MacAnova Version 1.0

MacAnova User's Guide

by

Gary W. Oehlert

University of Minnesota
School of Statistics
Technical Report Number 493

August 1987

MacAnova Version 1.0

MacAnova User's Guide

by

Gary W. Oehlert

University of Minnesota
School of Statistics

August 1987

Contents

1. Introduction

Preface. MacAnova is an interactive computer program for the analysis of linear models, primarily designed experiments. It features a syntax for entering, modifying, combining, and selecting data; graphical functions for plotting data and results; a grammar for specifying linear models; and numerical functions for fitting linear models to data. MacAnova was created in 1987 by Gary W. Oehlert of the Department of Applied Statistics, University of Minnesota to provide for the design and analysis of experiments on Macintosh microcomputers in a classroom environment. This manual refers to Version 1.0 of MacAnova.

Development of MacAnova was supported in part by the grant of an Apple Macintosh computer by the Minnemac Project at the University of Minnesota. Several colleagues, most notably Luke Tierney, Sandy Weisberg, and Christopher Bingham, have provided valuable suggestions and recommendations during the development of MacAnova. Jay Andersen (supported by an EDP grant from the University of Minnesota) provided the code for computing cumulative distribution functions, the plotting routines are adapted from GNUplot, and most Macintosh features are obtained from the TransSkel environment. The students in my Spring 1987 Statistics 5301 and 5163 classes survived (suffered through) early versions of MacAnova and provided several useful suggestions.

The syntax for MacAnova is modeled after S (Becker and Chambers 1984) but does not contain all the constructs in S. The parser, written in yacc (a compiler compiler available in UNIX), evolved from hoc, a desk calculator written also written in yacc (Kernighan and Pike 1984). The syntax for specifying linear models is taken from GLIM (Aitken *et al.* 1986).

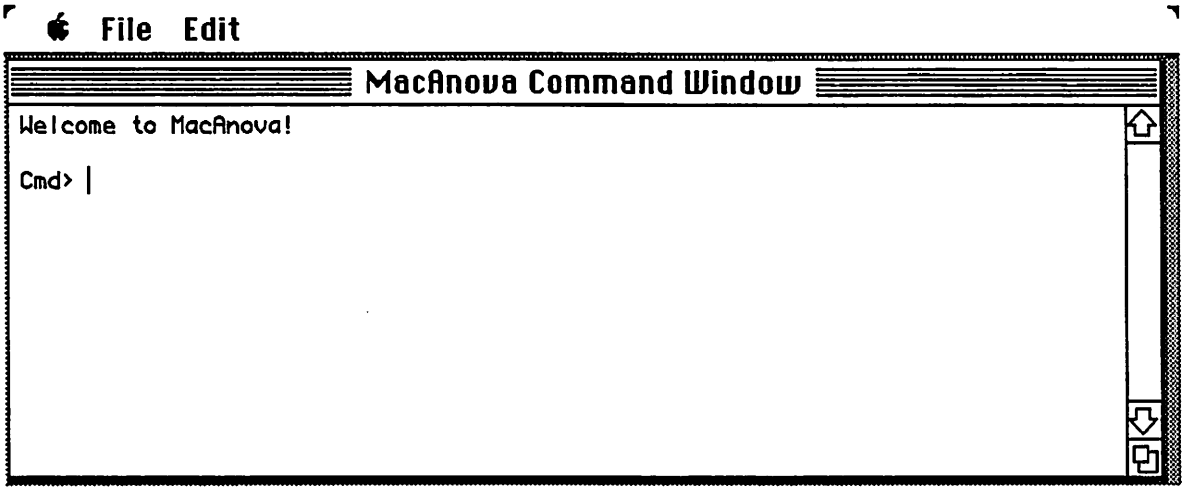
Disclaimer. Effort has been made to make MacAnova as error free as possible, but the program is rapidly evolving and has not yet been extensively tested. Neither the University of Minnesota, nor the Department of Applied Statistics, nor the author of this program claims any responsibility for errors that may arise from this program.

Comments on Version 1.0. MacAnova was originally intended to have much more design and much less analysis. However, I felt it was necessary to start out with the analysis aspects first so that the students using it would not be losing ground with respect to the data analysis capabilities available in prior computing environments. Future versions of MacAnova will try to remedy this imbalance.

The `anovacoefs()` function really needs to be revised so that the output format gives exactly the information wanted, rather than an awkward linear combination of it. (But if you think the explanation in Section 3 is complicated, you should try programming the general case!)

2. The Basics

Getting Started. Turn the computer on and insert a disk with MacAnova into the disk drive. If the disk icon is not open, open it. Now launch MacAnova by double clicking on the MacAnova icon. MacAnova will start up with a MacAnova Command Window as shown below. Commands to MacAnova are entered in two ways: typing text from the keyboard, and selecting an item from a menu. Text typed from the keyboard is entered into the command window at the insertion point (marked by a flashing vertical bar |), and backspacing erases text. MacAnova examines the text typed after the prompt (`Cmd>`) and processes the text as a command when a Return is typed. If you have a very long command, just keep typing (the typed characters will wrap around to the next line) and enter a Return when the command is done. You may enter more than one command on a single command line by separating the commands with semicolons.



Quitting. To exit MacAnova, select Quit from the File menu. This will stop MacAnova and return you to the desktop. Alternatively, select Quit & save output from the File menu. This will also stop MacAnova, but the contents of the command window are saved into a file called SPOOLFILE. (If SPOOLFILE already exists, the contents on the command window will be appended onto the end of the file.) This file may be printed or edited using MacWrite or MockWrite. When using either type of Quit, all of the variables in MacAnova's memory will disappear when you exit MacAnova. (But see `save()` and `restore()` in Section 4.)

Doing work. Between starting and stopping MacAnova, you probably want to get some work done. MacAnova commands are "functional". That means that commands are functions which have arguments (inputs) and return values (outputs). (Some functions have "side effects" -- related variables that are set up or deleted by the function. Residuals in regression and anova are examples of side effects.) The functions in MacAnova include statistical functions like `anova` (which does analysis of variance calculations) and `describe` (which does descriptive statistics like mean, variance, and quartiles), arthimetic functions like `log`, `exp`, `+`, `-`, etc., and housekeeping functions for reading and creating variables,

deleting variables and so forth. If the output from one function makes sense as the input to another, the functions may be composed (strung together). Thus `describe(log(x+3))` takes the data in `x`, adds three to it, takes the log of the sum, and then computes descriptive statistics on the logged values.

Variables. Data are stored in variables with user selected names up to 12 characters in length. Names must begin with a letter, but subsequent characters may be numbers as well as letters. Variable names are case sensitive, which means that `residuals`, `Residuals`, and `RESIDUALS` are all different names. It is a good idea to avoid names with capital letters, as MacAnova deletes and creates some variables with all capital names (eg, `RESIDUALS`).

