Introduction to SAS

Acknowledgement --- Credit where credit is due : This is a truncated version of "Introduction to SAS," a website which no longer exists, written by Brian Yandell, Department of Statistics, University of Wisconsin. It will provide you with a minimal SAS knowledge which is essential in STAT 401 at University of Idaho.

A. SAS for Windows at U. of Idaho

The SAS system resides on the Novell network at the University of Idaho. You must have access to the system or have SAS for Windows installed on a personal computer. You may also want to carry a disk to the computer labs to save results.

1. Getting Started

You are now in SAS for Windows. There is a menu bar at the top of the screen which shows:

```
FILE  EDIT  VIEW  LOCALS  GLOBALS  OPTIONS  WINDOWS  HELP
```

This menu bar helps you manage your program. Some examples of things you can do are open, save, print a SAS file, choose from different SAS subwindows, or get help.

There are three primary windows in SAS. The 'Program Editor' window is used to enter, edit, and submit SAS programs. The 'Log' window displays messages about the programs you submit. The 'Output' window displays output from procedures.

Only one window at a time is active. The active window is where your cursor is located. Commands you give through the SAS menu bar will be executed in the active window. You can scroll through windows using the arrow buttons, and can enlarge a window by clicking on the up arrow in the right hand corner of the window. To change windows use the mouse to select a window, or choose 'Windows' from the menu bar at the top of the screen.

2. Using the SAS System For Data Analysis

A SAS program consists of a series of statements. Each statement must end with a semicolon. The two main steps in preparing a SAS program are the DATA step which creates a SAS data set, and the PROCEDURE step which performs tasks on the data. To enter your SAS program make sure your cursor is in the 'Program Editor' window, and then you may begin typing.

3. Subwitting or Running a SAS Program

To submit a SAS program make sure that your cursor is in the 'Program Editor' window and press <F8> or click on the *person running* icon to submit a SAS program. Or through the menu bar at the top of the screen choose 'Locals', 'Submit'. Your program will disappear and the 'Log' window will start showing your SAS statements and notes with information about the programs progress. The output window will come up automatically when the program is finished, and will have the results of your procedures. You can print or scroll through any of the windows that you wish. If something is wrong with your SAS program, SAS will give red error messages in the LOG file. If you receive red error messages you should go back to the "Program Editor", recall your program by hitting <F4> or choosing "Locals" and "Recall text" from the menu bar, and then correct your codes and rerun the program. Before re-running a program it is often helpful to clear the LOG and OUTPUT windows of their text from the run that generated the error messages. An easy way to do this is to select the window and use <CNTRL>-<E> to clear it.

4. Function Keys

The menu bar at the top of the screen will help you manage your SAS program. SAS has created some function keys for the commonly used commands to help you save time. Some of these keys are <F4> recall the program, <F5> go to the 'Program Editor' window, <F6> 'Log' window, <F7> Output' window, and <F8> submit a program to be run. To get a complete list of function keys choose 'HELP' and then 'KEYS' from the menu bar.

5. To Save, Retrieve, and Print

Click the following menu commands from the menu bar at top of the screen with the mouse.

To Save: SAS will save the active window that your cursor is in.

> Click 'FILE'. Click 'SAVE AS' if the file is new or you want to rename the file. Otherwise click 'SAVE'. Specify the drive and the name of the file. You can save it on either a floppy disk or a hard drive. If you are working on University Computers it is a good idea to save your work on a floppy disk or to the G drive. Click 'OK'.

To Retrieve a File: If you want to work with a file that has been previously saved make sure the 'Program Editor' window is active.

> Click 'FILE'. Click 'OPEN' Select the drive where your file has been saved, choose the file name, and click on the 'OK' box. If successful you will see the file in the 'Program Editor' window.

Printing: You can print from any window.

> Make sure the window that you wish to print is active by selecting it with your mouse. Click 'FILE'. Click 'PRINT'. From the print window, select the proper printer options with your mouse. Click 'OK'.

6. **Exiting The Program**

Before exiting, save any files needed for future use. Then, Select 'File' from the menu bar. Select'Exit'.

7. **For More Information**

Two books for further reading are: *SAS user's Guide: Basic* and *SAS User's Guide: Statistics*. There is a SAS tutorial available on the Novell Network at any of the student labs. To run the tutorial, at the main menu select: 'Tutorials', and then 'SAS Tutorial'. The SAS Institute now provides much information online.

