  2.3
5.1  3.8  1.9  0.4  5.6  2.7  4.2  1.3  6.7  3.3  5.7  2.5
4.8  3.0  1.4  0.3  5.7  3.0  4.2  1.2  6.7  3.0  5.2  2.3
5.1  3.8  1.6  0.2  5.7  2.9  4.2  1.3  6.3  2.5  5.0  1.9
4.6  3.2  1.4  0.2  6.2  2.9  4.3  1.3  6.5  3.0  5.2  2.0
5.3  3.7  1.5  0.2  5.1  2.5  3.0  1.1  6.2  3.4  5.4  2.3
5.0  3.3  1.4  0.2  5.7  2.8  4.1  1.3  5.9  3.0  5.1  1.8
```

4.3 Data File: medical.dat

This data gives the date, patient name, cholesterol level, and blood pressure readings for 4 patients in a study of the effect of a cholesterol reducing drug. A code of 999 was used whenever a reading was missing.

The data below should be placed in the text file medical.dat.

```
02-02-93 Jason 306 145 97
02-02-93 John  212 132 85
02-09-93 John  215 129 89
02-09-93 Jason 295 142 85
02-09-93 Mary  303 150 102
02-09-93 Beth  323 145 97
02-16-93 Beth  320 149 104
02-16-93 Jason 285 140 87
02-16-93 Mary  999 145 97
02-16-93 John  220 135 93
02-23-93 Mary  307 149 99
02-23-93 John  215 133 92
02-23-93 Beth  319 143 99
02-23-93 Jason 288 143 90
03-02-93 Mary  310 155 103
03-02-93 Jason 285 140 88
03-02-93 Beth  315 157 106
03-02-93 John  999 133 92
03-09-93 Jason 286 144 93
03-09-93 Mary  305 149 95
03-09-93 John  202 135 97
03-09-93 Beth  312 147 94
03-16-93 Mary  295 152 103
03-16-93 Jason 284 143 94
03-16-93 Beth  305 158 108
03-16-93 John  205 137 92
03-23-93 Jason 285 140 90
03-23-93 Mary  290 153 94
03-23-93 Beth  296 999 999
03-23-93 John  206 132 89
03-30-93 Mary  293 149 98
04-06-93 Mary  290 150 102
04-06-93 Beth  293 149 98
```

4.4 Data File: milk.dat

This data from Cryer gives the average monthly milk production per cow for the period January 1962 to December 1975 (n=168). The columns are the months.

The data below should be placed in the text file milk.dat.

```
1962    589    561    640    656    727    697    640    599    568    577    553    582
1963    600    566    653    673    742    716    660    617    583    587    565    598
1964    628    618    688    705    770    736    678    639    604    611    594    634
1965    658    622    709    722    782    756    702    653    615    621    602    635
1966    677    635    736    755    811    798    735    697    661    667    645    688
1967    713    667    762    784    837    817    767    722    681    687    660    698
1968    717    696    775    796    858    826    783    740    701    706    677    711
1969    734    690    785    805    871    845    801    764    725    723    690    734
1970    750    707    807    824    886    859    819    783    740    747    711    751
1971    804    756    860    878    942    913    869    834    790    800    763    800
1972    826    799    890    900    961    935    894    855    809    810    766    805
1973    821    773    883    898    957    924    881    837    784    791    760    802
1974    828    778    889    902    969    947    908    867    815    812    773    813
1975    834    782    892    903    966    937    896    858    817    827    797    843
```

4.5 Data File: pig.dat

The file pig.dat contains data on the birth weight of poland china pigs in 8 litters (from Scheffe). The layout of the file is the litter number followed by the birth weights for the litter. Note that there are unequal litter sizes.

The data below should be placed in the text file pig.dat.

```
1  2.0  2.8  3.3  3.2  4.4  3.6  1.9  3.3  2.8  1.1
2  3.5  2.8  3.2  3.5  2.3  2.4  2.0  1.6
3  3.3  3.6  2.6  3.1  3.2  3.3  2.9  3.4  3.2  3.2
4  3.2  3.3  3.2  2.9  3.3  2.5  2.6  2.8
5  2.6  2.6  2.9  2.0  2.0  2.1
6  3.1  2.9  3.1  2.5
7  2.6  2.2  2.2  2.5  1.2  1.2
8  2.5  2.4  3.0  1.5
```

4.6 Data File: prodpref.dat

This data is from Andrews and Herzberg.

The purpose of this study was to determine whether a cost reduced product was equally preferred to the current product. A Paired Comparison Central Location Preference test was run where 200 judges were asked their preference on the pair of samples, coded 27 and 45. One half of the group tasted sample 27 first and 45 second, the other half tasted the samples in the reverse order. Besides being asked their preference, each judge was asked to rate six qualities of these samples on a 0 to 6 scale. The final column contains the preference: 1, prefers first sample tasted 2, prefers second sample tasted 3, no preference.

The data below should be placed in the text file prodpref.dat.

```
1     27    45    333424    543424    1
2     27    45    443324    533555    1
3     27    45    543314    543414    1
4     27    45    552232    343333    2
5     27    45    333333    223333    2
6     27    45    434233    543414    1
7     27    45    234333    433433    1
8     27    45    134333    533424    1
9     27    45    544533    465533    1
10    27    45    323333    133333    2
11    27    45    553524    443434    2
12    27    45    333333    443222    1
13    27    45    433434    542414    1
14    27    45    533442    553244    3
15    27    45    344333    244333    2
16    27    45    333433    553532    1
17    27    45    223333    554414    1
18    27    45    543252    443343    2
19    27    45    245433    665525    1
20    27    45    433424    333333    2
21    27    45    444242    443342    2
22    27    45    433424    533525    1
23    27    45    333324    433223    1
24    27    45    433333    443333    1
25    27    45    243424    544524    1
26    27    45    444443    243434    1
27    27    45    553424    653555    1
28    27    45    114333    454424    1
29    27    45    323323    113333    2
30    27    45    223333    533442    1
31    27    45    543424    443324    2
32    27    45    553543    553434    2
33    27    45    434434    323323    2
34    27    45    543432    333333    2
35    27    45    343333    543333    2
36    27    45    443333    543414    1
37    27    45    124333    424242    1
38    27    45    554434    333333    2
39    27    45    554434    234423    2
40    27    45    223333    333224    1
41    27    45    513221    443242    1
42    27    45    544434    664233    1
43    27    45    443454    550554    1
44    27    45    443333    323333    2
45    27    45    113333    113333    2
46    27    45    224433    534333    1
47    27    45    334424    342433    1
48    27    45    443333    443333    2
49    27    45    423333    533222    1
50    27    45    433424    643524    1
51    27    45    443333    543524    1
52    27    45    434434    433353    2
53    27    45    333333    443242    1
54    27    45    343223    654251    1
55    27    45    113333    544434    1
56    27    45    214433    113333    2
57    27    45    344332    444232    1
58    27    45    534415    223333    2
59    27    45    333352    633151    1
60    27    45    544524    443424    2
61    27    45    544524    544525    1
62    27    45    554554    444333    2
63    27    45    544434    543404    3
64    27    45    665151    333333    2
65    27    45    223333    233333    3
66    27    45    135355    444142    1
67    27    45    562254    652115    1
```

68	27	45	433323	643132	1
69	27	45	365333	354333	3
70	27	45	433323	433332	1
71	27	45	113333	443424	1
72	27	45	344333	533524	1
73	27	45	453333	553424	1
74	27	45	543241	444333	2
75	27	45	453345	333333	2
76	27	45	454434	443333	2
77	27	45	344333	233333	2
78	27	45	444443	654544	1
79	27	45	223434	233434	3
80	27	45	443424	333333	2
81	27	45	223333	424242	1
82	27	45	223333	643555	1
83	27	45	213333	322242	1
84	27	45	113333	113333	3
85	27	45	333423	432343	1
86	27	45	464434	423333	2
87	27	45	333333	433434	1
88	27	45	333423	233333	2
89	27	45	454432	233333	2
90	27	45	223333	524434	1
91	27	45	333334	543424	1
92	27	45	255435	455424	2
93	27	45	443433	444323	1
94	27	45	544424	533223	3
95	27	45	443333	333333	2
96	27	45	434323	664425	1
97	27	45	343423	542212	1
98	27	45	553514	654424	1
99	27	45	333333	543424	1
100	27	45	434435	333333	1
101	45	27	443253	333333	2
102	45	27	342333	433333	1
103	45	27	224324	224424	1
104	45	27	444433	333323	2
105	45	27	443323	543215	1
106	45	27	313433	433434	1
107	45	27	443333	443333	3
108	45	27	513433	413533	1
109	45	27	513433	333333	2
110	45	27	433434	234333	2
111	45	27	654544	543433	2
112	45	27	443233	543232	1
113	45	27	543433	444343	2
114	45	27	333333	442232	1
115	45	27	333333	333333	2
116	45	27	133333	543242	1
117	45	27	553424	553324	1
118	45	27	444434	333323	2
119	45	27	523343	453451	2
120	45	27	543333	223333	2
121	45	27	554435	653434	1
122	45	27	423222	113333	2
123	45	27	413333	033241	1
124	45	27	554424	554424	3
125	45	27	544424	444433	2
126	45	27	343433	544435	1
127	45	27	334454	233333	2
128	45	27	433332	333333	2
129	45	27	333434	333323	2
130	45	27	223333	433333	2
131	45	27	323433	443224	1
132	45	27	113353	313242	1
133	45	27	554424	554424	2
134	45	27	223333	444424	1
135	45	27	343433	323223	2
136	45	27	443333	443323	1
137	45	27	454232	323333	2
138	45	27	333333	443242	1
139	45	27	333333	443242	1
140	45	27	553243	333333	2
141	45	27	643211	664212	3
142	45	27	113333	433333	1
143	45	27	144333	423242	1
144	45	27	113433	413433	1
145	45	27	233333	433242	1
146	45	27	444434	343333	2
147	45	27	554222	343333	2
148	45	27	223333	544444	1
149	45	27	213233	613144	1
150	45	27	454424	443524	1
151	45	27	543544	433333	2
152	45	27	344323	654534	1
153	45	27	454333	543325	1
154	45	27	333334	533422	1
155	45	27	643151	545222	2
156	45	27	223333	434232	1
157	45	27	554433	444333	2
158	45	27	454233	353333	2

```
159  45  27  224333  334333  3
160  45  27  344342  243333  2
161  45  27  443342  333333  2
162  45  27  343433  455242  1
163  45  27  434242  443223  3
164  45  27  443424  443424  1
165  45  27  354333  554444  1
166  45  27  544452  443333  2
167  45  27  213333  544442  1
168  45  27  523252  113333  2
169  45  27  413433  552232  1
170  45  27  553433  653424  1
171  45  27  213333  333333  3
172  45  27  523424  433323  2
173  45  27  643555  433333  2
174  45  27  232522  544552  1
175  45  27  344433  444323  1
176  45  27  552534  552524  1
177  45  27  123333  524221  1
178  45  27  333433  423333  1
179  45  27  113333  223333  1
180  45  27  443252  544252  1
181  45  27  443335  543232  1
182  45  27  444434  444444  1
183  45  27  543242  543223  2
184  45  27  323333  644242  1
185  45  27  353433  543442  1
186  45  27  553452  553433  2
187  45  27  333433  543442  1
188  45  27  443333  543124  1
189  45  27  443333  443333  2
190  45  27  323353  544455  1
191  45  27  623151  523251  2
192  45  27  523515  523422  3
193  45  27  443333  443232  1
194  45  27  333423  543242  1
195  45  27  344333  334333  2
196  45  27  443335  552151  1
197  45  27  444433  333333  2
198  45  27  444433  444333  2
199  45  27  444433  343323  2
200  45  27  223333  443242  1
```

4.7 Data File: skin.dat

This data is from Andrews and Herzberg.

The aetiology of melanoma is complex and may include the influences of trauma, heredity and hormonal activity. In particular, exposure to solar radiation may be involved in the pathogenesis of melanoma. Melanoma is more common in fair-skinned individuals and most frequent in skin sites exposed to the sun. In white populations melanoma is more common in areas closer to the equator where the intensity of solar radiation is higher. Data from various parts of the world suggest that the incidence of melanoma is increasing. The data below, giving age-adjusted melanoma incidence, are from the Connecticut Tumor Registry from 1936-1972. Connecticut has the longest record of state population-based cancer statistics in the United States of America. The data also includes the sunspot relative number. Houghton, Munster and Viola (1978) have shown that the age-adjusted incidence rate for malignant melanoma in the state of Connecticut has risen since 1935 and that superimposed on the rise are 3-5 year periods in which the rise in the rate of incidence is excessive. These periods have a cycle of 8-11 years and follow times of maximum sunspot activity. The relationship between solar cycles and melanoma supports the hypothesis that melanoma is related to sun exposure and provides evidence that solar radiation may trigger the development of clinically apparent melanoma. The columns are the year, male incidence, total incidence, and sunspot relative index. The incidence are rates per 100,000.

The data below should be placed in the text file skin.dat.