There are several data types in MacAnova, but the most common are Real, Logical, and Character. Real data are just numbers like 2.4, -1, and 3.14159. Real data are allowed to take "missing values". When typed from the keyboard, the missing value code is a question mark, ?; when printed, a missing value prints as `MISSING`. Logical data indicate true or false and are symbolically represented by T and F. (There are system variables named T and F, which are logical variables with the values true and false respectively.) Character data are strings of characters. When typing character data on the keyboard, the character strings should be surrounded by double quotes, e.g. `"this is a character string"`. Note: any character is allowed inside a Character variable except a double quote, even a Return character. One common source of trouble is forgetting to add the closing double quote to a character variable, hitting Return, and expecting MacAnova to respond. Without the closing double quote, MacAnova is just waiting for more characters to put inside the string variable.

Real, Logical, and Character data can be organized as scalars (a single datum), vectors (a column of data), matrices, and more generally as arrays of up to 10 dimensions.

Syntax. MacAnova is a functional language and its syntax is functional. Variables may be processed via functions, and the output of functions may be combined if the outputs are compatible. Functions look like a function-name followed by a comma separated list of arguments in parentheses, e.g. `boxplot(x,y,z)`. Parentheses also indicate grouping or the order in which operations should be performed. Spaces between semantic units (like numbers or variable names) are ignored, but spaces within semantic units cause errors.

Arithmetic on Real data is simple. You may add, subtract, multiply , divide, and raise to a power in natural expressions by using the functions `+`, `-`, `*`, `/`, and `^` respectively. Addition and subtraction have lower precedence than multiplication and division which have lower precedence than raising to a power. Thus `(1+2)*3` is `9`, but `1+2*3` is `7`, and `2^3*2` is `16`, but `2^(3*2)` is `64`. The functions `abs`, `log`, `log10`, and `exp` provide absolute value, natural log, log base 10, and exponential functions. So, `abs(3-5)+log10(10)` is `3`.

Logical data are entered via T and F or generated via logical expressions on Real or Logical data. The logical expressions for Real data are equals `=` , not equals `!=` , less than `<` , greater than `>` , less than or equal to `<=` , and greater than or equal to `>=`. Thus `2=1`, `3 != 3`, `2 >= 5`, and `3 < 1` all have the value false, while `2=2`, `3 != 2`, and `3 > -1` all have the value true. Logical values may be combined using logical and `&` , logical or `|` , and logical not `!` . Thus, `F & T`, `F | F`, and `!T` all have the value false, while `F | T` and `!F` have the value true.

Assignment. Assigning a value to a variable is a function. In MacAnova, a left pointing arrow <- (the symbols "greater than" and "hyphen") indicates assignment:

```
Cmd> x<-6
```

makes x a scalar containing the real value 6. Any subsequent use of x will have the same effect as using the number 6; e.g. x+2 is 8. Character and logical data may be assigned in the same way:

```
Cmd> department <- "Applied Statistics"

Cmd> true <- T
```

In addition to making the assignment, assignment functions have a value equal to the value of the assigned variable. Among other things, this allows us to save intermediate values on the fly:

```
Cmd> y<-(x<-6)+8

Cmd> u<-w<-6+8
```

assigns 6 to x and 14 to y, u, and w.

If no assignment is made, the value of a command is printed:

```
Cmd> 3*5
(1)              15
```

results in 15 being printed, but

```
Cmd> x<-3*5

Cmd>
```

does not print anything. The command

```
Cmd> x
(1)              15
```

would then print 15.

The function `list()` will print a list of all currently defined variable names along with their types and dimensions. There are several built-in variables including T (true), F (false), E (the natural base), PI (π), and DEGPERRAD (the number of degress per radian). The function `delete()` will remove a list of variables. Thus

```
Cmd> delete(E,PI)
```

will remove the built-in variables E and PI from the variable list.

Vectors, etc. Vectors are formed using the function `cat` (short for concatenate). The expression `cat(1,2,3,5,5)` has a value which is a Real vector of length 4, and `z<-cat(1,7,4,9,6)` makes z a Real vector of length 5 with elements 1, 7, 4, 9, 6. If we typed `z2<-cat(z,z)`, then z2 would be a Real vector of length 10

containing two copies of the old z.

Elements of a vector may be selected in one of two ways. First, a specific list of elements may be given inside square brackets. Thus `z[4]` has the value 9, and `z[cat(1,1,2,2,4,5,3)]` is a Real vector with elements 1,1,7,7,9,6,4. Second, a logical vector of the same length as the source vector may be given in square brackets, and the value will be a vector with elements corresponding to the T elements in the logical vector. For `z` given above, `z>4` is a logical vector with elements F,T,F,T,T and `z[z>4]` is a vector with elements 7,9,6.

Lets repeat these examples and see what the MacAnova output looks like.

```
Cmd> z<-cat(1,7,4,9,6)

Cmd> z
(1)                 1           7           4           9           6

Cmd> cat(z,z)
(1)                 1           7           4           9           6
(6)                 1           7           4           9           6

Cmd> z[4]
(1)                 9

Cmd> z[cat(1,1,2,2,4,5,3)]
(1)                 1           1           7           7           9
(6)                 6           4

Cmd> z>4
(1)                 F           T           F           T           T

Cmd> z[z>4]
(1)                 7           9           6
```

Five values are printed per row when Real or Logical data are being printed. The number in parentheses on the left under the `Cmd>` prompt is the subscript or index of the first element printed in that row.

Matrices are formed using the function `matrix` which takes two arguments. The first is a vector of elements to go into the matrix, the second is the number of rows in the matrix. Data go into the matrix column by column.

```
Cmd> q<-matrix(cat(1,2,3,4,5,6),3)

Cmd> q
(1,1)               1           4
(2,1)               2           5
(3,1)               3           6
```

Matrices are printed row by row with up to five numbers printed per line. The numbers in parentheses at the beginning of each output line give the matrix subscripts of the first element of the line.

Elements of matrices may be accessed in an analogous manner to vectors, except that you must specify both rows and columns. Hence `q[3,1]` is the first column of the third row and `q[cat(1,2),2]` is the second column of the first and second rows. An empty row or column specification implies all rows or columns. Thus `q[3,]` is the third row. Elements of matrices retain both of their subscripts. The selected elements can be turned into an ordinary (one dimensional) vector by

using the function cat; cat(q[3,]) is a vector.

```
Cmd> q[3,1]
(1,1)                3

Cmd> q[,2]
(1,1)                4
(2,1)                5
(3,1)                6

Cmd> q[cat(1,2),2]
(1,1)                4
(2,1)                5

Cmd> q[3,]
(1,1)                3           6

Cmd> cat(q[3,])
(1)                3           6
```

Notice the difference in the subscripting above when cat is used.