B. SAS Language

Suppose the file `myfile.sas` contains some SAS codes. The `myfile.sas` usually contains three kinds of sas "paragraphs" or "steps", namely:

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

SAS reads your file in "free" format--use as many spaces or tabs as you like--but requires a semicolon (`;`) at the end of each "phrase" or "sentence". It is good practice to indent phrases in `data` or `proc` "paragraphs" for ease of reading.

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

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

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

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

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

```
   nocenter    do not center output (flush right instead)
   linesize=80 set width of page to 80
   ls=80       same as linesize
   pagesize=50 set length of page to 50
   ps=24       same as pagesize
```

C. SAS Data Steps

- Data Input
- Data Transformations
- Data Manipulation
- Random Numbers
- Experimental Design Using SAS

---

1. Data Input

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

```
data direct;
   input x y;
   cards;
1       17.5
```

```
3       20.5
;
```
Here is an example of data input from another file:
```
data pulse;
   infile 'a:\pulse.dat' missover;
   input x y;
```
The names `direct` and `pulse` are arbitrary, but can be used later in your SAS program to identify this particular data set. Details of input phrases (use either `infile` or `cards`, but not both):
```
data a;                          create new data set named "a"
   input x y z;                  input 3 numbers at a time as variables
x,y,z
   input trt $ x y;              input treatment "trt" as a character
string
                                 and x,y as numbers. Note the dollar sign
($).
   infile 'blah.dat' missover;   use file "blah.dat" for the data
                                 "missover" skips over missing data
rather
                                 than going to a new line
                                 (must appear BEFORE the input phrase)
   infile 'blah.dat' firstobs=2; skip first observation (first line)
                                 handy way to document column names
   infile 'blah.dat' lrecl=2000; allow for really long records
   cards;                        read data from following lines
                                 (must appear AFTER the input phrase)
;                                end of data entry for "cards" phrase
                                 (good convention, but not required)
```
Data values must have spaces between them (tabs can cause problems on some systems). All values must be on the same line if using the `missover` option. Missing data is represented by a period (.) as place holder. This can also be useful for estimation and prediction at new values using `proc reg`.

---

2. Data Transformations

There is no need to transform your raw data outside of SAS. In fact, it is good practice to leave your data file alone once it is debugged. Transforms are usually done in a separate data paragraph after data input. Here you need to identify the data set previously run. An example:
```
data logs; set direct;
   logy = log(y);
```
This creates a new data set `logs` from the set `direct` from data input above. The variable `logy` is created as the natural log of the variable `y`. Here are details of the first line and some transformations:
```
data a; set b;          create data set "a" using existing set "b"
   z = log(y);          create variable z as natural log of variable y
   z = log10(y);        log base 10
   z = sqrt(y);         square root
   z = x*y;             multiplication
                        (+ addition) (- subtraction) (/ division)
   z = y**2;            exponent: "y squared" or "y to the 2nd power"
```

```
   z = y**0.5;         "y to the 1/2 power" (same as sqrt(y))
   z = x**-2;          negative exponent: "1 over (x squared)"
   z = sin(x);         trigonometric sine function of x
                       (also cos(x), tan(x), ...)
```

Variance Stabilizing Transformations

```
data a; set b;
   z = sqrt(count);           /* counts (Poisson distribution) */
                              /* variance proportional to mean */
   z = log(conc);             /* concentrations, weights (log normal)
*/
                              /* SD proportional to mean */
                              /* constant coefficient of variation (CV)
*/
   z = arsin(sqrt(prop));     /* proportions (0-1) */
   z = arsin(sqrt(pct/100));  /* percentages (0-100) */
                              /* (Binomial distribution) */
                              /* variance proportional highest in
middle */
```

---

3. Data Manipulation

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

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

Suppose you had data set `field` with `3` treatments called `control, wet, dry` and you wanted to delete the `control` group for some procedures,

```
data trtonly; set field;              /* create trtonly from field */
   if trt = 'control' then delete;    /* delete control group */
```

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

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

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

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

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

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

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

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

```
data abc;
   input n0 n1 n2 n3 n4 n5;
   cards;
1.4     1.5     1.2     2.1     2.1     2.8
1.7     1.4     1.0     1.4     1.7     2.1
1.1     1.9     2.5     2.6     2.1     2.2
1.7     1.3     1.1     1.0     2.0     1.8
1.0     1.8     1.5     1.4     2.2     2.3
data resps; set abc;
   resp = n0; level = 0; output;
   resp = n1; level = 1; output;
   resp = n2; level = 2; output;
   resp = n3; level = 3; output;
   resp = n4; level = 4; output;
   resp = n5; level = 5; output;
   drop n0--n5;
```