```
1936  1.0  0.9  40
```

1937	0.8	0.8	115
1938	0.8	0.8	100
1939	1.4	1.3	80
1940	1.2	1.4	60
1941	1.0	1.2	40
1942	1.5	1.7	23
1943	1.9	1.8	10
1944	1.5	1.6	10
1945	1.5	1.5	25
1946	1.5	1.5	75
1947	1.6	2.0	145
1948	1.8	2.5	130
1949	2.8	2.7	130
1950	2.5	2.9	80
1951	2.5	2.5	65
1952	2.4	3.1	20
1953	2.1	2.4	10
1954	1.9	2.2	5
1955	2.4	2.9	10
1956	2.4	2.5	60
1957	2.6	2.6	190
1958	2.6	3.2	180
1959	4.4	3.8	175
1960	4.2	4.2	120
1961	3.8	3.9	50
1962	3.4	3.7	35
1963	3.6	3.3	20
1964	4.1	3.7	10
1965	3.7	3.9	15
1966	4.2	4.1	30
1967	4.1	3.8	60
1968	4.1	4.7	105
1969	4.0	4.4	105
1970	5.2	4.8	105
1971	5.3	4.8	80
1972	5.3	4.8	65

4.8 Data File: sleep.dat

This data is taken from Finkelstein and Levin.

The Able Company claimed in TV ads that taking its product Super-Drowsy reduced the time to fall asleep by 46% over the time necessary without pills. Able based this claim on a sleep study. Persons were asked to record how long they took to fall asleep ('sleep latency') in minutes, and their average for a week was calculated. In the next week these persons received Super-Drowsy and recorded their sleep latency. The following table gives the average sleep latency for each of the 73 persons first for the week without Super Drowsy and then for the week with Super Drowsy.

The data below should be placed in the text file sleep.dat.

1	61.07	20.36	26	16.07	7.50	51	41.79	11.79
2	46.07	37.50	27	33.21	26.79	52	22.50	22.50
3	84.64	61.07	28	51.43	20.36	53	39.64	18.21
4	31.07	9.64	29	28.93	18.21	54	78.21	26.79
5	54.64	28.93	30	139.29	37.50	55	46.07	16.07
6	26.79	11.79	31	78.21	39.64	56	46.07	28.93
7	58.93	31.07	32	43.93	35.36	57	61.07	33.21
8	13.93	9.64	33	111.43	7.50	58	39.64	28.93
9	71.79	35.36	34	56.79	31.07	59	56.79	18.21
10	82.50	33.21	35	106.07	43.93	60	43.93	24.64
11	37.50	24.64	36	98.57	77.14	61	43.93	24.64
12	35.36	46.07	37	43.93	70.00	62	31.07	24.64
13	37.50	9.64	38	114.64	26.79	63	46.06	24.64
14	114.64	16.07	39	39.64	33.21	64	22.50	9.64
15	35.36	9.64	40	91.07	31.07	65	50.36	41.79
16	73.93	35.36	41	18.21	20.36	66	41.79	18.21
17	17.50	11.79	42	63.21	18.21	67	35.36	18.21
18	94.29	20.36	43	37.50	20.36	68	65.36	78.21
19	22.50	9.64	44	41.79	22.50	69	37.50	31.50
20	58.93	30.00	45	40.00	43.93	70	48.21	9.64
21	46.07	100.71	46	50.36	41.79	71	102.86	43.93
22	31.07	9.64	47	48.21	60.00	72	86.79	43.93
23	62.14	20.36	48	63.21	50.36	73	31.07	9.64
24	20.36	13.93	49	54.64	20.36			
25	56.79	32.14	50	73.93	13.93			

4.9 Data File: tomato.dat

The following data is from Hicks. Tomato plants were grown in a greenhouse under treatments consisting of combinations of soil type and fertilizer type. A completely randomized two factor design was used with two replications per cell. The following data on yield in kilograms of tomatoes were obtained for the 30 plants under study.

```
            Fertilizer Type
Soil Type    1    2    3
I           5,7  5,5  3,5
II          5,9  1,3  2,2
III         6,8  4,8  2,4
IV          7,11 7,9  3,7
V           6,9  4,6  3,5
```

Below is the contents that should be placed in the text file tomato.dat:

```
I           5 7  5 5  3 5
II          5 9  1 3  2 2
III         6 8  4 8  2 4
IV          7 11 7 9  3 7
V           6 9  4 6  3 5
```

5 Language Reference

Here is a list of the SAS language terms used in this document.

- Arrays
- @ Symbol
- By Statement
- Cards Statement
- Data Catalogs
- Character Variables
- Command Prompt
- Continue Statement
- Data Statement
- Data Step
- Display Manager Statement
- Do Loops
- Drop Statement
- Filename Statement

5.1 @ Symbol

The @ symbol at the end of an input statement in the data step tells SAS that not to move to the next data card after reading the values for the variables in the current input statement.

If sets of input values are split accross lines, or if single data lines are used accross iterations of the data step, @@ should be used at the end of the input statement.

The @ symbol is also used during column input to specify the columns of the data card at which values for a variable are to be read.

5.2 @@ Symbol

If values for the variables in an input statement are split accross iterations of the data step the @@ symbol should appear at the end of the input statement to hold the data card for the next iteration of the data step and prevent data from being lost. Typically this situation occurs when there is either more or less data on each data card than is required for the input statement(s) in the data step.

5.3 ANOVA

Short form of ANalysis Of VAriance.

5.4 Arrays

The SAS array statement is used to easily tag variables in a data set for processing. The array statement is used in the data step. Example:

```
    array x{3} light water yield;
```

Following this statement `x{1}` refers to `light`, `x{2}` refers to `water` and `x{3}` refers to `yield`. This makes it easy to replace all zero values with the SAS missing value symbol by using the program

```
do i=1 to 3;
if x{i}=0 then x{i}= . ;
end;
```

If the data set has many variables then all numeric variables can be placed into an array with the statement

```
array x{*} _numeric_ ;
```

The number of entries in the array `x` is then obtained with `dim(x)`.

The first entry of an array has index 1.

For further information see the SAS Language Reference.

5.5 ARIMA

Short form of Auto-Regressive Integrated Moving Average process.

5.6 ARMA

Short form of Auto-Regressive Moving Average process.

5.7 ASCII

Literally, American Standard Code for Information Interchange, this means a data file that is to be interpreted as just a collection of characters, with no special word processor or other formatting commands in it.

5.8 By Statement

Used in conjunction with a SAS procedure, the `by` statement specifies variables in the data set which are used to break the data set into groups for processing by the procedure. The data set is assumed to be sorted in ascending order according to the `by` variables, but this can be overridden through the use of the `descending` modifier. The invocation

```
proc means
data=junk;
var z;
by x descending y;
run;
```

produces a separate analysis of the `z` values in the data set `junk` in each group determined by a specified `x` and `y` value. The groupings will appear in order of ascending `x` values, and within each `x` value in order of descending `y` values.

If the modifier `notsorted` appears in a by statement, processing proceeds for each group of consecutive data having the same value(s) of the by variable(s). This may produce many more groups than desired if the data is not sorted. Example:

`by x notsorted;`

The `by` statement also causes to temporary logical variables `first.` and `last.` to be created. These variables evaluate to true when the current data card corresponds to the first or last card of the by group. Example:

```
data final;
set first;
by id;
if first.id;
run;
```

The data set `final` will contain only the first entry of each `id` group.

For further information see the SAS Language Reference.

5.9 Cards Statement

The cards statement specifies that the values of the variables appearing in the input statement are to be read from the program editor window. The data cards are assumed to immediately follow the cards statement. Data cards should NOT be terminated with a semicolon.

The infile statement can accomplish the same thing as the cards statement by using the form `infile cards;`. This allows the use of the other options available with the infile statement.

5.10 Data Catalogs

SAS data sets are stored on disk in groups called a data catalog or library. When a data set is created, it can be given either a one level or a two level name.

Data sets given a one level name are part of the catalog called WORK and are erased when the SAS program is exited. Example: `data first;`.

Data sets with a two level name are stored on disk in a catalog or library which is referred to using a library reference or libref. Two steps are required in order to store a SAS data set on disk.

1. A SAS libref must be created using the libname statement.
2. The datastep must use a two level name. A two level name is of the form libref.name where name is a usual SAS data set name. Example:

```
libname jerry ‘c:\mydata\sasdata\’;

data jerry.grades;
...
```

This creates a SAS data set `grades` which will be stored on disk in the directory `c:\mydata\sasdata`. The data set will be available for use in future SAS sessions. Note that the libref will be erased at the end of the current SAS session.

The data set `grades` would be used in a new SAS session by first creating an appropriate libref, and then using the corresponding two level name when invoking a procedure. Example:

```
libname goody ‘c:\mydata\sasdata\‘;

proc means data=goody.grades;
...
```

Note that the libref is only required to point to the same subdirectory as when the data set was created.

As a general rule, only data sets that are difficult or time consuming to create or will be used in many different SAS sessions are stored on disk.

5.11 Character Variables

A character variable takes values which consists of letters or words. Numeric variables take number values.

5.12 Command Prompt

The command prompt can be used to issue commands to the underlying operating system, via the X command, and also to set certain SAS options, such as to assign a value to a function key (see keys or dm statement).

The command prompt is obtained by selecting GLOBALS–COMMAND–COMMAND from the menu bar. Note that some commands must be issued in a specific window. For such commands, make the menu pick from the menu bar in that window.

5.13 Continue Statement

The continue statement can be used in a do loop to branch to the bottom of the inner most enclosing do loop.

The return statement can be used to branch to the bottom of the data step program.

5.14 Data Set Options

Data set options are used in conjunction with the data set name to control the creation or reading of the data set. Data set options are enclosed in parentheses following the name of the data set.

The `drop=` or `keep=` options are used in the creation of a new data set and specify variables used in the data step which are dropped from (or kept in) the data set itself. Example:

```
data dumb(drop=x y);
input x y z;
cards;
...
run;
```

The data set `dumb` will contain only the varible `z`. The `keep=` option behaves similarly.

Similar behavior can be obtained by using the drop and keep statements. The data set options are preferred.

The additional options `firstobs=`, `obs=`, and `where=` control how the data set is read. These options, together with `drop=` and `keep=`, are most often used in the proc step to restrict analysis to certain observations in the data set. The `where=` option can not be used when `firstobs=` or `obs=` is used. Examples:

```
proc means data=dumb(where=(x<5 and y>9) drop=z);
run;

proc means data=dumb(firstobs=5 obs=20);
run;
```

The first example analyzes only the observations in `dumb` for which the two restrictions are met. Similar behavior can be obtained using the where statement.

The second example analyzes only observations 5 through 20 of the data set `dumb`.

5.15 Data Statement

The data statement is the first line of a SAS data step program. The data statement is used to give a name to a SAS data set. Example:

```
data first;
input name $ age;
...
```

creates a SAS data set with the name of `first`. The name of a SAS data set can be no more that 8 characters long with no embedded blanks.

More than one name can be used in a data statement. The output statement can then be used to construct multiple SAS data sets in a single data step program.

The data step should be terminated by a run statement.

5.16 Do Loops

Do-loops are used to repeat a sequence of operations. Here is an example.

Each row of data in the file iris.dat contains 12 measurements: measurements of four characteristics for each of the 3 species of iris. The 4 measurements are to be read into the 4 variables `m1-m4` and an additional variable `species` will contain a number designating the type of iris from which the measurements were taken, say, 1 for iris setosa, 2 for versicolor, 3 for virginica.

```
data iris;
infile ‘iris.dat’;
do species=1 to 3;
input m1-m4 @;
output;
end;
run;
```

The do loop reads in turn the 4 measurements for each type in a given row of the data set and defines the variable `species` appropriately.

Note that the do loop MUST BE paired with an `end` statement to mark the end of the loop.

The output statement is used to write the variables to the data set being created. If the output statement above were omitted, only the last group of measurments would appear in the SAS data set.

The counter for a do-loop can take values specified by a comma delimited list. Examples:

```
do sex='m','f';
...
end;
do parity=1,3,5;
...
end;
```

A counterless do-loop can be used to have more complicated processing after an if-branch. Example:

```
data old;
input age sex income;
if age<30 then do;
if income<20000 then cat=1;
if income>20000 and income <30000 then cat=2;
end;
if age<= 30 then do;
....
```

Do-loops and if-then statements can be nested.

The continue statement can be used to branch to the bottom of the inner most enclosing do loop.

The return statement can be used to branch to the bottom of the data step program.

The stop command can be used to end datastep processing.

5.17 Display Manager Statement

The display manager governs the appearance of the screen during a SAS session. The display manager is controlled by issuing commands during the session or in the user's autoexec.sas file.

Display manager commands issued during a SAS session are either typed at the command prompt in a window, are issued with a function key to which the command was assigned in the keys window, or are given using a dm statement in the program editor window.

Display manager commands issued in the program editor window or in the autoexec file take the basic form

```
dm window 'command' window;
```

where `'command'` is a display manager command and the window specification is optional. The window commands (which are either program, output, or log) allow switching of the active window either before or after the command is issued.

One of the most used display manager commands is keydef, which is used to assign commands to function keys. Example:

```
dm 'keydef f1 zoom';
```

assigns the zoom command to the function key f1.

For a list of some of the basic display manager commands, see keys. For a full listing of the display manager commands see the SAS Language Reference.

5.18 Drop Statement

Used in the datastep, the drop statement drops variables from the data set. Example:

```
data score;
infile ‘c:\courses\mh160.dat’;
input last $ first $ m1-m3 final;
grade=0.2*m1+0.2*m2+0.2*m3+0.4*final;
drop m1-m3 final;
run;
```

The data set `score` will contain only the 3 variables `last`, `first`, and `grade`.
It is preferable to use the drop= data set option.

5.19 Data Step

SAS programs consist of two parts. The datastep is the part in which raw data is put into a SAS data set for analysis. The procstep (or steps) is the part of the program in which data analysis occurs.

5.20 Filename Statement

SAS filename statements make it easier to refer to files stored on the computer system. The fileref takes the place of the longer system file specification which would otherwise be needed. This is mostly helpful if the same file will be referred to in many places in your program.

The form of the statement is

```
filename fileref ’system-filename’;
```

Here `fileref` is any valid SAS name, and the system-filename is the full path and filename specification of the file in system specific form, e.g., in Windows this would typically be of the form `c:\datadir\mydata.dat`; in UNIX `/datadir/mydata.dat`.

Filename statements can be placed in the autoexec.sas file.

5.21 IF, IF-THEN and IF-THEN-ELSE Statements

Conditional statements (if-then statements, if-then-else statements) are often used to assign values to variables. Example:

```
data new;
input prefix age sex $;
if prefix=821 then city=’auburn’;
if age <30 then young=1;
else young=0;
if age<30 and sex=’f’ then yw=1;
if age>80 or prefix=’749’ then nogood=1;
.....
```

The logical connectives AND and OR are used to create compound tests, as illustrated above.

A counterless do-loop can be used to have more complicated processing after an if-branch. Example:

```
if age<30 then do;
if income<20000 then cat=1;
if income>20000 and income <30000 then cat=2;
end;
....
```

The return statement can be used to branch to the bottom of the data step program.

Do-loops and if-then statements can be nested.

5.22 Infile Statement

The infile statement replaces the cards statement when data is to be read from a file already on disk.

The infile statement MUST appear BEFORE the input statement that reads data from the file. The simple form of the statement is