MacAnova also allows multidimensional arrays. Arrays are formed by the function array() which takes two arguments. The first argument is the data to go into the array, and the second argument is a vector containing the dimensions for the array. The number of data values in the first argument must exactly equal the product of the dimensions given in the second argument. The value of the array() function is an array with the data entered into the array with the first subscript varying fastest. Arrays may be subscripted just like matrices, with the obvious exception that each subscript must be specified, not just the first two.

Structures. In addition to scalars, vectors, matrices, and arrays, variables may also be structures. A structure is a variable consisting of one or more named components. Each of the components may be of any type: real, character, logical, vector, scalar, matrix, even another structure. For example, we might wish to keep our data in some self-describing form. Our data set, lets call it trees, could be stored as a structure with three components called source, names, and data. The source component would be a character variable giving a general description of the data and its source, the data component would be a matrix of real numbers containing the actual data, and the names component would contain the names of the columns in the data component matrix.

Components of a structure are selected by using the dollar sign $. Thus, in our example, trees$data is a matrix containing the data set, trees$source is a character variable describing the data, and (trees$names)[2] is the name of the second column of the data set.

The MacAnova functions describe(), split(), and cellstats() return structures as their values. In addition, you may create your own structures through the makestr() function. makestr() takes arbitrarily many arguments and returns as its value a structure with the arguments of makestr() as components. The names of the components are the names of the arguments.

```
Cmd> source <- "made up data for example"

Cmd> names <- cat("species","dbh")

Cmd> data <- matrix(cat(1,1,1,2,2,2,5.6,4.5,8.9,7.3,9.9,11.3),6)
```

```
Cmd> trees <- makestr(source,names,data)

Cmd> trees
component: source
(1)
made up data for example
component: names
(1)
species
(2)
dbh
component: data
(1,1)                 1          5.6
(2,1)                 1          4.5
(3,1)                 1          8.9
(4,1)                 2          7.3
(5,1)                 2          9.9
(6,1)                 2         11.3

Cmd> trees$source
(1)
made up data for example

Cmd> (trees$data)[1,]
(1,1)                 1          5.6
```

Help. MacAnova has built-in help available through the `Help()` function. Giving the command

```
Cmd> Help()
```

will produce a list of the topics for which help is available in MacAnova. Selecting Help from the File menu is equivalent to giving the command `Help()`. To get help on a particular topic, use the `Help()` function with a character variable argument selected from the list of topics, for example, `Help("anova")`.

External data. It is frequently useful to read data into MacAnova from a text file, for example, a file created by MockWrite or MacWrite. MacAnova can read Real data from a text file, but not Character or Logical data. Missing data may be included in the text file as question marks ?. The function `read` will read all the numbers in a file and return a vector containing the values from the file. The argument of `read` is a character variable giving the name of the data file.

```
Cmd> x<-read("myfile")
```

will read all the data from a file named myfile and put the values into `x`.

MacAnova also has the ability to read columns of numbers from a file into separate variables. This is done via `readcols`. (`readcols` is a Macro, see Section 4.) The arguments to `readcols` are the filename, the number of columns in the data file, and the names of the variables into which the columns should be placed. All of these arguments must be character variables. Suppose that myfile has three columns and that we wish to read these columns into variables named `a`, `b`, and `c`. Then we could type

```
Cmd> readcols("myfile","3","a","b","c")
```

Note again that all arguments are entered as character variables.

Simple statistics. While MacAnova is oriented towards analysis of variance, it does provide some simple statistical functions as well. The function `describe()` computes the count (length), mean, variance, median, maximum, minimum, and upper and lower quartiles of its argument; `stemleaf()` produces a stem and leaf diagram of its argument; `sort()` sorts its argument into increasing order; and `rankits()` produces the rankits (normal scores) for its argument. The output of `describe()` is a structure, with components named `n`, `min`, `q1`, `median`, `q3`, `max`, `mean`, and `var`.

```
Cmd> x<-cat(2,6,4,5,8,2,3,4,1,7,8,4,3,6,5)

Cmd> describe(x)
component: n
(1)                15
component: min
(1)                 1
component: q1
(1)                 3
component: median
(1)                 4
component: q3
(1)                 6
component: max
(1)                 8
component: mean
(1)            4.5333
component: var
(1)            4.6952

Cmd> describe(x)$max
(1)                 8

Cmd> stemleaf(x)

   1              1 | 0
   3              2 | 00
   5              3 | 00
(  3)             4 | 000
   7              5 | 00
   5              6 | 00
   3              7 | 0
   2              8 | 00

Cmd> sort(x)
(1)                 1           2           2           3           3
(6)                 4           4           4           5           5
(11)                6           6           7           8           8

Cmd> rankits(x)
(1)          -0.94578     0.51499           0     0.33489       1.245
(6)            -1.245     -0.7137    -0.33489     -1.7394     0.94578
(11)           1.7394    -0.16512     -0.51499      0.7137     0.16512
```

There are four functions for Student's t based procedures. The functions

`tval()` and `t2val()` calculate one and two sample t statistics. The t statistic produced by `tval()` is for the null hypotheses that the mean of its argument is zero. If the null hypothesis mean is other than zero, merely subtract that null hypothesis value from the argument. The t statistic produced by `t2val()` is for the null hypothesis that the difference in mean of its two arguments is zero. If some other difference is hypothesized, subtract that value from the first argument. `t2val()` uses a pooled estimate of the standard deviation.

The functions `tint()` and `t2int()` produce one and two sample t confidence intervals. `tint()` takes two arguments, a data vector and a confidence level, and produces a t confidence interval of the requested coverage for the mean of the data vector. `t2int()` takes three arguments, two data vectors and a confidence level, and produces a t confidence interval of the requested coverage for the difference of the means of the data vectors (group one - group two) using a pooled estimate of the standard deviation.

```
Cmd> x<-cat(3,2,5,4,6,8,6,4,3,7,3)

Cmd> y<-cat(7,9,6,5,7,6,9,8)

Cmd> tval(x)
(1)           8.0437

Cmd> tval(x-5)
(1)         -0.63088

Cmd> t2val(x,y)
(1)          -3.0795

Cmd> tint(x,.95)
(1)           3.3521       5.9207

Cmd> t2int(x,y,.99)
(1)          -4.8308      -0.1465
```

P-values. MacAnova has six cumulative distribution functions (cdf) which can be used for computing p-values of tests. These functions are `cumnor()`, `cumstu()`, `cumchi()`, `cumF()`, `cumbin()`, and `cumpoi()` for the normal, Student's t, chi-square, F, binomial, and poisson distributions. The arguments to these functions are the value of interest followed by distribution parameters. These parameters are degrees of freedom for the t and chi-square, numerator and denominator degrees of freedom for the F, sample size and probability for the binomial, and mean for the poisson.