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

---

4. Random Numbers

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

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

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

```
   x = ranuni(seed)           /* uniform between 0 & 1 */
   x = a+(b-a)*ranuni(seed);  /* uniform between a & b */
   x = ranbin(seed,n,p);      /* binomial size n prob p */
   x = rancau(seed);          /* cauchy with loc 0 & scale 1 */
   x = a+b*rancau(seed);      /* cauchy with loc a & scale b */
   x = ranexp(seed);          /* exponential with scale 1 */
   x = ranexp(seed) / a;      /* exponential with scale a */
   x = a-b*log(ranexp(seed)); /* extreme value loc a & scale b */
   x = rangam(seed,a);        /* gamma with shape a */
   x = b*rangam(seed,a);      /* gamma with shape a & scale b */
   x = 2*rangam(seed,a);      /* chi-square with d.f. = 2*a */
   x = rannor(seed);          /* normal with mean 0 & SD 1 */
   x = a+b*rannor(seed);      /* normal with mean a & SD b */
   x = ranpoi(seed,a);        /* poisson with mean a */
   x = rantri(seed,a);        /* triangular with peak at a */
   x = rantbl(seed,p1,p2,p3); /* random from (1,2,3) with probs */
```

```
                                        /* p1,p2,p3 */
```
The `seed` above is either `0` (use clock to randomly start sequence); positive (used as initial `seed` -- it should be odd and less than `2**31-1`); or negative (use the clock to restart the sequence every time). The performance is untested for `0` or negative `seed` -- use at your own risk. The `seed` is only examined on the first encounter with a random number generator in your program, so you cannot change the process once you begin.

---

5. Experimental Design Using SAS

Experimental designs can be laid out using SAS. Here is an example of a design with 4 treatments and 5 replicates per treatment. Suppose set `b` has identifiers called `id`. This assigns `trt` the values 1,2,3,4, each with 5 replicates.

```
data uniform;
   do i = 1 to 20;
      x = ranuni(0);
      output;
   end;
data a;
   merge b uniform;
proc sort; by x;
data c; set a;                  /* _N_ = line number */
   trt = ceil(_N_ / 5);                 /* ceil = next highest integer */
proc sort; by id;
proc print;
   var id trt
```

D. SAS Procedures

- Introduction to Procedures
- Sorting and Running a `proc` by Subgroups
- Numerical Summaries
- Graphical Summaries

---

1. Introduction to Procedures

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

```
proc print;
```

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

```
proc print data=a;
```

explicitly uses the data set a rather than the previously or created one. Procedures usually produce printed output (in `myfile.lst`), but do not create a new or add to existing data sets unless this is made explicit with an `output` phrase. For instance,

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

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

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

with `out=` being the keyword for the data set name `d1` and `a= b=` and `c=` being any number of optional keywords for new variables. The names after the equals -- `a1, b1` and `c1`, respectively -- are up to you. They are the names of these variable that you can later use. Now to specifics. Here I give some phrases which may be useful. Others can be found in the SAS/STAT book. For the `output` phrase, I indicate some keywords.

---

2. Sorting and Running a `proc` by Subgroups

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

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

will first sort the data by treatment `trt` and then run the `means` procedure separately for each treatment group. This is much cleaner than running SAS 3 times, each time retaining only the treatment group under study. However, it does produce a lot more output! The `by` phrase is on all procedures. BUT you MUST sort before you use it. You can sort by several things at once:

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

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

NOTE: While you can get separate printed listings for each treatment (in `myfile.lst`), you get only one data set if you use `by`.

---

3. Numerical Summaries

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

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

```
proc means noprint;           /* useful form if you do not want
printout */
   var x y;
   output out=b mean=mx my std=sx sy;/* output means and SD for x,y */
```

---

4. Graphical Summaries

The main character-based graphic routines are `univariate plot` (1-dimensional) `plot` (two-dimensional). There is a system of fancy graphics routines (begining with letter `g`) introduced briefly at the end of this section. [Feedback so far is that manuals for these are confusing.] In addition, SAS has a module called INSIGHT which some have found very nice for graphics and general user interface.

```
proc univariate plot normal;  /* histogram type summaries */
   var x;
```

The `plot` option to `proc univariate` produces a stem-and-leaf plot, a box plot, and a normal probability plot. The `normal` option tests for normal distribution.