```
infile 'c:\datafile\mydata.dat';
```

The filename is the full name of a file on the computer system including the path and extension. In a Windows session, this would be of the form given above while on a UNIX platform the form would be `'/datafile/mydata.dat'`.

The associated input statement should process the data in the file as usual.

Instead of specifying the filename directly, a SAS fileref can be used.

Two options are available with the infile statement. Example:

```
infile 'c:\datafile\mydata.dat' missover linesize=100;
```

The missover option is used to assign the missing value symbol to variables not given a value in a data line, and the linesize option changes the length of each line read from the data file from the default value of 80.

The infile statement can be used in place of the cards statement by using the special filename `cards`. The syntax is:

```
infile cards missover linesize=100;
```

This allows the missover and linesize options to be used when the raw data is given in the program editor window.

5.23 Informats

Informats are used to read in data that does not conform to the default SAS length restrictions. Informats are available for character, numeric, and other data forms. Example:

```
input name :$15.  age income;
```

Here the informat of the form `:$w.` (trailing period required) tells SAS to read characters up to the first blank and assign the first `w` of these characters as the value of the variable. Here the first 15 characters will be assigned to the variable `name`. The default informat is equivalent to `:$8.` and `w` must be between 1 and 200.

The informat `&$w.` reads characters up to the first occurance of 2 successive blanks and assigns the first `w` of these characters as the value of the variable. This informat is useful values of character variables have single embedded blanks. The width `w` must be between 1 and 200.

The informat `w.` is used to read in numeric data with `w` digits. This informat is most often used with column input.

The informat `date7.` reads dates of the form `02MAR93`.

The informat `mmddyy8.` reads dates of the form `03-02-93`.

There are many other SAS informats available which are useful for processing dates and other specialized situations. See the SAS Language Reference for additional information.

5.24 Input Statement (List)

The input statement is used in the datastep to read raw data from data cards and assign the data as values of SAS variables.

The most basic type of input statement is list input. Here the variables are simply listed in the order in which they are to be read from the data cards. Example:

```
data second;
input name $ age income;
....
```

The trailing `$` denotes the fact that `name` is to be a character variable, while `age` and `income` are numeric variables.

NOTE: The value of a character variable read using this simple input statement will be truncated to a length of 8 characters and can contain no embedded blanks. This restriction on character variables can be overcome by using a formatting description called a SAS informat.

As SAS reads raw data from a data card an internal SAS pointer moves along the card and points to the next position from which data will be read. For list input, the pointer moves until reaching the next non-space (non-blank) character.

Placing the @ or @@ at the END of an input statement holds the column pointer in position for the correct processing of data cards in which multiple or incomplete records appear on each card.

5.25 Input Statement (Column)

Column input is often used when the raw data has been coded in order to be stored in compact form. SAS is told explicitly in which column the data for each variable begins and ends, so that the proper data is read into each variable. This done by positioning a pointer indicating the column(s) from which the next data value is to be read. Here is part of a data set for illustration.

```
1   333424
2   443324
3   543314
```

Here the first number in each row is the person number and the group of 6 numbers is the rating (on a scale of 1-5) by that person of 6 products. The ratings of the six products are to be analyzed separately. Here's one input statement.

```
input person @5 r1 1. r2 1. r3 1. r4 1. r5 1. r6 1.;
```

The `@5` positions the pointer in column 5. Each variable `r1-r6` is then read from each sucessive column by specifying the informat 1. (period required) as the width of the data. String variables can be handled similarly. The column pointer moves only by the amount specified by the informat.

The previous input statement can be shortened to

```
input person @5 (r1-r5) (1.);
```

This makes use of a shorthand form of column input and also subscripted variables.

Multiple @ statements can be used in a single input statement.

Placing the @ or @@ at the END of an input statement holds the column pointer in position for the correct processing of data cards in which multiple or incomplete records appear on each card.

Further details on using column input can be found in the SAS Language Reference.

5.26 Keep Statement

Used in the datastep, the keep statement explicitly specifies which variables are to be retained in the data set. Example:

```
data score;
infile ‘c:\courses\mh160.dat’;
input last $ first $ m1-m3 final;
grade=0.2*m1+0.2*m2+0.2*m3+0.4*final;
keep last first grade;
run;
```

The data set `score` will contain only the 3 variables `last`, `first`, and `grade`. If the keep statement were not used, the data set would contain 7 variables: `last`, `first`, `m1`, `m2`, `m3`, `final`, and `grade`.

It is preferable to use the keep= data set option.

5.27 Keys Window

The keys window displays the current assignment of function keys to display manager commands.

The keys window can be viewed by issuing the command `keys` at the command prompt.

The commands assigned to function keys can be changed by simply typing the new command over the existing one in the keys window. The new assignment is effective immediately.

Alternately, the DM statement can be used in the program window or the autoexec.sas file.

Some of the typical commands assigned to function keys are:

`output`–makes the ouput window the active window

`page`–toggles whether the output window honors pagebreaks. Can also be explicitly set with the commands `page on` or `page off`. MUST be issued in the output window.

`program` (or `pgm`)–makes the program window the active window

`recall`–recalls text into the program window. Must be issued from the program window.

`submit`–submits the content of the program window for processing. Must be issued from the program window.

`zoom`–toggles the active window between full screen and cascade/tile size. Can also be explicitly set with `zoom on` or `zoom off`.

For further information see the SAS Language Reference.

5.28 Label Statement

Labels of length up to 40 characters (including blanks) can be attached to SAS variables. This partially overcomes the 8 character limitation on SAS names. The most common usage of the label statement is in the data step in which the variable is created. Example:

```
data temp;
input x y type $;
label x='Distance Travelled'
type='Species';
......
```

5.29 Linesize Option

The linesize option is used in either the infile statement or in the options statement. A synonym is ls.

Used in the infile statement, the linesize option sets the length of the line read from a data file. Default is 80. Example:

```
infile 'ozone.dat' missover linesize=100;
```

Used in the options statement, the linesize option sets the width of lines written to the output window. Example:

```
options ls=60;
```

5.30 Libname Statement

The libname statement is used to create a library reference, or libref, for a SAS catalog. The syntax is

```
libname name 'path';
```

where `name` is a SAS name and `path` is a path in the usual format for the underlying operating system. In the Windows environment, a typical use would be

```
libname jerry 'c:\sas\mydata\';
```

5.31 MANOVA

Short form of Multivariate ANalysis Of VAriance.

5.32 Missing Values

Values of variables which are missing or unobtainable can be set to the SAS missing value symbol . (period). Missing values are usually simply disregarded when the data set is analyzed by a SAS procedure. Check the detailed notes on the procedure in the SAS manuals to see exactly how missing values are handled. Missing values often enter a data set through the use of the missover option of the infile statement in the data step.

5.33 Missover Option

The missover option in the infile statement is used to assign the SAS missing value to variables not given a value in a data record. Example:

```
infile 'ozone.dat' missover linesize=100;
```

5.34 _N_

An internal SAS counter in the data step which contains the current iteration number of the data step.

5.35 Names

SAS names, for variables or data sets, are limited to a length of 8 characters with no embedded blanks. Labels can be used to give more descriptive names to variables in the program output.

5.36 Numeric Variables

A numeric variable takes values which are numbers. Character variables takes values which are letters or words.

5.37 Output Statement

The output statement is used in the datastep to write the current values of all variables to a data set. There is an IMPLICIT output statement at the end of each datastep iteration UNLESS an output statement appears somewhere in the datastep. The two datastep programs

```
data first;
input x y z;
cards;
...

data second;
input x y z;
output;
cards;
....
```

will function in an identical manner. If `output` appears by itself on a line, the data is written to the data set specified at the start of the datastep.

```
data junk;
do i=1 to 20;
   input x y z;
   x2=x**2;
   y2=y**2;
   diff=z-x2-y2;
   output;
end;
cards;
.....
```

The output statement here causes the values of the variables `i`, `x`, `y`, `z`, `x2`, `y2`, and `diff` to be written in the data set `junk` for each value of `i`. The data set junk will contain 20 observations. Without the output statement, only the data read when `i=20` would appear in `junk`.

The output statement can be used to create several SAS data sets in a single datastep.

```
data junk(keep=x y z) treasure(keep=diff);
do i=1 to 20;
   input x y z;
   output junk;
   x2=x**2;
   y2=y**2;
   diff=z-x2-y2;
   output treasure;
end;
cards;
.......
run;
```

This program creates 2 datasets. Using the `keep=` data set option causes `Junk` to contain the variables `i`, `x`, `y`, and `z` while `Treasure` contains the single variable `diff`.

5.38 Pageno Statement

The pageno option sets the number of the next page of SAS output. This is often typed in the program editor window to make a clean start after debugging programs. Example:

```
options pageno=1;
```

5.39 Pagesize Option

The pagesize option (synonym `ps`) sets the number of lines per page in the SAS output window. Example:

```
options pagesize=60;
```

This statement should NOT appear inside a data step or a proc step.

5.40 Retain Statement

The retain statement is used to hold the values of variables accross iterations of the data step. Normally, all variables in the data step are set to missing at the start of each iteration of the data step.

The usage `retain x y;` retains the values of the variables `x` and `y` accross data step iterations. The usage `retain;` will retain the values of all variables used in the data step accross iterations of the data step.

As a simple example, look at the following program attempting to compute values of y(n)=2*y(n-1) with y(1)=1.

```
data dumb;
if _n_=1 then y=1;
else y=2*y1;
y1=y;
if _n_ =100 then stop;
run;
```

Since values are set to missing at the start of a data step iteration, the data set dumb will contain one value 1 and the other 99 values of y will be missing.

The following program produces the desired data set.

```
data smart;
retain y1;
if _n_=1 then y=1;
else y=2*y1;
y1=y;
if _n_ =100 then stop;
run;
```

5.41 Return Statement

The return statement branches to the bottom of the data step implicit loop. The implicit output statement of the data step loop will be executed following the return if there is no explicit output statment in the data step program.

5.42 Run Statement

The run statement tells SAS to process the preceding program statements. Example:

```
proc means data=first;
run;
```

Here the run statement starts the processing of the means procedure. The position of the run statement determines when title statements take effect.

The run statement should also be used to mark the end of a datastep.

5.43 Set Statement

The set statement is used to modify an existing SAS data set.

As an example, suppose the data set `course` contains variables `m1-m3` and `final`. A new variable `grade` can be added to the data set as follows:

```
data course;
set course;
grade=(m1+m2+m3)/3+final;
run;
```

A new data set could have been created by changing the first line:

```
data new;
set course;
grade=(m1+m2+m3)/3 +final;
run;
```

Variables can be eliminated from an existing data set in a similar way using drop or keep. Example:

```
data new;
set course;
drop m1-m3;
run;
```

Data records can be deleted from a data set using either the where statement or subsetting if statement. As an example, suppose the data set `rain` contains yearly rainfall by city for the years 1900-1994. The variables in the data set are `year`, `rainfall`, and `city`. Then

```
data aurain;
set rain;
where year >= 1970 and city='auburn';
run;
```

creates a new data set containing auburn rainfall data for the years 1970-1994. This could also be accomplished using:

```
data aurain;
set rain;
if year >= 1970 and city='auburn';
run;
```

Two (or more) data sets can be concatenated (joined) together using set:

```
data new;
set first second;
run;
```

The data set `new` consists of all of the observations in the data sets `first` and `second`. This is useful when data is collected in batches at different points in time.

5.44 Stop Statement

The stop statement ends processing of a data step. Processing continues with the first command following the data step.

5.45 If (Subsetting)

The subsetting if statement is used in conjunction with the set statement to remove data records from a data set.

Data records can be deleted from a data set using either the where statement or subsetting if statement. As an example, suppose the data set `rain` contains yearly rainfall by city for the years 1900-1994. The variables in the data set are `year`, `rainfall`, and `city`. Then

```
data aurain;
set rain;
if year >= 1970 and city='auburn';
run;
```

creates a new data set containing auburn rainfall data for the years 1970-1994.

The where statement provides more efficient processing than the subsetting if statement. However, SAS functions can NOT be used within the where statement, but are allowed in a subsetting if statement.

5.46 Subscripted Variables

Subscripted variables can be used to easily read groups of values in the input statement.

The following example creates and reads values for five variables: `q1`, `q2`, `q3`, `q4`, and `q5`. The data step is written

```
data quiz;
input q1-q5;
cards;
.....
```

5.47 Symbol Statement

Symbol statements (symbol, symbol1,..., symbol255) are used to control some of the behavior of the plot command in proc gplot. The symbol and symbol1 statements are interchangeable. Options specified in a symbol statement apply in all higher numbered symbol statements unless explicitly set to other values.

Symbol statements are used (starting with symbol1) until use of that symbol statement would result in duplication of the color, plotting symbol, or interpolation methods that have already appeared in the plot, or until the defined symbol statements are exhausted.

There are many options that can be set in the symbol statement. In black on white printing (as set up using the colors statement in the goptions statement) the most commonly used are line style, line width, and interpolation method. Example:

```
symbol l=3 w=5 i=join;
```

The option `l=n`, where `1<=n<=46`, specifies the style of line to be used in the plot. `l=1` corresponds to a solid line, and the remaining choices correspond to a variety of dashed lines.

The option `w=n` specifies an integer magnification factor for the thickness of the lines within the plot. This option is not supported on all devices.

The `interpol=` option (which can be abbreviated `i=`) sets the interpolation method (the way in which points on the graph are connected). Some common options are: `i=none` (points are not connected), `i=join` (points are connected by straight lines), i=needle (produces a needle graph), i=stepl (produces empirical distribution function graphs), i=stdm (produces a plot with error bars). There are further options available to control the features of the error bars.

For further information consult the SAS/GRAPH Users Guide.

5.48 Title Statement

Up to 10 title lines can be placed on printouts generated by SAS by using title statements. Each title line will be used until it is replaced or erased. Examples:

```
title 'My printout'; /* same as: title1 'My printout'; */
title2 'second line of title';
title1 'a new first line';
title3 ;  /* erases title lines 3-10 */
```

Title statements take effect at the first following run statement.

A title line containing a quote can be produced by using two consecutive single quotes. Example: `title 'Mary''s Printout';`.

5.49 Updating Data Sets

To correct errors in large data sets the update statement can be used:

```
data new;
update old changes;
by indexvar;
run;
```

Here changes is a data set containing the corrections to be made to the data set old. Records in old and changes with the same value of the variable indexvar are updated using the information in changes. Missing values in the data set changes do not overwrite data in old.

To concatenate (join) two (or more) data sets together the set statement can be used.

5.50 Where Statement

The where statement can be used with any procedure to restrict the action of that procedure to observations satisfying the conditions of the where statement. Example:

```
proc means data=dumb;
where x<5 and y>3;
run;
```

Here proc means analyzes only the observations in the data set dumb for which x<5 and y>3.

Similar behavior can be obtained by using the where = data set option.

The where statement can be used in conjunction with the set statement to select observations from an existing data set. Here's an example.

Suppose the data set iris contains variables species and m1-m4. Then

```
data onetwo;
set iris;
where species=1 or species=2;
run;
```

creates a new data set `onetwo` containing the data for species 1 and species 2 only. The logical connectives `and` and `or` can be used to create more complicated selection schemes.

The where statement provides more efficient processing than the subsetting if statement. SAS functions can NOT be used within the where statement, but are allowed in a subsetting if statement.

5.51 X Command

The X command can be used to execute operating system commands without leaving SAS. The form is

```
X 'operating system command'
```

and should be typed at the SAS command prompt.

Operating system commands can also be executed by selecting GLOBAL–COMMAND–HOST COMMAND from the menu bar in a window.

6 Procedures Reference

Here is a list of the SAS procedures discussed in this help document.

- Proc Access: dBase and DIF Files
- Proc ANOVA
- Proc Arima
- Proc Contents
- Proc Discrim
- Proc Freq
- Proc G3d
- Proc GLM (basics)
- Proc GLM (advanced)
- Proc GLM for MANOVA
- Proc Gplot
- Proc IML (Basics)

- Proc Means
- Proc Npar1way
- Proc Plot
- Proc Print
- Proc Rank
- Proc Reg
- Proc Sort
- Proc Spectra
- Proc Ttest
- Proc Univariate

6.1 Proc Access: dBase and DIF Files

Files created by some other programs (dBase or Lotus 1-2-3, for example) can be used with SAS with some effort. SAS data sets can also be put into a form usable by these programs.

This discussion applies to SAS 6.08 release TS 407, or later, for the Microsoft Windows environment.

SAS can read a dBase or DIF file directly, or a dBase or DIF file can be turned into a SAS data set.

First, look at how SAS can read a dBase file directly. These examples assume that `mylib` is a valid SAS libref. Suppose the dBase file is `c:\mydata.dbf`. A SAS "view descriptor" called `mydata` is created as follows.

```
proc access dbms=dbf;
create mylib.mydata.access;
path='c:\mydata.dbf';
create mylib.mydata.view;
select all;
run;
```

From this point on, the view descriptor `mydata` can be used as though it were the name of a SAS data set in the catalog `mylib`. It is not possible to add new variables to this data set, however.

To put the data in a dBase file into a SAS data set, the procedure is slightly longer.

```
proc access dbms=dbf;
create mylib.mydata.access;
path='c:\mydata.dbf';
create mylib.mydata.view;
select all;
```

```
proc access viewdesc=mylib.mydata out=mylib.mydata;
run;
```

This has created a SAS data set called `mydata` in the catalog `mylib` which contains the variables in the dBase file. There is no link between the dBase file and the SAS data set.

For DIF files, replace `dbms=dbf` with `dbms=dif` above. Filenames will most likely have a .dif extension too.

To turn a SAS data set into a dBase (version 3) file, suppose the SAS data set is `mydata` in the catalog `mylib`. Then

```
proc dbload dbms=dbf data=mylib.mydata;
path='c:\mydata.dbf';
label;
limit=0;
load;
version=3;
run;
```

creates the dBase file `c:\mydata.dbf`. To make a DIF file

```
proc dbload dbms=dif data=mylib.mydata;
path='c:\mydata.dif';
label;
limit=0;
load;
run;
```

does the job.

For more information, see the online help under SAS SYSTEM HELP–DATA MANAGEMENT–SAS/ACCESS or the SAS/ACCESS Manual.

6.2 Proc ANOVA

Proc anova is used to perform an analysis of variance when the data was collected using a balanced design. Proc anova uses less computational resources than proc glm and is recommended only for large projects in which experts have insured that the design is balanced. Proc GLM is safer in the hands of beginners.

6.3 Proc Arima

Analysis of time series data in the time domain is done with this procedure. Box-Jenkins methodology (the fitting of ARIMA models to time series data) and also transfer function (input type) models can be used. Frequency domain analysis of time series can be done using Proc Spectra.

The framework for the analysis is that the observed time series X(t) is stationary and satisfies an ARMA equation of the form

```
X(t) -phi(1) X(t-1) - ... -phi(p) X(t-p)=
Z(t) -theta(1)Z(t-1)-...-theta(q) Z(t-q)
```

where `Z(t)` is a white noise process. The constants `phi(1)`,..., `phi(p)` are called the autoregressive coefficients and the number `p` is called the order of the autoregressive component. The constants `theta(1)`,..., `theta(q)` are called the moving average coefficients and the number `q` is called the order of the moving average component. It is possible for either `p` or `q` to be zero.

Use of proc arima to fit ARMA models consists of 3 steps. The first step is model identification, in which the observed series is transformed to be stationary. The only transformation available within proc arima is differencing. The second step is model estimation, in which the orders p and q are selected and the corresponding parameters are estimated. The third step is forecasting, in which the estimated model is used to forecast future values of the observable time series.

As an example, the data file milk.dat containing data on milk production taken from Cryer will be analyzed. Here are the commands that could be used for each of the 3 steps.

```
proc arima data=milk;
identify var=milk(12)
nlag=30
center
outcov=milkcov
noprint;
run;
estimate p=1 q=3
nodf
noconstant
method=ml
plot;
run;
forecast
lead=10
out=predict
printall;
run;
```

OPTIONS FOR THE IDENTIFY STATEMENT:

The `var=` statement is required and specifies the variable(s) in the data set to be analyzed. The optional numbers in parenthesis specify the LAG at which differences are to be computed. A statement `var=milk` would analyze the milk series without any differencing; `var=milk(1)` would analyze the first difference of milk; `var=milk(1,1)` the second difference of milk.

The `var=` statement produces 3 plots for the specified variable: the sample autocorrelation function, the sample inverse autocorrelation function, and the sample partial autocorrelation function. These crude plots and tables of their values are printed in the output window. Higher quality plots can be produced through the use of other options (detailed below) and proc gplot.

The nlag= option causes the 3 plots to print values up to lag 30. If not specified, the default is nlag=24 or 25% of the number of observations, whichever is less.

The center option subtracts the average of the series specified by the var= statement. The average is added back in automatically during the forecast step.

The outcov= option places the values of the sample correlation functions into a SAS data set. These values can be used to produce high quality plots of these functions using proc gplot. The variables output are: LAG, VAR (name of the varible specified in the var= option), CROSSVAR (name of the variable specified in the crosscorr= option), N (number of observations used to compute the current value of the covariance or crosscovariance), COV (value of the cross covariances), CORR (value of the sample autocorrelation function), STDERR (standard error of the autocorrelations), INVCORR (values of the sample inverse autocorrelation function), and PARTCORR (values of the sample partial autocorrelation function).

The noprint option suppresses the output of the low quality graphs normally created by the var= statement. This option is used primarily with the outcov= option.

OPTIONS FOR THE ESTIMATE STATEMENT:

The p=1 q=3 options specify the auto-regressive and moving average orders to be fit. Other forms of these specifications are: q=(3) to specify that ONLY the parameter theta(3) is allowed to be non-zero; p=(12)(3) for a seasonal model (1-phi(12)B**12)(1-phi(3)B**3) where B is the backshift operator; p=(3,12) for a model in which only phi(3) and phi(12) are allowed to be non-zero.

The nodf option uses the sample size rather than the degrees of freedom as the divisor when estimating the white noise variance.

The method option selects the estimation method for the parameters. The choices are ml for maximum (Gaussian) likelihood estimation, uls for unconditional least squares, and cls for conditional least squares.

The plot option produces the same 3 plots as in the identify statement for the RESIDUALS after the model parameters are estimated. This is another useful check on whiteness of the residuals.

OPTIONS FOR THE FORECAST STATEMENT:

The lead option specifies the number of time intervals into the future for which forecasts are to be made.

By using the out= and printall options in the forecast statement, a SAS dataset will be created which will contain the values of the original series and the predicted values of the series using the model at all times. This can be useful for an analysis of the past performance of the model.

In practice, several different estimate statements are tried sequentially to see which model best fits the data. Proc arima is interactive, in the sense these sequential attempts can be made without restarting the procedure. Simply submit the successive estimate statements; the original identify statement will be retained.

Transfer function models can be fit by using the crosscorr option of the identify statement and the input option of the estimate statement. The mechanics of this procedure are illustrated for a dataset fake which contains two time series which are related by a transfer function model. In this case, Y depends on X. First, the process

`X` is modeled using the identify and estimate statements. Then `Y` is identified and the cross-correlation between the prewhitened processes `X` and `Y` is estimated. The program might look like this.

```
proc arima data=fake;
identify var=x
center
nlag=40;
estimate p=1 q=1
noconstant
nodf
method=ml
plot;
identify var=y
center
nlag=40
crosscorr=(x);
run;
```

From the cross correlation information, the lags at which the input process `X` influences `Y` can be tentatively identified. Note that only causal models are allowed; non-zero cross correlations at negative lags cannot be modeled in proc arima. For illustration, say the non-zero lags are 2 and 4. The process `Y` might be estimated as follows.

```
estimate input=( 2$(2) x )
p=1 q=3
noconstant
nodf
method=ml
plot;
run;
```

The input is of the form `cB**2+ dB**4= B**2( c + dB**2)`. It is this latter form that gives the form of the input statement.

Note that the estimate statement always refers to the most recent identify statement to decide what variable(s) are to be included in the model. Thus differencing and centering are handled automatically (if used) EXCEPT that differencing must be explicitly specified in the crosscorr statement.

For more details see the online help under SAS SYSTEM HELP–MODELING & ANALYSIS TOOLS–ECONOMETRICS & TIME SERIES–ARIMA or the SAS/ETS Guide.

6.4 Proc Contents

Proc contents lists the structure of the specified SAS data set. The information includes the names and types (numeric or character) of the variables in the data set. The most common form of usage is

```
proc contents data=second;
run;
```

This lists the information for the data set `second`. The invocation

```
proc contents data=_all_;
run;
```

lists the contents of all data sets in the current catalog.

For further information see the online help under SAS SYSTEM HELP–DATA MANAGEMENT–CONTENTS or the SAS Procedures Guide.

6.5 Proc Discrim

Proc discrim is used to conduct discriminant analysis.

The purpose of discriminant analysis is to classify an experimental unit as being from one of two (or more) populations on the basis of observations obtained on that unit. For example, a bank might attempt to classify a new loan applicant as a 'good payer' or 'dead beat' on the basis of the answers given on a loan application.

The common syntax is:

```
proc discrim
data=bank
method=normal
pool=yes
slpool=0.001
posterr
out=results;
class type;
var hist balance income;
run;
```

The `data=` line specifies the data set used in the analysis.

The `method=normal` option selects discrimination based on the assumption that the data is from multivaiate normal populations.

The `pool` option can be set to either `yes`, `no`, or `test`. This controls whether the covariance matrices are assumed equal in the analysis (`yes`), assumed to be UNequal (`no`), or tested for equality (`test`) with subsequent analysis performed according to the outcome of this test.

The `slpool=` statement sets the significance level of the test for equality of covariance matrices when `pool=test` is used. It is otherwise disregarded.

The `posterr` option prints out estimated error probabilities for the computed discrimination rule.

The `out=` option creates a new data set which contains the variables in the original one together with a new variable called `_into_` of the same type as the `class` variable. The `_into_` variable gives the class into which the observation is assigned by the discrimination rule.

The `class` statement, which MUST BE USED, specifies the variable for which classification is to occur.

The `var` statement specifies the variables to be used for discrimination. If omitted, all variables in the data set (except the one specified in the `class` statement) are used. It is best to specify the variables explicitly to avoid the unintended use of extraneous variables in the analysis.

Sometimes one set of data is used to construct the discrimination rule and a second set is used to test the rule. This can be accomplished as follows.

```
proc discrim
data=construc
testdata=tryout
method=normal
pool=yes
slpool=0.001
posterr;
class type;
var hist balance income;
run;
```

The `testdata=` option specifies a SAS data set which is used to tryout the discrimination rule found by analyzing the `data=` data set. It is assumed that the `testdata=` data set contains the same independent variables and class variable as the `data=` data set. Diagnostics of the discrimination rule on the `testdata=` data set are printed.

To use one data set to construct a discrimination rule using `method=normal` for later use, use the `outstat` option:

```
proc discrim
data=first
outstat=calib
method=normal
.....
```

The special data set `calib` then contains the discrimination rule constructed from the data set `first`. To apply this rule to the data set `second`:

```
proc discrim
data=calib
testdata=second
method=normal
.....
```

For further information see the SAS/STAT User's Guide, volume 1.

6.6 Proc Freq

Proc freq is used to perform one, two, and n way analysis of catagorical data. The typical invocation and options are

```
proc freq
data=info
page;
by x;
tables y*z
/chisq
expected
exact
sparse
nocum
nopercent
nocol
norow;
run;
```

The backslash MUST be used (once) to select any of the options of the `tables` statement.

The `data=` line is required and specifies the data set to be analyzed.

The `page` option specifies that only one table should appear on each page. Without this specification, multiple tables will be printed on each page as space permits.

The optional by statement produces separate analyses for each value of the `by` variable(s).

The `tables` statement produces a table of values for analysis. The form `y*z` produces a 2 way table with the values of `y` defining table rows and the values of `z` defining table columns. A 1 way table is obtained by specifying a single variable name. Multiple tables may be requested within a single tables statement. The short form (q1--q3)*a requests 3 tables and is equivalent to `q1*a q2*a q3*a`. Note the double hyphen required here.

Options for the tables statement follow the /.

The `chisq` option to the tables statement performs the standard Pearson chi-square test on the table(s) requested.

The `expected` option prints the expected number of observations in each cell under the null hypothesis.

The `exact` option requests Fisher's exact test for the table(s). This is automatically computed for 2 x 2 tables.

The `sparse` option produces a full table, even if the table has many cells containing no observations.

The `nocum` option suppresses printing of cumulative frequencies and percentages in the table.

The `nopercent` option suppresses printing of the cell percentages and cumulative percentages.

The `nocol` and `norow` options suppress the printing of column and row percentages.

For further information see the SAS/STAT User's Guide volume 1.

6.7 Proc GLM (basics)

Proc GLM is the preferred procedure for doing univariate analysis of variance (ANOVA), multivariate analysis of variance (MANOVA), and most types of regression.

Proc anova could also be used to do the analysis of variance when the design is balanced. The only advantage in using proc anova is that it uses less computational resources. Its use is recommended only for large projects in which experts have insured that the design is balanced.

The file pig.dat contains data on the birth weight of poland china pigs in 8 litters (from Scheffe). The layout of the file is the litter number followed by the birth weights for the litter. Note that there are unequal litter sizes. The underlying model is that each litter has some mean birth weight. The hypothesis to be tested is that these means are the same for all of the litters.

A data set `pig` is created which contains 2 variables `litter` and `weight`. The variable `litter` contains the litter number; each corresponding value of `weight` contains the birth weight of a piglet. The model is that `weight` depends on `litter`. The question of interest is whether this dependence really exists. Note that `litter` is a qualitative variable here which serves only to distinguish between (potentially) different populations.

A basic analysis using proc glm proceeds as follows.