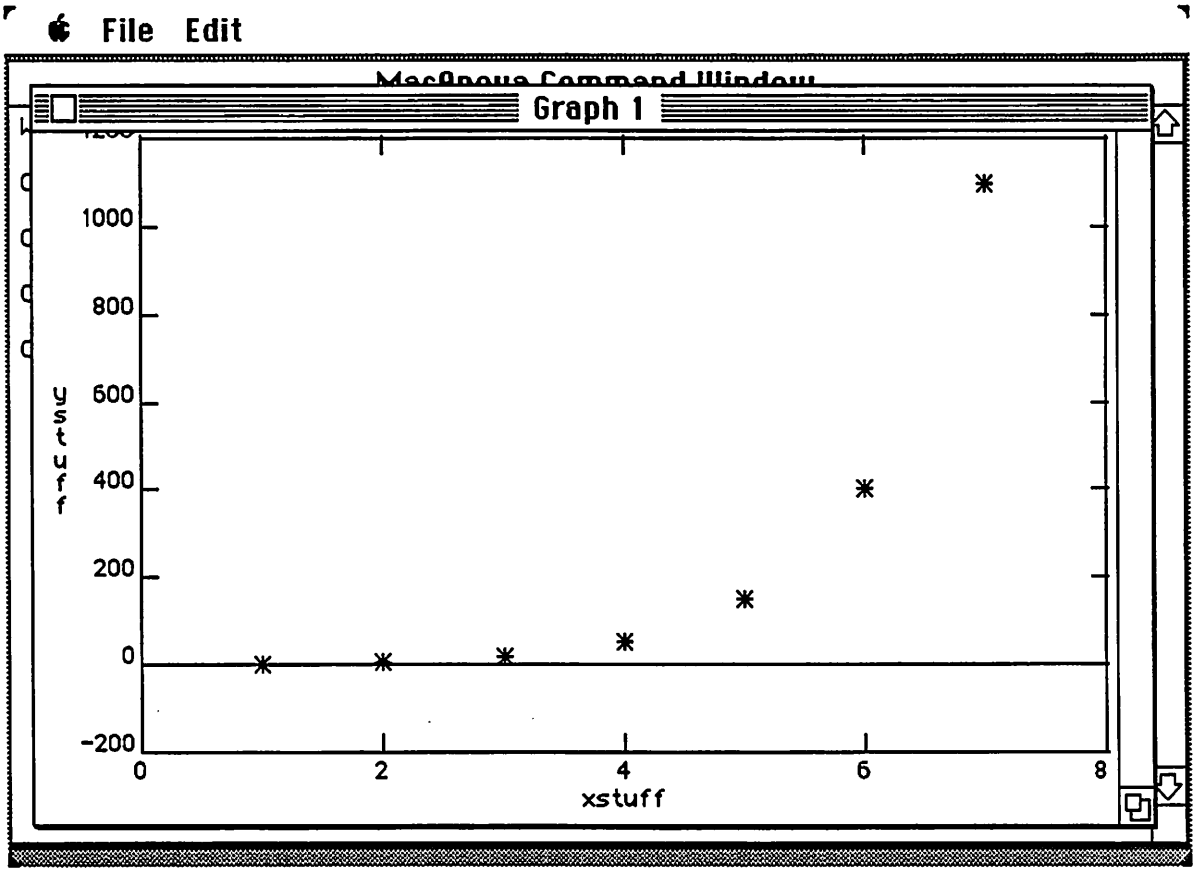
These cdf functions return the probability that a random variable would be less than or equal to the first argument; this is a lower-tail p-value. If you wish an upper-tail p-value (the usual case for the chi-square and F), take one minus the value of the cdf. For a two tailed z- or t-test, take two times the smaller of the upper and lower tail p-values. There is a macro `twotailt()` (for a discussion of macros, see Section 4) which will compute two tailed p-values for t-tests; `twotailt()` is illustrated below. Note that arguments to a macro must be character variables.

```
Cmd> cumnor(1.96)
(1)            0.975

Cmd> cumstu(2.1,10)
```

```
(1)          0.96896

Cmd> twotailt("2.1","10")
(1)         0.062077

Cmd> cumF(4,2,10)
(1)          0.94708
```

Plotting. MacAnova includes a plotting facility with makes use of several of the special features of the Macintosh computer. The basic plotting function is `plot()` which takes two arguments, the x and y values, which must be of the same length. `plot()` then produces a scatterplot the the y variable versus the x and puts this plot into its own graphics window as shown below.

```
Cmd> xstuff<-cat(1,2,3,4,5,6,7)

Cmd> ystuff<-exp(xstuff)

Cmd> plot(xstuff,ystuff)
```



The other plotting routine in MacAnova is `boxplot()`, which produces a Tukey box and whisker diagram of its argument. If more than one argument is given, parallel boxplots are produced. `boxplot()` can also accept a structure as an argument. In this case, parallel boxplots of the components of the argument are

produced. The function `split()` (see Section 5) can be used for splitting one data vector according to the values of a second.

Graphics windows are Macintosh windows in their own right. They may be moved by dragging with the mouse, resized by dragging the size box, and closed by clicking in the close box. (A graphics window may also be closed by selecting Close from the File menu.) Once you close a graphics window, it is gone until you reissue the plot command that created it. You do not need to close a graphics window to continue working in MacAnova. Selecting Command from the File menu will bring the Command Window to the front. You may now enter commands as usual. The graphics window is still on the screen, but covered up by the Command Window. Another approach is to select Hide from the File menu. This completely removes a graphics window from the screen, but remembers its definition. After selecting Command or Hide, a graphics window may be redrawn and moved to the front by selecting Graph 1 (or Graph 2 etc.) from the File menu. There are four graphics windows, labelled Graph 1 through Graph 4, which may be manipulated separately. The graphics windows are filled in order as you make plots. If all four graphics windows are in use, you may not draw more plots until one of the graphics windows is closed. (Closing in effect empties the window and allows another plot to use it.)

Graphics can be printed or saved in the following ways. If you are connected to a printer, then pressing the 4, command (clover), and Shift keys at the same time will print the contents of the active window. If you press the 4, Caps Lock, command, and Shift keys you will print the contents of the entire screen. You may save the contents of a graphics window in three steps. First, select Copy from the Edit menu. This will place the contents of the active graphics window in a temporary storage area known as the clipboard. Next select Scrapbook from the apple (🍎) menu. When the Scrapbook window appears on the screen, select Paste from the Edit menu. This will paste the clipboard (which has your graphics window) into the Scrapbook. The Scrapbook is permanently stored on the disk, so it will not go away when you quit MacAnova. You may use MacWrite to paste your graphics images from the Scrapbook into a document (using Copy and Paste from the Edit menu).

Other Macintosh features. You may edit the contents of the Command Window. Click the mouse to position the vertical bar anywhere you please. Typing will now insert characters and backspacing will delete characters from that location. Clicking and dragging the mouse will highlight a selection. You may delete the selection and copy it to the clipboard by selecting Cut from the Edit menu, and you may simply copy the selection to the clipboard by selecting Copy from the Edit menu. If you select Paste from the Edit menu, the current contents of the clipboard are entered into the Command Window at the location of the vertical bar (if any text is highlighted, Paste will write over the highlighted selection).