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

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

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

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

Here is a fancier way to construct Interaction Plots and some Diagnostic Plots, which allows you to use the full value of statistical modelling. The basic idea is to fit the desired model, save the least squares means (`lsmeans`) as a dataset, and print from that. Diagnostic information is saved with the `output` phrase.

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

Here is a way to keep `uniform` axes for separate plots by `location`:

```
proc plot uniform; by location;
```

```
   plot y*x;
```
You can set several plot features:
```
   plot y*x / vaxis=10 to 100 by 5;   /* vertical axis ticks */
   plot y*x / haxis=10 to 20 by 2;    /* horizontal axis ticks */
   plot y*x / vzero hzero;            /* include origin on plot */
   plot ry*py / href=0;                      /* horizontal reference
line */
   plot y*x py*x='*' / overlay;              /* overlay two plots */
```

E. Linear Models in SAS

(Regression & Analysis of Variance)

The main workhorse for regression is `proc reg`, and for (balanced) analysis of variance, `proc anova`. The general linear model `proc glm` can combine features of both. Further, one can use `proc glm` for analysis of variance when the design is not balanced. Computationally, `reg` and `anova` are cheaper, but this is only a concern if the model has 50 or more degrees of freedom.

- Regression
- Analysis of Variance (ANOVA)
- Analysis of Covariance (ANCOVA)
- General Linear Models (GLM)

---

1. Regression

Here are simple uses of `proc reg` for standard problems:
```
proc reg;              /* simple linear regression */
   model y = x;

proc reg;              /* multiple regression */
   model y = x1 x2 x3;
```
The `model` phrase indicates which variables are response (`y`) and which are predictors (`x`, or `x1,x2,x3`). Here are some print options for the model phrase:
```
   model y = x / noint;               /* regression with no intercept
*/
   model y = x / p;           /* print predicted values and residuals
*/
   model y = x / r;           /* option p plus residual diagnostics */
   model y = x / clm;         /* option p plus 95% CI for estimated
mean */
   model y = x / cli;         /* option p plus 95% CI for predicted
value */
   model y = x / r cli clm;   /* options can be combined */
```

CAUTION: *SAS listings label the standard error of the estimated mean as the* `STD ERROR PREDICT`. *Be wary and know what these things mean! Some of the residual diagnostics go beyond the Stat 401 material cover. You may explore these on your own.*

It is possible to let SAS do the predicting of new observations and/or estimating of mean responses. The way to do this is to enter the `x` values (or `x1,x2,x3` for multiple regression) you are interested in during the data input step, but put a period (.) for the unknown `y` value. That is,

```
data new;
   input x y;
   cards;
1 0
2 3
3 .
4 3
5 6
;
proc reg;
   model x = y / r cli clm;
```

Try it, and check standard errors and confidence intervals by hand. Here are some other model options for more advanced stuff:

```
   model y = x / covb; /* covariance matrix for estimates */
   model y = x / collin;       /* collinearity diagnostic */
   model y = x / collinoint;   /* collin without intercept */
```

The `output` phrase can have several keywords (which can be used together):

```
   output out=b predicted=py;  /* predicted values in "py" */
   output out=b p=py;          /* same as predicted */
   output out=b residual=ry;   /* residual values in "ry" */
   output out=b r=ry;          /* same as residual */
   output out=b stdr=sr;       /* standard error of residuals "sr" */
   output out=b student=sy;    /* studentized residuals "sy" */
```

Only one `output` phrase can be used, but you can combine keywords on one line:

```
   output out=b p=py r=ry stdr=sr student=sy;
```

Those new variables created in set `b` are available for later plotting, etc.