```
proc glm
data=pig;
class litter;
model weight = litter;
run;
```

The `class` statement specifies that `litter` is a QUALITATIVE variable. All qualitative independent variables in the model should be listed in the class statement. Independent variables which appear in the model but not in the class statement are treated as QUANTITATIVE variables in the model. This allows proc glm to perform regression and analyze analysis of covariance type models.

The `model` statement specifies the dependent variable (`weight`) and independent variable (`litter`).

The output of this procedure is a standard analysis of variance table. Two types of sums of squares are produced. In virtually all cases, it is the TYPE III sums of squares and their associated tests that are of interest.

A more in depth analysis is obtained as follows.

```
proc glm
data=pig;
class litter;
model weight=litter /solution e;
means litter /bon tukey scheffe alpha=0.10 ;
run;
```

The `solution` option of the model statement prints a solution to the normal equations, i.e., the estimate of the parameter vector in the general linear model.

The `e` option of the model statement produces a print out which specifies the general form of the estimable functions for the model. This is useful for more advanced analysis using contrast and estimate statements.

The `means` statement produces pairwise comparisons of the means by the method(s) specified in the following options statements. Note that comparisons are only done for the main effects.

The `bon` option of the means statement produces Bonferroni type groupings of the means.

The `tukey` option produces groupings of the means using Tukey's method of multiple comparison.

The `scheffe` option produces groupings of the means using Scheffe's method of multiple comparison.

Actual confidence intervals (rather than just groupings by similar means) produced by these methods can be obtained by specifying the `clm` option in the means statement.

The `alpha=0.10` option specifies that the multiple comparison methods are to be done at the 90% confidence level. If not specified, the value defaults to alpha=0.05 (95% level).

Further options are described in the online help under SAS SYSTEM HELP–MODELING & ANALYSIS TOOLS–DATA ANALYSIS–(ANALYSIS OF VARIANCE) GLM or in the SAS/STAT User's Guide volume 2.

6.8 Proc GLM (advanced)

Here some further analysis using proc glm is outlined. This goes beyond the one way ANOVA discussed in Proc GLM.

The following data is from Hicks and is found in the file tomato.dat. Tomato plants were grown in a greenhouse under treatments consisting of combinations of soil type and fertilizer type. A completely randomized two factor design was used with two replications per cell. The following data on yield in kilograms of tomatoes were obtained for the 30 plants under study.

```
               Fertilizer Type
Soil Type 1 2 3
I 5,7 5,5 3,5
II 5,9 1,3 2,2
III 6,8 4,8 2,4
IV 7,11 7,9 3,7
V 6,9 4,6 3,5
```

The analysis here is as a two factor analysis of variance. Interaction may be assumed, but first the two factor additive model is used.

There are 2 qualitative factors: soil and fertilizer. The data is put into a SAS data set `toms` with 3 variables: `soil`, `fert`, and `yield`. The analysis proceeds as follows.

```
proc glm
data =toms;
```

```
class soil fert;
model yield=soil fert;
means soil fert /bon;
run;
```

This provides the ususal analysis of variance table and a grouping of similar means due to each qualitative factor using the Bonferroni method of multiple comparison with alpha=0.05 (the default level). Under the assumed additive model, this information is sufficient for a complete analysis of the data.

If the additive model with interaction is to be used, the model statement must be modified. There are two equivalent ways of writing the statement. One is

```
model yield= soil fert soil*fert;
```

This introduces the soil-fertilizer interaction term. The other short hand way of obtaining this model is

```
model yield= soil | fert;
```

The vertical bar creates main and crossed effects for the variables in question. This method generalizes to more than two variables. The notation

```
y=a | b | c
```

would generate a model in which the main, two way, and three way interactions of the factors a, b, and c were present in the model, i.e., the 3 factor full model.

Nested models are specified using parenthesis. A model with effects A and B with B nested within A is specified with a model statement

```
model y=b(a);
```

If the initial F tests show that interaction is indeed present, a means statement of the form

```
means soil*fert /bon;
```

will NOT produce Bonferroni groupings since the bon (and other) options in the means statement apply only to main effects. The p-values for pairwise t-tests comparing the treatments in the interaction case are obtained with

```
lsmeans soil*fert /pdiff;
```

This will provide all pairwise comparisons and the corresponding p-values for test of no difference. By suitably increasing the level of significance the Bonferroni ranking can be obtained.

Especially in the case of the interaction model it may be desirable to do customized tests of hypothesis. These can be performed by using the contrast statement. As an example, suppose it is desired to test the hypothesis that there is no difference in yield between soils IV fertilizer 1 and soil IV fertilizer 2. This hypothesis can be written as a test that a particular contrast of the model parameters is zero. (To be sure that the contrast specified is desired one, it is a good idea to use the solution option of the model statement so that the order of the parameters is known.) Recall that the General Linear Model can be written in the form Y=XB+E, where Y is the n x 1 vector of observations, X is an n x k matrix of known constants, and B is a k x 1 vector of unknown parameters. The form for the hypothesis test is assumed to be LB=0 where the matrix L is known. In this case the form for the complete analysis would be

```
proc glm
data=toms;
class soil fert;
model yield=soil | fert;
contrast 'IV-1 vs. IV-2'
fert 1 -1 0
soil*fert 0 0 0  0 0 0  0 0 0  1 -1 0;
```

Note that trailing zeroes need not be specified and that effects entering the test with zero coefficients can be omitted.

Multiple contrast statements can be specified.

Using an estimate statement rather than a contrast statement produces an estimate of LB. The syntax is otherwise the same as for the contrast statement.

A final important option in proc glm is the output statement. This statement, which must follow the model statement, creates a SAS data set containing the variables in the original data set together with new variables as specified in the output statement. An illustration of some of the common options is

```
output
out=results
predicted=pred
residual=resid
L95M=lowmean
U95M=highmean
L95=lowpred
U95=highpred;
```

The `out=` option gives the name of the new SAS dataset.

The `predicted=` option gives the name of the variable in the `out=` data set which contains the predicted value of the dependent variable. By adding records to the original data set which specify values of the INDEPENDENT variables in the model but set the corresponding value of the DEPENDENT variable to missing, one can obtain predictions given by the model for unobserved settings of the independent variables. This is particularly useful in regression analysis.

The `residual=` option gives the name of the variable in the `out=` data set which contains the value of the residual.

The `L95M=` and `U95M=` options give the names of the variables in the `out=` data set which contain the lower and upper endpoints of a 95% confidence interval for the mean.

The L95= and U95= options give the names of the variables in the out= data set which contain the lower and upper endpoints of a 95% confidence interval for the predicted value.

Repeated measures analysis can be done by using the repeated statement. See Proc GLM for MANOVA for details.

6.9 Proc G3d

Proc g3d is used to make 3 dimensional plots. The typical invocation is

```
proc g3d data=stuff;
plot light*water=yield / grid;
run;
```

This plots yield values on the z axis as a function of light and water on the x and y axis. The grid option makes a grid in the x-y plane.

Additional options which can be specified after the / of the plot command are noaxes to suppress axes and axis labels, xticknum=n to specify the number of x axis ticks which are used, yticknum=n to specify the number of y axis ticks which are used, zticknum=n to specify the number of z axis ticks which are used, zmax=value to specify the maximum z value to include in the plot, and zmin=value to specify the minimum z value to include in the plot. There are numerous other options.

The scatter command can be used instead of plot to produce a scatter plot of the data. Example:

```
proc g3d data=stuff;
scatter light*water=yield;
run;
```

Numerous options are available.

For additional information see the SAS/GRAPH User's Guide.

6.10 Proc Gplot

Proc gplot is used to make presentation quality graphs. Crude graphs can be made using proc plot.

Options controlling the hardware available for SAS to use for the high quality graphics procedures are specified in the goptions statement. Read about the goptions statement before continuing.

To illustrate the basics, suppose the dataset quad contains the values of y=x**2 for x=1,...,10. A nice plot of this quadratic is desired. The basic plot is obtained as follows.

```
proc gplot
data=quad;
plot y*x;
run;
```

The result of these commands will depend on the interpol setting in the goptions statement. If `interpol=join` is set, the 10 data points will be connected by straight lines.

By default, the axes are scaled automatically. The invocation

```
proc gplot
data=quad;
plot y*x/ haxis=0,5,10 vaxis=0 to 120 by 20;
run;
```

gives tick marks at 0,5, and 10 on the horizontal axis, and marks the vertical axis at intervals of 20 from 0 to 120. This illustrates the two methods of manual scaling.

For a more elaborate example with the same data:

```
proc gplot
data=quad;
title ‘My First Plot’;
title2 ‘A Simple Quadratic’;
plot y*x /vref=4 href=16;
symbol interpol=join l=3 w=5;
run;
```

The `vref=` option draws a horizontal line at `y=4`; the `href=` option draws a vertical line at `x=16`.

The symbol statement specifies the interpolation method (here join), the line style (`l=3`) specifies the type of line to draw, and the width option (`w=5`) makes the line 5 times its normal thickness. (The `w` option is not supported on all devices.) SYMBOL STATEMENTS ARE GLOBAL. This symbol statement will be used by all subsequent graphs.

Multiple plots can be made in 3 ways.

```
proc gplot
data=junk;
plot y*x z*x /overlay;
run;
```

plots `y` versus `x` and `z` versus `x` using the same horizontal and vertical axes.

```
proc gplot
data=junk;
plot y*x;
plot2 z*x;
run;
```

plots `y` versus `x` and `z` versus `x` using different vertical axes. The second vertical axes appears on the right hand side of the graph.

```
proc gplot
data=junk;
plot y*x=z
run;
```

uses `z` as a classification variable and will produce a single graph plotting `y` against `x` for each value of the variable `z`.

```
proc gplot
data=junk;
plot y*x;
by z;
run;
```

produces separate graphs of `y` against `x` for each value of `z`.
For additional information see the SAS/GRAPH User's Guide.

6.11 Proc IML (Basics)

The Interactive Matrix Language (IML) provides a complete environment for performing matrix manipulations. This can be used to manipulate SAS data sets and perform a complete statistical analysis. This section introduces the basic features. Accessing SAS datasets in IML and the IML macro programming language, and IML graphics programming are discussed elsewhere.

Start IML by issuing the command `proc iml;` on a line in the program window. Don't forget to use LOCALS–SUBMIT to actually have the command processed. Leave IML by issuing the command `quit;`.

NOTE: Use of a data step or another procedure will terminate the IML session.

Native data elements in IML are matrices. The command

```
a={1 2, 3 4, 5 6};
```

creates a `3x2` matrix `a` with the indicated entries. The entries may be spread accross several lines, so it may be easier to input large matrices as

```
a={
1 2,
3 4,
5 6};
```

Note that spaces separate entries in each row, and commas are used to separate rows.

The entries of a matrix can be all numbers or all character strings. The two types cannot be mixed in the same matrix.

Some of the typical IML commands are listed here.

- Basic commands
- Indexing commands

- Matrix reduction commands
- Horizontal and vertical concatenation commands
- Identity matrix
- Constant matrix
- Design matrices
- Repetition factors
- Special matrix functions

The print statement is used to print the indicated matrices in the output window. Typical usage is

```
print a b[rowname=name format=4.1];
```

which will print the matrix `a` in the default format, and the matrix `b` formatted with 4 significant digits one of which follows the decimal. The matrix `b` will also have the vector name used as names for the rows.

The reset statement specifies certain options for printing and numeric calculation.

Labels, row names, column names, and numeric formats can be associated with a matrix through the mattrib statement.

The full list of SAS functions is available online under HELP–SAS SYSTEM–SAS LANGUAGE–SAS FUNCTIONS. A partial listing can be found here.

As usual, comments can be enclosed between /* and */.

Matrices can be stored on disk and retrieved for later use by using the store and load commands.

Further details can be found in the online help under HELP–SAS SYSTEM–MODELING AND ANALYSIS TOOLS–INTERACTIVE MATRIX LANGUAGE or Chapters 2 and 4 of SAS/IML Software.

6.11.1 Accessing SAS Data Sets in IML

It is easy to either read a SAS data set into a matrix or turn a matrix into a SAS data set. Keep in mind that matrices must contain either numeric or character variables but not both.

All the numeric variables in the SAS data set `class` are read into a matrix `m` with the commands

```
use class;
read all var _num_ into m;
close class;
```

The `use` statement opens a SAS data set for read access. The data set can have either a one or two level name. More than one data set can be open at the same time. Data sets should be closed when not immediately needed.

The `read` statement reads the values from the data set. The `var_num_` option specifies that all numeric variables are used as columns of the matrix `m`. (This is the default selection and may be omitted from the command.) Other options are `var _char_` for all character variables, `var _all_` for all variables, and `var {height age}` to read only the variables `height` and `age` into the matrix.

The `close` statement closes the SAS dataset.

The invocation