The most important use of these editing features is for correcting typing mistakes. Suppose you have entered a long command and made a mistake. Instead of retyping the command, edit the incorrect version and do a Copy/Paste operation to copy the corrected version to the current command line. Then click on the end of the line with the mouse and type return.

As you do work, the contents of the Command Window will scroll off the top of the screen. MacAnova saves these lines, and they can be reviewed (and edited if you wish) by using the scroll bar on the right side of the window. Clicking in the

up or down arrows scrolls up or down a couple of lines, clicking in the shaded region between the arrows and the square moves up or down a page, and dragging the square (called the thumb) moves directly to the indicated location in the text.

MacAnova saves about 20000 characters from the Command Window, but the larger the contents of the Command Window the more slowly MacAnova will run. For this reason, you may wish to periodically clean up your Command Window by deleting unwanted material.

3. Linear Models

Introduction. MacAnova performs standard multiple regression and analysis of variance (ANOVA) calculations. Regression and ANOVA are just two aspects of the same basic procedure (linear models), and the MacAnova interface to regression and ANOVA reflects this underlying similarity. The major difference between the `regress()` and `anova()` functions in MacAnova is the type of output printed, not the underlying principles or calculations.

Linear models (including regression and ANOVA) have a response variable (also called a dependent variable or target variable or Y) which is assumed to be a linear combination of one or more other variables (variously called the independent variables or predictor variables or carrier variables or X variables) plus random error. The random error is assumed to have zero mean and constant variance. To do testing, we also make the assumption that the errors follow a normal distribution. The least squares fit of a linear model to some data chooses the coefficients of the X variables so that the sum of squared errors between the Y data and the estimates based on the linear combination of the X variables is smallest.

The distinction between regression and ANOVA lies mainly in the type of X variables which appear (different types of X variables are appropriate in different problems). In ANOVA, the X variables are usually indicators for categories, for example, treatment groups in an experiment. Thus an X variable which took values 1,4,4,3,3,4,2 might indicate that responses 2,3, and 6 received treatment 4, response 7 received treatment 2 and so on. The X variable might just as have been coded as A,D,D,C,C,D,B. Interest in ANOVA focusses on the mean response in each group or combination of groups. In regression, the X variables usually indicate quantities so that a 4 in an X variable means "twice as much" as a 2 in the same X variable. Interest in regression often focusses on how much the mean response will change for a given change in one of the X variables. Variables which indicate grouping are called **factors**, and variables which do not indicate grouping are called **variates**. (This terminology comes from the GLIM package, Aitken et al. 1986.) In MacAnova, the X variables in ANOVA may be either factors or variates, while all X variables used in regression are assumed to be variates, regardless of their actual status.

In MacAnova, a variable is a variate unless it is specifically declared to be a factor by using the `factor()` function. For example, to create a variable `b` which is a factor assigning 6 responses alternately to two groups, you could give the commands

```
Cmd> b<-factor(cat(1,2,1,2,1,2))
```

or

```
Cmd> b<-cat(1,2,1,2,1,2)

Cmd> b<-factor(b)
```

Specifying models. The functions `regress()` and `anova()` each take a single Character string argument containing the model to be used in the regression or ANOVA. Models are specified similarly to GLIM. A regression model has the

form: response variable = X variable 1 + X variable 2 + ... + X variable k. Suppose that we want to fit a linear model relating strength of wood dowels to their diameter and density and that the data are in the variables `str`, `diam`, and `density`. We would use a regression model `"str = diam + density"`. Here diameter and density are both variates. The corresponding linear model is

$$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \varepsilon_i,$$

Note that an intercept is automatically included in the model. If we do not wish to have an intercept, we could use the model `"str = diam + density - 1"`; the `"-1"` part indicates that there should be no constant in the model. There is a limit of 10 variables in a regression model.

The models for the `anova()` function have the same basic form but are allowed to be more intricate. Suppose that `a`, `b`, and `c` are factors. Then the ANOVA model `"y = a"` is a one way classification or one way ANOVA model for the grouping indicated by a, and corresponds to the linear model

$$Y_{ij} = \mu + \alpha_i + \varepsilon_{ij}$$

The model `"y = a + b"` is a two way ANOVA with corresponding linear model

$$Y_{ij} = \mu + \alpha_i + \beta_j + \varepsilon_{ij}$$

The interaction of two factors is represented in a model by their dot product. The two way factorial model `"y = a + b + a.b"` corresponds to

$$Y_{ij} = \mu + \alpha_i + \beta_j + (\alpha\beta)_{ij} + \varepsilon_{ij}$$

Parentheses may be used to group terms into submodels, and the dot may be used to combine submodels as well as factors. For example, `(a+b).c` is equivalent to `a.c + b.c`, `(a+b).(c+d)` is equivalent to `a.c + b.c + a.d + b.d`, and so on. Star notation is a shortcut for producing factorial models. If `A` and `B` are submodels, the `A*B` is equivalent to `A + B + A.B`. Thus, `a*b*c` is equivalent to `a + b + a.b + c + a.c + b.c + a.b.c`. Slash notation is a convenient shortcut for nested models: `a/b` is equivalent to `a+a.b`, `a/b/c` is equivalent to `a + a.b + a.b.c`, and so on.

Terms may be removed from a model by using "subtraction". The model `a*b*c - a.b.c` is equivalent to `a + b + a.b + c + a.c + b.c`. A more extensive form of removal is available with "minus star " subtraction, which removes not only the literal term as in "subtraction" but also any term which contains the literal term. Thus, `a*b*c -* a.b` is equivalent to `a + b + c + a.c + b.c`.

Variates may be included in ANOVA models with the sole restriction that variates cannot interact (be dotted with) other variates. Variates may be dotted with factors or interactions of factors. A variate in an ANOVA model is usually referred to as a covariate, and the computations for a variate amount to computing the regression coefficient for the covariate. If a variate is dotted with a factor, a separate regression coefficient for the covariate is computed in each group of the factor. Suppose `grp` is a factor and `x` is a variate. Then `"y = x"` is to a simple regression model, `"y = grp"` is a one way classification model, `"y = a + x"` is a model with common slope and separate intercepts for each group, `"y = a.x"` is a model with a common intercept and separate slopes for each group, and `"y = a + a.x"` is a model with separate lines for each group.

MacAnova enforces a limit of 10 distinct variables in an ANOVA model, but

these variables may be combined via interaction (dotting) for up to 1024 different terms in the model.