---

2. Analysis of Variance

Experiments involving a single factor or several factors with no missing data (balanced designs) can use the quick and easy `proc anova` to analyze the variation explained by those factors (analyis of variance, or ANOVA). More complicated ANOVA designs can be done PROVIDED the data are balanced. However, designs with imbalance among two or more factors should use `proc glm`.

```
proc anova;                    /* one-way analysis of variance */
   class trt;
   model y = trt;

proc anova;                    /* 1-way with multiple comparisons */
   class trt;
   model y = trt;
```

```
   means trt / lsd tukey;             /* LSD and Tukey's studentized
range  */

proc anova;                    /* two-way anova */
   class fert var;
   model y = fert var;
   means fert var / lsd;       /* means by fert and var with LSD */

proc anova;                    /* two-way anova with interaction */
   class fert var;
   model y = fert var fert*var;       /* interaction signified by
asterisk */
   means fert var / lsd;
   means fert*var;             /* for each fert-var combination */
```

The `class` phrase is required, identifying all factors as categorical variables. The `model` phrase has only a few options, and these are not often used. The `means` phrase is quite handy to do multiple comparisons. Options include:

```
   means trt / t;              /* Least Significant Difference */
   means trt / lsd;            /* Least Significant Difference */
   means trt / bon;            /* Bonferroni */
   means trt / tukey;           /* Tukey's studentized range  */
   means trt / lsd alpha=.05;  /* LSD at level 5% (default) */
   means trt / lsd cldiff;     /* force pairwise tests of means */
```

The `cldiff` option can be useful at times, but it only gives differences CI for the differences, not the means themselves. None of these options works when looking at 2-way combinations such as `means fert*var;`.

If you want to save predicted values or residuals, or to evaluate contrasts, you must use `proc glm` instead of `proc anova`. See below.

---

3. Analysis of Covariance (ANCOVA)

Analysis of Covariance, or ANCOVA, combines features of ANOVA and regression. That is, examine treatment differences adjusted for covariate. Similarly, determine whether there is a significant relationship between `X` and `Y` after adjusting for treatment. ANCOVA is typically done using `proc glm`.

```
proc glm;                      /* analysis of covariance */
   class trt;                  /* trt = factor, x = covariate */
   model y = x trt;

proc glm;                      /* analysis of covariance */
   class trt;                  /* with different slopes */
   model y = x trt x*trt;
```

---

4. General Linear Models (GLM)

The general linear models (GLM) procedure works much like `proc reg` except that we can combine regressor type variables with categorical (`class`) factors. The organization of the printout is slightly different from `reg` and `anova`, and some model and output options are different. Further, if you want model parameter estimates, it is best to explicitly request the `solution` option in the `model` phrase.

```
proc glm;              /* simple linear regression */
   model y = x / solution;

proc glm;              /* multiple regression */
   model y = x1 x2 x3 / solution;

proc glm;              /* one-way analysis of variance */
   class trt;
   model y = trt;

proc glm;              /* additive two-factor anova */
   class fert var;
   model y = fert var;

proc glm;              /* full two-factor anova (i.e. interaction
included) */
   class fert var;
   model y = fert | var;  /* or, model y = fert var fert*var; */

proc glm;               /* full three-factor anova (i.e. all
interactions included) */
   class var1 var2 var3;
   model y = var1 | var2 | var3;

proc glm;              /* analysis of covariance */
   class trt;          /* trt = factor, x = covariate */
   model y = x trt;
```

The `class` phrase works like in `proc anova`. However, here we can have both categorical (identified in `class`) and continuous variables in the `model`. The `model` phrase indicates which variables are response (`y`) and which are predictors (`x`, or `x1,x2,x3`). You won't get parameter estimates (`solution`) if there is a `class` phrase unless you ask for them. Here are some options:

```
   model y = trt x / solution; /* print parameter estimates and SEs */
   model y = x / noint;               /* no intercept (as in proc reg)
*/
   model y = x / p;           /* print predicted values and residuals
*/
   model y = x / clm;         /* option p plus 95% CI for estimated
mean */
   model y = x / cli;         /* option p plus 95% CI for predicted
value */
   model y = x / cli alpha=.01;       /* only .01, .05 and .10
available */
```

The `means` phrase works much the same in `proc glm` as in `proc anova`. Contrasts can be set up if `means` aren't enough. Here is an example from the glue data. The `contrast` phrase contains a quoted title, variable name and the contrast coefficient values. Note that the order of factor levels is lexicographic, which may not be what you expect. This can be

checked by examining the order under the `solution` option to the `model` phrase. Further, these can get very complicated for higher order designs. Consult a book for further help.

```
contrast 'A vs. rest' glue 1 -.25 -.25 -.25 -.25;
contrast 'BD vs. CE' glue 0 .5 -.5 .5 -.5;
```

Predicted and residual (and other) values can be passed to other procedures and data steps using the `output` phrase in the same manner as `proc reg`.