```
use class;
read all;
close class;
```

creates a column vector for each variable in the SAS data set. The name of each vector is the name of the corresponding variable in the data set.

The commands

```
create class from m [colname={'height' 'age'}];
append from m;
close class;
```

creates a SAS data set `class` from the matrix `m` and assigns `height` and `age` as the names of the variables associated with the columns of `m`.

For further information see the online help under HELP–SAS SYSTEM–MODELING AND ANALYSIS TOOLS–INTERACTIVE MATRIX LANGUAGE or Chapter 6 of SAS/IML Software.

6.11.2 IML Macro Programming Language

IML has all the capabilities of a programming language. User defined functions and call routines (referred to as modules) can be used to extend the built in capabilities.

User defined functions consist of programming statements enclosed between start and finish statements. As an example, the function add which computes the sum of its arguments is defined by

```
start add(x,y);
sum=x+y;
return(sum);
finish add;
```

The parameters in a module definition are dummy variables.

Variables used within a module are local variables.

The value to be returned by the function module is given in a `return` statement.

Modules and matrices can be stored on disk and reloaded for use in later IML sessions using store and load commands.

The usual programming constructs are available for use both inside and outside of modules. In particular, if-then and if-then-else constructs are available, as are do-end loops. The use of these constructs is as in the SAS data step. GOTO statements can be used too. The label for the GOTO statement consists of a name followed by a colon. Example:

```
start absolute(x);
if x>0 then goto pos;
y=-x;
return (y);
pos:
return(x);
finish absolute;
```

is an incarnation of the absolute value function.

Further information can be found in the online help under HELP–SAS SYSTEM–MODELING AND ANALYSIS TOOLS–INTERACTIVE MATRIX LANGUAGE or Chapter 5 of SAS/IML Software.

6.11.3 IML Graphics Programming

Proc IML has its own graphics commmands which can be used to produce high quality graphics output, similar to that produced by proc gplot. Some of the basic commands are illustrated here.

There are two coordinate systems used in IML graphics. These coordinate systems are used to relate the range of data values to physical positions on the output device.

The **world coordinate system** is the coordinate system determined by the range of values of the data. In the world coordinate system, the range of values along each axis are arbitrary. A **window** is a rectangular area defined in the world coordinate system.

The **normalized coordinate system** is the coordinate system used to logically describe positions on the output device, usually the screen or page. The lower left corner of the output device ALWAYS has coordinate (0,0) in the normalized system; the upper right corner ALWAYS has coordinate (100,100) in the normalized system. A **viewport** is a rectangular area in the normalized system.

By defining multiple viewports and windows and linking them, it is possible to put multiple graphs on a single page of output.

Here is a short program that defines three vectors x and y and z. The graph of y against x and z against x are placed on the same page.

```
proc iml;
/* Define the data */
x={1,2,3,4,5};
y={1,4,9,16,25};
z={1,-4,9,-16,25};

reset clip;  /* clip data at edge of viewport*/

/* Start making the graphs */
call gstart;  /* MUST be used to start IML graphics session */

call gopen('xyz-plot'); /* Open new graphics segment */
call gset('font','swiss'); /* font and height for text */
call gset('height',1.0);
call gwindow({-5 -5 7 30}); /* declare window for y vs. x graph */
/* slightly larger than data range */
/* to leave room for text */
```

```
call gport({0 50 100 100}); /* declare port for y vs. x graph */
/* port is upper half of screen */
call gpoint(x,y,'plus'); /* plot the data points marking */
/* each with a plus sign */
call gdraw(x, y,1); /* connect the dots using linestyle 1 */

call gxaxis({0 0},5,6, , ,'2.',.5); /* draw x and y axis and label them */
call gyaxis({0 0},25,6, , ,'2.',.5);

call gtext(3,-3,'X Values'); /* Horizontal text labelling x axis */
call gvtext(-1,20,'Y Values'); /* Vertical text labelling y axis */
call gtext(2,27,'Graph of Y versus X'); /* Title text */

/*Now make z versus x plot */
call gwindow({-5 -30 7 30}); /* declare window for z vs. x graph */
/* slightly larger than data range */
/* to leave room for text */
call gport({0 0 100 50}); /* declare port for z vs. x graph */
/* port is lower half of screen */
call gpoint(x,z); /* plot the data points marking */
/* each with a plus */
call gdraw(x,z,2); /* connect the dots using linesytle 2 */

call gxaxis({0 0},5,6, , ,'2.',.5); /* draw x and y axis and label them */
call gyaxis({0 -25},50,11, , ,'3.',.5);

call gtext(3,-25,'X Values'); /* Horizontal text labelling x axis */
call gvtext(-1,20,'Z Values'); /* Vertical text labelling y axis */
call gtext(2,27,'Graph of Z versus X'); /* Title text */

call gshow;  /* MUST be used to display graphs */
call gclose; /* close graphics segment */
```

Most of the commands are self-explanatory. Here are some details. For further information see the SAS/IML Software manual or the online help under SAS SYSTEM–MODELLING AND ANALYSIS TOOLS–INTERACTIVE MATRIX LANGUAGE.

`gpoint(x,y,'plus')` sequentially plots the pairs of points determined by the corresponding entries of the vectors `x` and `y`. The vectors `x` and `y` must have the same length. The optional plot symbol (here `'plus'`) can be specified.

`gdraw(x,y,1)` sequentially connects the pairs of points determined by the corresponding entries of the vectors `x` and `y`. The vectors `x` and `y` must have the same length. The optional line style (here the default value `1`) selects a line style.

`gxaxis({0 0},5,6, , ,'2.0',.5)` draws the x axis beginning at the point `{0 0}`, of length `5`, with `6` tick marks. Each label is in the format `2.0` and of height `.5` units. Space must be left for the 2 optional and omitted arguments.

The first two arguments of the text commands `gtext` and `gvtext` specify the lower left corner of the position of the first letter of text in world coordinates. The text height and style are determined by the `gset` commands given earlier.

6.11.4 IML Indexing Commands

- `a=do(3,18,5)` yields `a={3 8 13 18}`

- `a=2:5` yields `a={2 3 4 5}`

6.11.5 IML Basic Commands

- + addition (numeric)
- + concatenation (strings)
- - subtraction
- * matrix multiplication
- @ Kronecker (direct) product
- ** matrix power
- # elementwise multiplication
- ## elementwise power
- / elementwise division
- a‘ transpose
- T(a) transpose
- inv(a) matrix inverse
- det(a) determinant
- | logical OR
- & logical AND
- ˆ= logical not equal to
- a[2,3] entry in row 2 column 3 of a

6.11.6 IML Matrix Reduction Commands

`d=c[3,]` sets `d` equal to the third row of `c`.

`d=c[,5]` makes `d` equal to the fifth column of `c`.

`d=c[{1 3},{2 3 4}]` makes `d` a 2x3 matrix containing the entries in rows 1 and 3 and columns 2,3 and 4 of `c`.

`d=c[ ,+]` creates a column vector `d` whose entries are the sums of the row entries in `c`.

`e=c[+,]` makes `e` a row vector whose entries are the sums of the columns of `c`.

6.11.7 Concatenation of Matrices

The horizontal and vertical concatenation operators can be used to make big matrices out of small pieces. Assume `a` and `b` are 3x5 matrices. Then

```
a||b
```

is 3x10 and `a//b` is 6x5.

6.11.8 Identity Matrix

`i(k)` is the kxk identity matrix.

6.11.9 Constant Matrix

`j(r,c,value)` is the rxc matrix all of whose entries are value. If `value` is not specified, `value=1`.

6.11.10 Design Matrices

`design({1,1,3,2,1})` produces a design matrix for a one factor ANOVA (without additive side conditions) in which the treatments were 1,1,3,2,1.

`designf({1,1,3,2,1})` produces a design matrix for a one factor ANOVA with additive side conditions.

6.11.11 Repetition Factors

Repetition factors can be used to repeat an element used in constructing a matrix. `a={ [2] 3 [3] 4}={3 3 4 4 4}`.

6.11.12 Special Matrix Operations

`diag(matrix)` returns a matrix with the same diagonal entries as the argument, but zeroes off the diagonal.

`diag(vector)` returns a matrix with vector as its diagonal and 0 elsewhere.

`vecdiag(squarematix)` returns a vector whose entries are the diagonal entries of squarematrix.

`call eigen(eval, evec, matrix)` returns the vector of eigenvalues (`eval`) and matrix (`evec`) whose columns are the normalized eigenvectors of `matrix`. `Matrix` must be symmetric. In calling this routine, `eval` and `evec` are just names to be given to the returned matrices.

`ginv(a)` returns the Moore-Penrose generalized inverse of `a`.

`ncol(a)` returns the number of columns in the matrix `a`.

`nrow(a)` returns the number of rows in the matrix `a`.

`shape(a,r,c)` reshapes the matrix `a` to have `r` rows and `c` columns.

`solve(a,b)` solves the system ax=b for x. The answer is computed in a way that is more accurate than x=inv(a)*b.

`call svd(u,q,v,a)` returns the singular value decomposition (`u*diag(q)*v‘`) of the specified matrix `a`. In calling this routine, `u`, `q`, and`v` are just names for the returned matrix values.

For additional information see the online help under HELP–SAS SYSTEM–MODELING AND ANALYSIS TOOLS–INTERACTIVE MATRIX LANGUAGE or Chapter 7 of SAS/IML Software.

6.11.13 Reset Statement (IML)

The reset statement specifies certain options for printing and numeric calculation. Typical usage is

```
reset
        noprint
        fw=4;
```

Some of the available options are (defaults in parenthesis)

- print (noprint) specifies if intermediate results are printed
- pagesize=(21) number of lines per output page
- linesize=(78) width of output lines
- fw=(9) field width for numeric values
- flow (noflow) specifies if operations are shown as executed
- fuzz=(1E-12) (nofuzz) specifies if numbers smaller than fuzz are printed as zero
- (center) nocenter specifies if output is cetered or left justified
- storage=(work) specifies catalog for storage of modules and/or matrices

6.11.14 Mattrib Statement (IML)

Labels, row names, column names, and numeric formats can be associated with a matrix through the mattrib statement. Example:

```
mattrib m
        rowname=({ 'week one' 'week two'})
        colname=({ mon tue wed thur fri})
        label='Test Scores'
        format=3.0;
```

will cause the 2x5 matrix `m` to be printed with the label 'Test Scores' rather than the matrix name (`m`). The indicated names will appear bordering the matrix. Character vectors can be used for the rowname and colname options. The entries will be formatted with a total width of 3 and no places right of the decimal. Any of the rowname, colname, label, or format entries may be omitted.

6.11.15 Storing Modules and Matrices (IML)

Modules and matrices can be stored on disk and reloaded for use in later IML sessions.

First, use the reset storage=name; command to make `name` the current storage catalog. If permanent disk storage is desired, `name` should be a two level name. Only one IML storage catalog can be active at a time. The default catalog is SASUSER.IMLSTOR which is a temporary catalog.

The command

```
store  module=(add, subtract) a b c;
```

stores the modules `add` and `subtract` in the catalog along with the matrices `a`, `b`, and `c`. The word `_all_` can be used to store either all modules and/or all matrices. The simple form `store;` also stores all modules and matrices.

The command

```
load module=(add, subtract) a b c;
```

reloads the modules `add` and `subtract` and also the matrices `a`, `b`, and `c` for use in the current session. The word `_all_` can be used with load in the same way it is used with store. The simple form `load;` loads all modules and matrices from the current catalog.

The remove statement is used in the same way to remove modules and/or matrices from a catalog.

The command `show storage;` lists the modules and matrices that have been stored in the current catalog.

6.12 Proc GLM for MANOVA

A multivariate analysis of variance (MANOVA) can be carried out using Proc GLM. The method of doing such an analysis is described here. The special case of repeated measures is also considered.

The basic MANOVA model can be written in the form Y=XB+E where Y is an nxd matrix with the d dimensional observation vectors as rows, X is an nxk matrix of known constants, and B is a kxm matrix of unknown parameters.

As an example, look again at the Fisher Iris data of Project 2 and found in iris.dat. Let's consider only the sepal length and width measurements for each species. A simple model would be that the mean vector of these measurements depends only on species. Suppose SAS data set `sepal` contains the variables `sepallen`, `sepalwid`, and `species`. To test the hypothesis of equal mean vectors for the 3 species use:

```
proc glm data=iris;
class species;
model sepallen sepalwid=species;
manova h=_all_;
run;
```

The model statement specifies the model for the 2 independent variables in this case as depending only on species.

The manova h=_all_ statement carries out a manova analysis using the model and tests all hypotheses in the model of the form LB=0. This is the useful form for the first part of an analysis.

Variables not listed in the class statement are treated as quantitative variables. Multivariate linear regression models can be analyzed within proc glm. The independent variables of the regression appear in the model as above, but should not be listed in a class statement.

Hypotheses of the form LBM=0 can be tested by specifying the transpose of the matrix M in the manova statement. In the above analysis this could be used to test that the mean sepal length is the same as the mean sepal width, as follows:

```
proc glm data=sepal;
class species;
model sepallen sepalwid=species;
manova h=_all_
m=(1 -1) prefix=diff;
run;
```

Note that the m= statement specifies the TRANSPOSE of the matrix M in the hypothesis LBM=0.

The prefix= option specifies a (family) of names to be used for the new variables created by the M matrix transformation.

Contrast statements can be used to obtain customized hypothesis tests here, as discussed in proc glm (advanced). Contrast statements must appear before the manova statement.

The means statement can also be used.

A repeated measures design can also be analyzed. Suppose cholesterol readings are taken on each subject at 3 different times. In a purely multivariate model the 3 dimensional observation vectors for each subject could be assumed to form a sample from a multivariate normal distribution with unknown covariance matrix. The analysis would then proceed as above. If the covariance matrix is assumed to satisfy a sphericity condition a repeated measures analysis can be performed. Assume that the data set chol has data records with variables c1-c3 which contain the cholesterol readings for each patient. The program

```
proc glm data=chol;
model c1-c3=;
repeated time;
run;
```

will analyzed the data under the repeated measures structure and label the repeated measure as time in the output.

For further information see the SAS/STAT User's Guide, volume 2.

6.13 Proc Means

Proc means is used to compute simple statistics from a data set. Here is an example using some of the most common options.