Error terms. All sums of squares and degrees of freedom not contained in model terms are combined into a term called ERROR1. Some models, such as split plot models, have more than one error term. In MacAnova, a term may be relabelled as an error term by enclosing it in `E( )`. The only effect of this is the relabelling of the term in the output, and the renumbering of other error terms. Sums of squares are computed as usual. For example, `"y = a + b + E(c)"` will have terms a, b, ERROR1 (equal to c) and ERROR2.

Output. The `regress()` and `anova()` functions return no values. Instead, they print out standard summaries of their actions and set some side-effect variables. The output for the `regress()` function includes the coefficients, their standard errors and t statistics, the R-squared, mean square and degrees of freedom for error, and an ANOVA table for the regression. The output of the `anova()` function is just the ANOVA table. Examples of output follow.

```
Cmd> x1<-cat(1,2,3,4,5,6,7,8,9,10)

Cmd> x2<-cat(1,2,3,1,2,3,1,2,3,4)

Cmd> y<-cat(2,4,5,4,3,6,7,4,5,8)

Cmd> regress("y=x1+x2")
                          Coef          sd           t
CONSTANT                 1.9714      1.1663      1.6904
x1                      0.32698     0.18455      1.7718
x2                      0.46825     0.54101     0.86552

N: 10   MSE:      2.0116  DF: 7  R^2:     0.52429

Sequential sums of squares

                                          DF          SS          MS
CONSTANT                                   1       230.4       230.4
x1                                         1      14.012      14.012
x2                                         1      1.5069      1.5069
ERROR1                                     7      14.081      2.0116

Cmd> a<-factor(cat(1,1,1,1,2,2,2,2))

Cmd> b<-factor(cat(1,2,3,4,1,2,3,4))

Cmd> z<-cat(2,3,4,3,5,6,4,5)

Cmd> anova("z=a+b")
                                          DF          SS          MS
CONSTANT                                   1         128         128
a                                          1           8           8
b                                          3           1     0.33333
ERROR1                                     3           3           1
```

Both `regress()` and `anova()` set the following side-effect variables: STRMODEL a character variable containing the model used, TERMNAMES a character vector containing the names of the terms in the model, SS a real vector containing the

sums of squares for each term in the model, DF a real vector containing the degrees of freedom for each term in the model, RESIDUALS a real vector containing the residuals from the model, and HII a real vector containing the diagonal of the hat matrix. In addition, `regress()` also creates REGCOEF a real vector containing the regression coefficients and XTXINV a real matrix containing X transpose X inverse, where X is the matrix form of the independent variables. These side-effect variables are deleted at the beginning of each call to `regress()` or `anova()`, so any side-effect variables that you desire to keep should be copied to a more permanent location prior to further work.

Balance. A data set is balanced for a given model if the model contains only factors and if each factor-level combination occurs an equal number of times in the data set. When a data set is balanced for a model, the order of the terms in the model does not affect the sums of squares for the terms.

Any model which is not balanced is called unbalanced. The sums of squares for terms in an unbalanced model **may** depend on the order of the terms in the model. When a data set is unbalanced for a model, MacAnova computes sequential sums of squares. Sequential sums of squares are defined to be the improvement in model fit (as measured by SSE) by adding the current term to the model containing only the preceding terms.

Analysis of unbalanced data can be tricky, and null hypotheses tested may depend on where a term occurs in a model. Several different models with the terms in different orders may need to be computed in order to get all necessary sums of squares.

Parameterization and degrees of freedom . (Advanced topic.) MacAnova uses a variant of the classical (Scheffé) parameterization. Consider factors B and C with b and c levels respectively. Factor B is parameterized by b columns: a column of ones followed by b-1 columns, the ith of which has ones in the group i positions, minus ones in the group b positions, and zeros elsewhere. Factor C is parameterized similarly. The B.C interaction is parameterized by the bc columns which are the pairwise products of the columns in the B and C parameterizations. (Three way interactions are three way products and so on.) A variate X is parameterized by X itself, and an interaction such as B.C.X is parameterized by the product of X and the columns parameterizing B.C.

The result of this parameterization is that the models fit by MacAnova are strictly hierarchical. It is impossible to fit the (b-1)(c-1) dimensional B.C interaction without fitting the main effects of B and C, either explicitly with B and C terms or implicitly through the parameterization used. Hence the degrees of freedom for interaction terms (and even main effects if you consider removing the constant) depend on which terms **precede** them in the model. The following table illustrates the degrees of freedom for various terms and models (assuming complete data):

	df		
Model	B	C	B.C
B + C + B.C	b-1	c-1	(b-1)(c-1)
B + B.C + C	b-1	0	b(c-1)
B.C + B + C	0	0	bc - 1
B.C	0	0	bc - 1
B.C - 1	0	0	bc

B	b-1	0	0
B-1	b	0	0
B+B.C-1	b	0	b(c-1)

For unbalanced data, the ANOVA is calculated by setting up an X matrix as shown above and doing a regression, while for balanced data, the ANOVA is calculated by sweeping out cell means. The functions `anovacoefs()` and `cellstats()` described below can be used to recover the regression coefficients, cell means, and treatment effects.

Contrasts. A contrast in a factor is a linear combination of the factor effects where the coefficients in the linear combination sum to zero. A sum of squares may be computed for a contrast suitable for testing the null hypothesis that the contrast linear combination of the true (not estimated) factor effects is zero; the MacAnova function `contrast()` computes this sum of squares. `contrast()` takes two arguments. The first is a character variable giving the name of one of the factors in the current ANOVA model. This name must be given exactly as it appears in the ANOVA output with no leading or trailing blanks. The second argument is the vector of contrast coefficients. This vector must have as many elements as there are categories for the factor chosen, and the elements must sum to zero.

If the data are unbalanced, the sum of squares for a contrast depend on what model terms precede the contrast in the computations. MacAnova assumes that a contrast occurs in a position equivalent to the first position at which the factor in question occurs. For example, in a contrast on factor `d` with model "`b.c + d.c + e`", the term `b.c` would precede the contrast, and the contrast sum of squares would be adjusted for `b.c`, but not for `e`.

The following example of contrast is for the sample ANOVA given above:

```
Cmd> contrast("b",cat(-2,1,4,-3))
(1)             0.15
```

Cell by cell statistics. Once an ANOVA model is active, the function `cellstats()` will compute cell by cell means, variances, and counts for user specified terms. The argument to `cellstats()` is a character variable containing a term in the factors of the current ANOVA model. Note, this term need not be present in the model, but it must be made up of factors from the model. The output of `cellstats()` is a structure with components named mean, var, and count. `cellstats()` works with either balanced or unbalanced data, and the result does not depend on the order of the terms in the model.