```
proc means
     data=first
     alpha=0.10
     vardef=n
     mean std t prt n lclm uclm;
var x y;
by x;
run;
```

The required `data=first` statement selects the data set used in the analysis.

The `alpha=` option sets the significance level for any confidence intervals computed (defaults to alpha=0.05 if not specified).

The `vardef=n` option sets the divisor for the sample variance to be the sample size n (default is to use the degrees of freedom n-1 and is the same as `vardef=df`). NOTE: do not use the `vardef` option if you also use the `t` and/or `prt` options. THE T AND PRT VALUES GENERATED WHEN VARDEF=N ARE INCORRECT.

The optional last line gives a list of the statistics to be computed. The `mean` and `std` options give the sample mean and sample standard deviation for each variable in the analysis.

The `t` option computes the value of the t-statitistic for a test of zero mean for each variable.

The `prt` option computes the p-value for a test of the null hypothesis of zero mean for each variable.

The `n` option prints the sample size.

The `lclm` and `uclm` options produce the lower and upper endpoints of a 100(1-alpha)% confidence interval for the mean.

If no statistics are specified, `n mean std min` and `max` are printed for each variable. `Min` and `max` are the sample minimum value and maximum value.

The `var` option can be used to limit the output to a specific list of variables in the data set. If it is not used, the requested statistics will be computed for all variables in the data set.

The optional by statement generates separate analyses for each level of the by variable(s).

For additional information see the online help under SAS SYSTEM HELP–MODELING & ANALYSIS TOOLS–DATA ANALYSIS–(ELEMENTARY)MEANS or the SAS Procedures Guide.

6.14 Proc Npar1way

Proc npar1way performs nonparametric tests in a one factor model having 2 or more levels. Proc univariate can be used in the case of a single sample. Example:

```
proc npar1way
data=stuff
anova
edf
median
wilcoxon;
class litter; /* REQUIRED */
var weight height;
run;
```

The `data=stuff` option selects the SAS data set to be analyzed.

The `anova` option performs a one way ANOVA using the values of the `class` variable to define the populations.

The `edf` option calculates statistics based on the empirical distribution function. The Kolmogorov-Smirnov and Cramer-von Mises statistics are always calculated. If there are only 2 populations (as determined by the value of the `class` variable) the Kuiper statistic is also computed.

The `median` option performs the median or Brown-Mood test, depending on the number of populations.

The `wilcoxon` performs the Wilcoxon rank sum test or the Kruskal- Wallis test, depending on the number of populations.

The REQUIRED `class` statement specifies exactly one variable in the data set. The values of this variable are used to determine the population from which the corresponding observation comes. This variable MUST take at least 2 different values.

The optional `var` statement specifies the response variables which are to be analyzed. If the statement is omitted, all numeric variables in the data set are used.

For further details consult the SAS/Stat Users Guide volume 2.

6.15 Proc Plot

Proc plot can be used to produce rather crude graphical output on any printer. High quality graphics are produced with proc gplot.

Suppose the data set `airline` contains the number of air passenger miles flown each year in the variable `miles` and also the variable `year`. The commands

```
proc plot data=airline;
plot miles*year;
run;
```

produces a plot of the data. If the variable `revenue` contains revenue data for each year, two seperate plots can be obtained by

```
proc plot data=airline;
plot miles*year;
plot revenue*year;
run;
```

This could also be done using

```
proc plot data=airline;
plot miles*year revenue*year;
run;
```

A plot with both `miles` and `revenue` on one graph is obtained using the /overlay option.

```
proc plot data=airline;
plot miles*year revenue*year /overlay;
run;
```

For further details, see the online help or the SAS Procedures Guide.

6.16 Proc Print

Proc print lists the values of the variables in a SAS data set in the output window. Proc contents displays only the names and types of variables in a data set.

The most basic form is

```
proc print data=first;
run;
```

In this form, a complete listing of the values of all variables in the data set `first` will be printed in the output window. The `data=` statement is required. A more complete example is

```
proc print
data=second
label
noobs
heading=vertical;
var x y;
by x;
run;
```

The `label` option uses variable labels as column headings rather than variable names (the default).

The `noobs` option omits the `OBS` column of output.

The `heading=vertical` option prints the column headings vertically. This is useful when the names are long but the values of the variable are short.

The `var` option specifies the variables to be listed and the order in which they will appear.

The optional by statement produces output grouped by values of the by variable(s).

For further information see online help under SAS SYSTEM–REPORT WRITING–PRINT or the SAS Procedures Guide.

6.17 Proc Rank

Proc rank computes the ranks of the values of numeric variables. The ranks can then be used to conduct nonparametric statistical tests, among other uses. See also proc univariate and proc npar1way for conducting nonparametric tests. Example:

```
proc rank
data=stuff
out=newstuff /* highly recommended */
descending
fraction
ties=high;
ranks rankedx rankedy;
var x y;
run;
```

The `data=` option specifies the SAS data set containing the variables for which ranks are to be computed. If not specified the ranks will be placed in a data set named DATAn where n is an integer.

The `out=` option specifies the name of the SAS data set that will contain the ranks.

The `descending` option creates rankings with the largest value having rank 1. The default gives the smallest value rank 1.

The `fraction` option divides each rank by the number of nonmissing values taken by the variable.

The `ties=` option determines how tied values are assigned ranks. The default `ties=mean` uses midranks; `ties=low` uses the smallest of the corresponding ranks; `ties=high` uses the largest of the corresponding ranks. Using `ties=high` together with the `fraction` option produces the values of the empirical distribution function of the sample.

The `ranks` statement assigns names to the variables holding the ranks of variables listed in the `var` statement. The first name in the `ranks` statement is the name of the variable in the `out=` data set which holds the ranks of the first variable in the `var` statement, and so on. If the `ranks` statement is NOT used the variable names in the original data set are used as the names of variables in the `out=` data set with the values replaced by the ranks.

The `var` statement specifies the variables in the SAS data set for which ranks are to be computed. If omitted, all numeric variables in the data set are ranked. Variables not listed in the `var` statement are passed unchanged to the `out=` data set. If the `ranks` statement is NOT used the variable names in the original data set are used as the names of variables in the `out=` data set with the values replaced by the ranks. If the `ranks` statement is used the `var` statement MUST be used.

Other options are available to compute normal scores and Savage (exponential) scores. A by statement can also be used. For further information see the SAS Procedures Guide.

6.18 Proc Reg

The proc reg procedure is used to perform regression analysis.

Proc GLM can also be used to do this analysis by leaving the quantitative variables out of the `class` statement. In some ways, proc glm is superior to proc reg because proc glm allows manipulations in the model statement (such as x*x to obtain quadratic factors) which are not allowed in proc reg. However, proc reg allows certain automatic model selection features and a crude plotting feature not available in proc glm.

The variables analyzed using proc reg must be numeric variables all of which appear in a SAS data set. If `x`, `y`, and `z` are 3 numeric variables the basic invocation is

```
proc reg data=stuff;
model z= x y;
run;
```

There are many options available in the model statement. As in proc glm, the options are listed after a backslash on the model statement line. One example is

```
proc reg data=stuff;
model z= x y /
noint
selection=stepwise
sle=.05
sls=.05;
run;
```

The `noint` option specifies that the fitted model is to have NO intercept (constant) term.

The `selection=` option specifies how variables are to be introduced into the model. The default (if selection= is not used) is equivalent to `selection=none`, in which all the variables in the model statement are used. Setting `selection=stepwise` introduces a variable into the model provided it is significant at the `sle` level and deletes a variable from the model if it is NOT significant at the `sls` level. Setting `selection=rsquare` selects the model which has the maximum value of the square of R.

A final important option in proc reg is the `output` statement. This statement, which must follow the `model` statement, creates a SAS data set containing the variables in the original data set together with new variables as specified in the output statement. An illustration of some of the common options is

```
output
out=results
predicted=pred
residual=resid
L95M=lowmean
U95M=highmean
```

```
L95=lowpred
U95=highpred;
```

The `out=` option gives the name of the new SAS dataset.

The `predicted=` option gives the name of the variable in the `out=` data set which contains the predicted value of the dependent variable. By adding records to the original data set which specify values of the INDEPENDENT variables in the model but set the corresponding value of the DEPENDENT variable to missing, one can obtain predictions given by the model for unobserved settings of the independent variables.

The `residual=` option gives the name of the variable in the `out=` data set which contains the value of the residual.

The `L95M=` and `U95M=` options give the names of the variables in the `out=` data set which contain the lower and upper endpoints of a 95% confidence interval for the mean.

The `L95=` and `U95=` options give the names of the variables in the `out=` data set which contain the lower and upper endpoints of a 95% confidence interval for the predicted value.

For further information see the SAS/STAT User's Guide, volume 2.

6.19 Proc Sort

Proc sort is used to sort SAS data sets. This sorting is useful in producing printouts in the proper format when using by statements in other SAS procedures. Here's a typical example of using proc sort.

```
proc sort data=mixedup
     out=ordered;
by x  descending y;
```

In this example the original data set `mixedup` is sorted by ascending `x` values. The default sort order is ascending. The data is then sorted further within each value of `x` by descending `y` value. The sorted data set is named `ordered`.

The option `out=` specifies the name for the sorted data set. If `out=` is not used, the original data set is modified.

There is no limit to the number of variables which can be specified in the `by` statement. The by statement MUST appear.

For further information see SAS SYSTEM HELP–DATA MANAGEMENT–SORT in the online help or the SAS Procedures Guide.

6.20 Proc Spectra

Proc spectra is used to perform frequency domain analysis of time series data. Time domain (Box-Jenkins) analysis of time series is done using Proc Arima.

The main objective of frequency domain analysis is to identify periodicities in the time series. The modeling options available in proc arima are not available within proc spectra.

Typical invocation is:

```
proc spectra
data=skin
center
whitetest
p
s
out=spec;
var sun; /* required */
weights 1 2 3 2 1;
run;
```

The center option subtracts the mean of the time series from the observations before computing the periodogram.

The whitetest option performs Fisher's test for white noise.

The p option computes the periodogram of the series.

The s option computes the estimate of the spectral density of the series. If the s option is used, the weights option is required.

The out= option creates a SAS data set containing the output of the analysis. If the p option is specified, the periodogram values are included in the out data set; if the s option is specified, the estimates of the spectral density are included. The periodogram and spectral density estimates are in variables named p_01, p_02,... and s_01, s_02,... according to the order in which variables are listed on the var= statement. The out data set includes variables freq and period to allow for easy plotting of the periodogram or spectral density as a function of either frequency or period.

The var= statement specifies the variable(s) to be analyzed. This statement is required.

The weights= statement is required if the s option is used. Weights proportional to those specified are used to define a window and smooth the periodogram to produce the estimate of the spectral density.

For further information, see the SAS/ETS Guide.

6.21 Proc Ttest

Proc ttest is used to test the hypothesis of equality of means for two normal populations from which independent samples have been obtained. The test statistic is computed under the hypothesis that the two populations have equal variances, and an approximate test statistic is computed under no assumption about the equality of variances. Typical invocation is

```
proc ttest data=second;
     class type;
     var x y;
run;
```

THE CLASS SPECIFICATION IS REQUIRED. The class statement specifies a variable (here type) in the data set which takes exactly two values. The values of this class variable define the two populations to be compared.

The optional var statement restricts comparisons to the variables listed; if not used, all variables in the data set (execpt the class variable) are compared pairwise.

For further information see the online help under SAS SYSTEM HELP–MODELING & ANALYSIS TOOLS–DATA ANALYSIS–(analysis of variance) TTEST or the SAS/STAT User's Guide volume 2.

6.22 Proc Univariate

Proc univariate performs parametric and nonparametric analysis of a sample from a single population. See also proc means, proc ttest, proc glm, and proc npar1way.

Printed output for each variable includes the sample mean, the sample standard deviation, the t statistic and two sided p-value for a test of zero population mean, the sign statistic and two sided p-value for a test of zero population median, the Wilcoxon signed rank test statistic and two sided p-value for a test of zero population median, the quartiles and interquartile range, and some percentiles (1, 5, 10, 90, 95, 99). Other options listed below may produce additional output. Example:

```
proc univariate
data=stuff
freq
normal
plot
vardef=df;
var weight height;
```

The data= option specifies the SAS data set to be analyzed.

The freq option generates a frequency table showing the values, frequencies, percentages, and cumulative frequencies for each variable.

The normal option computes the Shapiro-Wilk or Kolmogorov test of the hypothesis that the data comes from a normal population. The p-value of the test is also printed.

The plot option produces a stem and leaf plot, box plot, and normal probability (quantile-quantile) plot for each variable. These are crude plots which do not use presentation quality graphics.

The vardef= option specifies the divisor used in computing the sample variance. vardef=df, the default, uses the degrees of freedom n-1; vardef=n uses the sample size n.

The optional var statement restricts the analysis to specific variables in the data set. If the var statement is not used, all numeric variables in the data set are analyzed.

Other options include computing weighted averages with arbitrary weights, using value-frequency pairs to specify the raw data, computing additional percentiles, and specifying the method used to compute percentiles. The procedure can create a new data set containing any of the computed quantities. A by statement can also be used. For further information see the SAS Procedures Guide.

7 Functions Reference

Here's a list of some of the functions available for use in SAS. For a complete list of functions which are available see the SAS online help under SAS SYSTEM HELP–SAS LANGUAGE–SAS FUNCTIONS–FUNCTION CATEGORIES or the SAS Language Reference.

Functions must appear as part of a SAS statement. Usually, this is as part of an assignment statement. Functions may be used as part of a logical test, e.g., in an if-then statement. Examples:

```
y=function(x);
function(x); /* not valid */
if function(x)>5 then ...;
```

- Beta Distribution Function
- Beta Distribution Quantiles
- Binomial Distribution Function
- Chi-Square Distribution Function
- Chi-Square Quantiles
- Dif Operator
- F Distribution Function
- F Distribution Quantiles
- Gamma Distribution Function
- Gamma Distribution Quantiles
- Hypergeometric Distribution Function
- Lag Operator
- Mean Function
- N Function
- Nmiss Function
- Normal Distribution Function
- Normal Distribution Quantiles
- Poisson Distribution Function
- Random Binomial Deviate
- Random Cauchy Deviate

- Random Normal Deviate
- Random Poisson Deviate
- Random Uniform Deviate
- Std Function
- Sum Function
- T Distribution Function
- T Distribution Quantiles

7.1 Dif Operator

The dif operators (dif, dif1, dif2,..., dif100) difference the values of a variable at the idicated lag and insert missing values as appropriate. The dif operators can only be used on the right hand side of assignment statements.

```
y=dif(x);  /* same as dif1(x) */
y=dif2(x); /* first difference of x at lag 2 */
y=dif(dif(x)); /* second difference of x */
```

7.2 Beta Distribution Function

`probbeta(x,a,b)` returns the value of the distribution function of the beta distribution with parameters `a` and `b`. Example:

```
y=probbeta(.25,3,7);
```

7.3 Binomial Distribution Function

`probbnml(p,n,m)` returns the value of the distribution function of the binomial distribution with parameters `p` and `n`. Example:

```
y=probbnml(.25,10,3);
```

7.4 Chi-Square Distribution Function

`probchi(x,df,nc)` returns the value of the distribution function of the chi-square distribution with `df` degrees of freedom and optional non-centrality parameter `nc`. If not specified, `nc=0`. `nc` is the sum of the squares of the means. Examples:

```
y=probchi(3,17);
z=probchi(3,17,5);
```

7.5 F Distribution Function

`probf(x,ndf,ddf,nc)` returns the value of the distribution function of the F distribution with `ndf` numerator degrees of freedom, `ddf` denominator degrees of freedom, and optional non-centrality parameter `nc`. If not specified, `nc=0`. `nc` is the sum of the squares of the means. Examples:

```
y=probf(3.57,4,12);
z=probf(3.57,4,12,2);
```

7.6 Gamma Distribution Function

`probgam(x,a)` returns the value of the distribution function of the gamma distribution with parameter `a`. Example:

```
y=probgam(4.3,2);
```

7.7 Hypergeometric Distribution Function

`probhypr(N,K,n,x)` returns the value of the distribution function of the hypergeometric distribution with total population size `N`, category of interest size `K`, and sample size `n`. Example:

```
y=probhypr(52,13,5,2);
```

7.8 Mean Function

The `Mean` function returns the average of the values of the argument variables which have nonmissing values. Examples:

```
avg=Mean(x, y, z);
avg=Mean(of x1-x10);
avg=Mean(of x y z);
```

The first and last example assign `avg` the average of the variables `x`, `y`, and `z` which have nonmissing values. Note the differences in syntax. The last example returns the average of the variables among `x1, ..., x10` which have nonmissing values.

7.9 N Function

The `N` function returns the number of argument variables which have nonmissing values. Examples:

```
count=N(x, y, z);
count=N(of x1-x10);
count=N(of x y z);
```

The first and last example assign `count` the number of the variables `x`, `y`, and `z` which have nonmissing values. Note the differences in syntax. The last example returns the number of the variables among `x1, ..., x10` which have nonmissing values.

7.10 Nmiss Function

The `Nmiss` function returns the number of argument variables which have missing values. Examples:

```
count=Nmiss(x, y, z);
count=Nmiss(of x1-x10);
count=Nmiss(of x y z);
```

The first and last example assign `count` the number of the variables `x`, `y`, and `z` which have missing values. Note the differences in syntax. The last example returns the number of the variables among `x1, ..., x10` which have missing values.

7.11 Normal Distribution Function

`probnorm(x)` returns the value of the distribution function of the standard normal distribution. Example:

```
y=probnorm(1.96);
```

7.12 Poisson Distribution Function

`poisson(m,n)` returns the value of the distribution function of the poisson distribution with parameter `m`. Example:

```
y=poisson(3.2, 10);
```

7.13 T Distribution Function

`probt(x,df,nc)` returns the value of the distribution function of the t distribution with `df` degrees of freedom and optional non-centrality parameter `nc`. If not specified, `nc=0`. `nc` is the value of the mean. Examples:

```
y=probt(1.96,10);
z=probt(1.96,10,3);
```

7.14 Lag Operator

The lag operators (lag, lag1, lag2,..., lag100) return the value of the variable at the indicated lag, or the SAS missing value, as appropriate. The lag operators can only be used on the right hand side of assignment statements.

```
y=lag(x); /* same as y=lag1(x) */
y=lag4(x);
```

7.15 Beta Distribution Quantiles

`betainv(p,a,b)` returns the pth quantile, `0<p<1`, of the beta distribution with parameters `a` and `b`. Example:

```
y=betainv(.05,1,2);
```

7.16 Chi-Square Quantiles

`cinv(p,df,nc)` returns the pth quantile, `0<p<1`, of the chi-square distribution with degrees of freedom `df`. If the optional non-centrality parameter `nc` is not specified, `nc=0`. `nc` is the sum of the squares of the means. Examples:

```
y=cinv(0.95,5);
z=cinv(0.95,5,3);
```

7.17 F Distribution Quantiles

`finv(p,ndf,ddf,nc)` returns the pth quantile, `0<p<1`, of the F distribution with `ndf` numerator degrees of freedom, `ddf` denominator degrees of freedom, and optional `nc` as the non-centrality parameter. If not specified, `nc=0`. `nc` is the sum of the squares of the means. Examples:

```
y=finv(.95,3,12);
z=finv(.95,3,12,4);
```

7.18 Gamma Distribution Quantiles

`gaminv(p,a)` returns the pth quantile, `0<p<1`, of the gamma distribution with parameter `a`. Example:

```
y=gaminv(.95,7);
```

7.19 Normal Distribution Quantiles

`probit(p)` returns the pth quantile, `0<p<1`, of the standard normal distribution. Example:

```
y=probit(.975);
```

7.20 Sum Function

The `Sum` function returns the sum of the argument variables which have nonmissing values. Examples:

```
total=Sum(x, y, z);
total=Sum(of x1-x10);
total=Sum(of x y z);
```

The first and last example assign `total` the sum of the variables `x`, `y`, and `z` which have nonmissing values. Note the differences in syntax. The last example returns the sum of the variables among `x1, ..., x10` which have nonmissing values.

7.21 Std Function

The `Std` function returns the standard deviation of argument variables which have non-missing values. The divisor is n()-1. Examples:

```
sdev=Std(x, y, z);
sdev=Std(of x1-x10);
sdev=Std(of x y z);
```

The first and last example assign `sdev` the standard deviation of the variables `x`, `y`, and `z` which have nonmissing values. Note the differences in syntax. The last example returns the standard deviation of the variables among `x1, ..., x10` which have nonmissing values.

7.22 T Distribution Quantiles

`tinv(p,df,nc)` returns the pth quantile, `0<p<1`, of the t distribution with `df` degrees of freedom and optional non-centrality `nc`. If not specified `nc=0`. `nc` is the value of the mean. Examples:

```
y=tinv(.95,10);
z=tinv(.95,10,5);
```

7.23 Random Binomial Deviate

`ranbin(seed,n,p)` returns a pseudo observation of a binomial random variable with parameters `n` and `p`. Example:

```
y=ranbin(0,10,.5);
```

7.24 Random Cauchy Deviate

`rancau(seed)` returns a pseudo observation of a cauchy random variable. Example:

```
y=rancau(0);
```

7.25 Random Normal Deviate

`rannor(seed)` returns a pseudo observation of a standard normal random variable. Example:

```
y=rannor(0);
```

7.26 Random Poisson Deviate

`ranpoi(seed,m)` returns a pseudo observation of a poisson random variable with parameter `m`. Example:

```
y=ranpoi(0,3.2);
```

7.27 Random Uniform Deviate

`ranuni(seed)` returns a pseudo observation of a uniform random variable on the interval (0,1). Example:

```
y=ranuni(0);
```

8 Configuration Options

Configuration options control the behavior of various parts of the SAS system.

- Autoexec.sas File
- Config.sas File
- Display Manager Statement
- Filename Statement
- Goptions Statement
- Libname Statement
- Linesize Option
- Options Statement
- Pagesize Option
- Symbol Statement
- Title Statement

8.1 Autoexec.sas File

Commands in the autoexec.sas file are read when the SAS program is started and are used to set default configuration options for SAS. Some options are set in the config.sas file which is also read at program invocation.

Users should set their personal values for options in files, say myauto.sas and myconfig.sas, and then ivoke SAS with the command line such as

```
sas autoexec=myauto.sas config=myconfig.sas
```

The exact form for the invocation is system/setup dependent.

Here is a sample autoexec file.

```
/*
Set page and line size for
output
*/

options ps=60 ls=70;

/*
Define some librefs
*/
libname jv 'c:\sas\jv\';
libname gdevice0 'c:\sas\jv';

/*
Set some basic graphics options
so screen output looks like printer
output.
*/
goptions cback=white
        colors=(black)
        targetdevice=psll
        interpol=join
        ftext=swiss;
/*
Define some symbols for different
line styles.
*/
symbol1 l=1;
symbol2 l=2;
```

For further information consult the SAS Programming Companion for the environment in which you run SAS, e.g., Microsoft Windows, UNIX, etc.

8.2 Config.sas File

The config.sas file is read by SAS at startup. and sets default configuration options for the SAS program. Other options are set in the autoexec.sas file which is also read at program invocation.

Users should set their personal values for the options in files, say myauto.sas and myconfig.sas, and then invoke SAS with a command line such as

```
sas autoexec=myauto.sas config=myconfig.sas
```

The exact form for the invocation is system/setup dependent.

For further information consult the SAS Programming Companion for the environment in which you run SAS, e.g., Microsoft Windows, UNIX, etc.

8.3 Goptions Statement

The goptions statement controls most of the HARDWARE features associated with high quality plotting/graphing done using proc gplot.

Frequently used settings should be placed in a goptions statement appearing in your autoexec.sas file. The goptions statement can be used in the program editor window. The goptions statement should not be used inside a proc or data step.

Here is a basic goptions statement that will serve most needs of Microsoft Windows users. This example sets things up for black on white printing.

```
goptions
device=win
targetdevice=winprtm
cback=white
colors=(black)
ftext=swiss
interpol=join;
```

The `cback` option sets the background color (to white here).

The `colors` option selects the possible foreground colors, here black only. The `back` and `colors` options here should make the appearance of graphs on screen look as much as possible like their printed counterparts. If you have a color printer, you may wish to allow more colorful graphs on screen as well as in print.

The `targetdevice` specifies hardware driver for printed output. There are many options available here. The one selected uses the standard Microsoft Windows driver.

The `ftext` option specifies the font to use for text on the graph/plot. Many options are available.

The `interpol=` option specifies the default type of interpolation to be used to connect the dots of the graph. Most common is `interpol=join` in which data points are joined by straight lines. Other common options here are `interpol=needle` for needle graphs, and `interpol=` for no joining of data points. The setting specified here can be overridden in the actual plot procedure.

A more complicated setup may be required when using SUN workstations. Here is a prototype:

```
filename sparc pipe 'lpr -Psparc';
goptions device=xcolor
target=ps
rotate=landscape
cback=white
colors=(black)
interpol=join
```

```
gsfname=sparc
gsfmode=replace
gaccess=sasgaedt
gsflen=80
noprompt
gprolog='25210D0A'x;
```

The rotate option is used to produce landscape style output. This option is not implemented for all printers and may cause unpredictable results on others.

For additional information see the SAS/GRAPH User's Guide.

8.4 Options Statement

The options statement is used to change the default values of SAS system parameters. The options statement should NOT appear inside a data step or a proc step. Example:

```
options
pagesize=60
linesize=80
pageno=1
nodate
nocenter;
```

The nodate option suppresses printing of the date and time information on each page of the output; the date option restores the default behavior.

The nocenter option produces left justified output; the center option restores the default centering of output.

The options statement usually appears in the autoexec.sas file, but can be typed in the program editor window.

9 Books About SAS

Here's a list of books that may be useful.

Rebecca J. Elliott, "Learning SAS in the Computer Lab", Duxbury (1995) ISBN 0-534-23442-9.

Rick Aster, "Professional SAS Programmer's Pocket Reference", Windcrest/McGraw-Hill (1993).

Schlotzhauer and Littell, "SAS System for Elementary Statistical Analysis", SAS Institute (1987) ISBN 1-55544-076-2.

Cody and Smith, "Applied Statistics and the SAS Programming Language", third edition Prentice Hall (1991) ISBN 0-13-500554-X.

"SAS Language and Procedures: Usage", SAS Institute (1989) ISBN 1-55544-371-0.

DiIorio, "SAS Application Programming: A Gentle Introduction", PWS-Kent (1991) ISBN 0-534-98464-9.

SAS Reference Books include the following. A copy of each is available for use in the Division of University Computing office in the L building.

“SAS/ACCESS User’s Guide”

“SAS Companion for the Microsoft Windows Environment” version 6 first edition, SAS Institute (1993) ISBN 1-55544-527-6.

“SAS/ETS User’s Guide,” version 6 second edition, SAS Institute (1993) ISBN 1-55544-554-3.

“SAS/GRAPH User’s Guide,” release 6.03 edition, SAS Institute (1988) ISBN 1-55544-087-8.

“SAS Language Reference version 6”, SAS Institute (1993) ISBN 1-55544-381-8.

“SAS Procedures Guide,” version 6 third edition, SAS Institute (1990) ISBN 1-55544-378-8.

“SAS/STAT User’s Guide, volume 1” version 6 fourth edition, SAS Institute (1990) ISBN 1-55544-376-1.

“SAS/STAT User’s Guide, volume 2,” version 6 fourth edition, SAS Institute (1990) ISBN 1-55544-376-1.

“SAS/IML Software,” version 6 first edition, SAS Institute (1990) ISBN 1-55544-377-X.

10 Bibliography

D.F. Andrews and A.M. Herzberg, “Data”, Springer Verlag (1985).

J. D. Cryer, “Time Series Anlaysis”, Duxbury Press (1986).

M. O. Finkelstein and B. Levin, “Statistics for Lawyers”, Springer Verlag (1990).

H. Scheffe, “Analysis of Variance”, John Wiley (1959).

C. R. Hicks, “Fundamental Concepts of the Design of Experiments”, third edition, Addison Wesley (1982).