The following is based on the example ANOVA model given above. The first example gives statistics for the 4 cells descibed by the `b` term, while the second gives statistics for the 8 (2x4) cells in the `a.b` term.

```
Cmd> cellstats("b")
component: mean
(1)             3.5         4.5          4          4
component: var
(1)             4.5         4.5          0          2
component: count
(1)               2           2          2          2
```

```
Cmd> cellstats("a.b")
component: mean
(1,1)                2          3          4          3
(2,1)                5          6          4          5
component: var
(1,1)                0          0          0          0
(2,1)                0          0          0          0
component: count
(1,1)                1          1          1          1
(2,1)                1          1          1          1
```

Estimated effects. It is generally not adequate to stop after computing an ANOVA table; we usually want to see the estimated effects for the various terms. This can be done in MacAnova, but the process can be a bit awkward. Lets begin with the balanced case. There are four steps. First, compute the ANOVA table for the submodel that consists of terms completely contained in the term of interest. For example, suppose our model is `"y=a+b+a.c+a.c.d"`. If `a.c.d` is the term of interest, then `a+a.c` is the appropriate submodel; if `a.c` is the term of interest, then `a` is the appropriate submodel; and if `a` or `b` is the term of interest, `1` (the constant) is the appropriate submodel. The second step is to save the residuals from the ANOVA into another variable, say `res`. Third, compute the ANOVA table for the full model, but use the previous residuals `res` as the Y variable. Finally, use `cellstats()` with the term of interest to get cell statistics. The cell means are the ANOVA effects for that term. This process is illustrated in the following example, where we are interested in the interaction effects in the two way factorial model.

```
Cmd> anova("z=a*b")
                                        DF          SS          MS
CONSTANT                                 1         128         128
a                                        1           8           8
b                                        3           1     0.33333
a.b                                      3           3           1
ERROR1                                   0           0           0

Cmd> anova("z=a+b")
                                        DF          SS          MS
CONSTANT                                 1         128         128
a                                        1           8           8
b                                        3           1     0.33333
ERROR1                                   3           3           1

Cmd> res<-RESIDUALS

Cmd> anova("res=a*b")
                                        DF          SS          MS
CONSTANT                                 1           0           0
a                                        1           0           0
b                                        3           0           0
a.b                                      3           3           1
ERROR1                                   0           0           0

Cmd> cellstats("a.b")$mean
(1,1)             -0.5        -0.5           1           0
(2,1)              0.5         0.5          -1           0
```

Note here that the cell means sum to zero across both rows and columns as pure interaction effects should.

For unbalanced data, the `anovacoefs()` function must be used. `anovacoefs()` prints the regression coefficients for the columns in the parameterization of the term as described above. These coefficients must be recombined to get the usual ANOVA effects. For a main effect at k levels, the parameterization is (grand mean, effect of level 1, effect of level 2, ... , effect of level k-1). The grand mean is in the model prior to any term unless it has been specifically removed, so the grand mean in the main effect is colinear with the first grand mean and has `MISSING` for its coefficient. The effects of levels 1 through k-1 may be read from the output, and the effect of level k is obtained through the rule that the sum of the effects for the k levels must be zero. Thus the effect of level k is minus the sum of the effects of the first k-1 levels. If the grand mean was removed from the model and the term of interest is the first term in the model, then the grand mean in the main effect parameterization will have a non`MISSING` coefficient that should be added to the effects of the k levels. Consider the following example.

```
Cmd> resp <- cat(1,2,3,4,5,6,7)

Cmd> rows <- factor(cat(1,1,2,2,3,3,3))

Cmd> cols <- factor(cat(1,2,1,2,1,2,2))

Cmd> anova("resp=rows*cols")
WARNING: unbalanced data so SS are sequential
                                          DF          SS          MS
CONSTANT                                   1         112         112
rows                                       2          25        12.5
cols                                       1         2.4         2.4
rows.cols                                  2         0.1        0.05
ERROR1                                     1         0.5         0.5

Cmd> anovacoefs("rows")
(1)          MISSING     -2.0833    -0.083333

Cmd> anovacoefs("CONSTANT")
(1)           3.5833
```

Up to here, the grand mean (CONSTANT) is in the model, so the first element in the rows parameterization is MISSING. The effect for the third row is -(-2.0833-0.0833) = 2.16666. Now lets try it without the grand mean.

```
Cmd> anova("resp=rows*cols-1")
WARNING: unbalanced data so SS are sequential
                                          DF          SS          MS
rows                                       3         137      45.667
cols                                       1         2.4         2.4
rows.cols                                  2         0.1        0.05
ERROR1                                     1         0.5         0.5

Cmd> anovacoefs("rows")
(1)           3.5833     -2.0833    -0.083333
```

`rows` now has three degrees of freedom, and this is reflected in the fact that the grand mean part of the parameterization is not `MISSING` and is equal to the old

CONSTANT coefficient.

Consider now an interaction term, say rows.cols. Because of the product parameterization used for interactions, the first column of the interaction coefficients corresponds to the parameterization for rows, and the first row of the interaction corresponds to cols. Since both rows and cols precede the interaction in the model, the corresponding parts of the interaction are colinear with the preceding terms and have MISSING coefficients.

```
Cmd> anovacoefs("rows.cols")
(1,1)          MISSING     MISSING
(2,1)          MISSING    0.083333
(3,1)          MISSING    0.083333
```

The 0.08333 values are the interaction effects for the first two rows of the first column. The last row and column of interaction effects may be calculated using the rule that all rows and columns must add to zero. Thus the effects are

0.0833	-0.0833
0.0833	-0.0833
-0.1666	0.1666

Consider now the case where not all the terms lower in the hierarchy are included in the model. In this case, some of the first row or first column effects will be present and must be added into the interaction effects.

```
Cmd> anova("resp=rows+rows.cols")
WARNING: unbalanced data so SS are sequential
                                          DF          SS          MS
CONSTANT                                   1         112         112
rows                                       2          25        12.5
rows.cols                                  3         2.5     0.83333
ERROR1                                     1         0.5         0.5

Cmd> anovacoefs("rows.cols")
(1,1)          MISSING    -0.58333
(2,1)          MISSING    0.083333
(3,1)          MISSING    0.083333
```

In this example the column effects have been subsumed into the interaction effect. Again, the column effects must sum to zero, so they are -0.58333 and 0.58333. To get the standard form for the interaction effect, the column effects must be added to the 3x2 table of interaction effects calculated above. Thus

-0.5	0.5
-0.5	0.5
-0.75	0.75

Since row effects were already in the model, rows of the interaction effects sum to zero. Column effects were not already in the model but are included in the interaction term, so columns of the interaction effects do not sum to zero.

The general rule for computing interaction effects from anovacoefs() output is to first find all the highest order pure interaction effects in the table by using the rule that marginal sums must be zero. Then all marginal effects must be inspected. If any are nonMISSING, they must be added across the table of interaction effects. This works in any number of dimensions.

4. Additional Features

Macros. A macro is a collection of commands grouped together to make it easy to give all the commands at once. In addition, macros take arguments that can be substituted into the body of the commands. This makes it easy to use a different set of variables each time a macro is used.

A macro looks like a function call: the macro name followed by a paranthesized list of arguments. The arguments to macros must be character variables. When MacAnova encounters a macro in the command line, it expands out the list of commands stored in the macro and substitutes each of the character arguments for its place holder in the macro definition ($1 for argument 1, $2 for argument 2, and so on). If the macro `mycat` were `cat($1,$2)` and you typed `mycat("12","35")`, MacAnova would expand `mycat` out, filling $1 and $2 to get `cat(12,35)`. Processing then continues as if you had typed `cat(12,35)` to begin with.

MacAnova has three builtin macros: `readcols()`, `twotailt()`, and `boxcox()`. `readcols()` (discussed in Section 2) allows you to read data from a file by columns into variables in MacAnova. The arguments are filename, number of columns, and column names (no more than 10). To read the data in file myfile into columns `a`, `b`, and `c`, use the command `readcols("myfile","3","a","b","c")`. `twotailt()` was also discussed in Section 2. It is a macro for computing two tailed p-values for the Student's t-test. The first argument is the t-value and the second argument is the degrees of freedom. `boxcox()` is a macro for computing the Box-Cox transformation of a set of data. It takes two arguments, the data and the Box-Cox parameter.

To see the contents of a macro, use the `print()` function. The argument to `print()` is a character string containing the name of the macro or variable to be printed. (`print()` works with any variable, but is most useful for macros.) For example,

```
Cmd> print("twotailt")
(1)
TMP<-cumstu($1,$2)
if( TMP < 0.5 ) {
     PVAL <- 2*TMP
}
if( TMP > 0.5) {
     PVAL <- 2*(1-TMP)
}
delete(TMP)
PVAL
```

The function macro() may be used to create new macros. macro() takes one argument, a character variable containing the commands to be grouped. The character variable may contain place holders $1, $2, etc. for argument substitution when the macro is expanded.

```
Cmd> tmp <- "exp(describe(log($1))$mean)"

Cmd> geomean <- macro(tmp)

Cmd> print("geomean")
(1)
```

```
exp(describe(log($1))$mean)

Cmd> y<-cat(1,2,3,4,5)

Cmd> geomean("y")
(1)             2.6052
```

Save and restore. All of MacAnova's variables, macros, and so forth are kept in dynamic memory, not on the disk. That means that when you exit MacAnova, all of those variables disappear into the ether. This may be problem if you have spent a long time building up a set of variables. The solution is `save()` and `restore()`. `save()` takes as an argument a character variable containing the name of a file into which the MacAnova variables should be saved. `restore()` is the inverse of `save()`. `restore()` also takes a character argument giving a file name. If that file is a MacAnova save file, `restore()` will replace the current set of variables with those in the restore file.

```
Cmd> x<-cat(1,2,3)

Cmd> save("junk")

Cmd> delete(x)

Cmd> x
UNDEFINED

Cmd> restore("junk")

Cmd> x
(1)                 1           2           3
```

Compound lines and conditional execution. A compound line is a left brace "`{`", followed by one or more input lines, followed by a right brace "`}`" on a line by itself. Compound lines are used for grouping several statements together. Just as an individual command line has a value (which might or might not be printed depending on whether or not there was an assignment), a compound line has a value. The value of a compound line is the value of the last line in the compound line. No prompt is printed for the additional lines of a compound line, and no intermediate values are printed unless specifically requested via `print()`. Thus,

```
Cmd> {
x<-6
y<-3
x
}
(1)                 6

Cmd> {
x

}

Cmd> {
print("x")
print("y")
```

```
}
(1)                 6
(1)                 3
```

Note that a blank line has no value and thus prints nothing.

The principle use of compound lines is in conditionals. A conditional takes the form

```
Cmd> if(logical) compound statement
```

MacAnova evaluates the logical variable. If it is true, the compound statement is executed. If it is false, the compound statement is skipped.

```
Cmd> if(2>1) {
5
}
(1)                 5

Cmd> if(2<1) {
3
}
```

Batch files. Batch files are external files containing one or more semicolon separated commands on a single command line. The commands in the file can be executed by giving the command

```
Cmd> batch("filename")
```

where `filename` is the name of the batch file. The only advantage of batch files over macros is that the contents of batch files can easily be changed using a desk accessory like MockWrite.

5. Functions

anova(Model) produces an ANOVA table for the linear model in the character variable Model.

Models are specified as response = sum of terms , where terms are of the form name or name.name. ... .name . Variables in terms may be either factors or variates, though only one variate may appear in a single term. The shortcut formula model1*model2 expands to model1 + model2 + model1.model2 , and the formula model1/model2 expands to model1 + model1.model2. There is a maximum of 10 different X variables in a model.

Sums of squares are computed sequentially for nonbalanced designs. The variables RESIDUALS, HII, DF, SS, TERMNAMES, and STRMODEL are created as side effects of anova.

Example: anova("y= a + b + a.b")

Assuming a and b are factors, this will produce a two way analysis of variance with interaction for the response in y.

See also anovacoefs(), cellstats(), contrast(), factor(), and regress().

anovacoefs(Term) extracts the model effects (regression coefficients) in an unbalanced ANOVA for the term specified in the character variable Term.

A factor with k levels is parameterized by a constant and k-1 columns for the first k-1 group effects. The kth effect is minus the sum of the first k-1. Interaction terms are parameterized as products of the main effects terms yielding arrays of coefficients with the leftmost subscript varying fastest.

Example: anova("y= a + b + a.b")
anovacoefs("a.b")
This will produce the a by b interaction components.

array(datav,dimensionv) takes the data in datav and makes an array containing that data with dimensions as given in the vector dimensionv.

There must be exactly as many data in datav as the product of the dimensions in dimensionv. The data are entered with the leftmost dimension varying fastest and the rightmost varying slowest. The data in datav may be of any type.

See also matrix.

batch(Name) executes the commands in the file with name given in the character variable Name.

The commands in the file must all be on one line (separated by semicolons) or be a compound line.

boxcox("var","Pow") is a macro which computes the Box-Cox transformation of the data in var.

Recall that the arguments to a macro must be character variables, for example, boxcox("x","0.5").

See also macro and transformations.

boxplot(var1, var2, ... , vark) or **boxplot(Struc)** produces parallel Tukey boxplots for the variables var1 through vark or the components of structure Struc.

See also split.

cat(a,b,c,...,k) concatenates the data in variables a,b,...,k into a single vector containing all the data. The arguments must all be of the same type.

cellstats(Term) computes cell statistics of the multiway layout indicated by the term in the charater variable Term.

The term must be made of factors in the current ANOVA model. The value is a structure with three components: mean, var, and count. Each component is an array with as many dimensions as there are factors in the term. The dimensions of the arrays correspond to the factors in the order the factors first appear in the ANOVA model, not in the order they are specified in Term.

Example: anova("y=a+b+c+b.c")
cellstats("c.b")
gives cell statistics for the b.c term.

contrast(Factor,vectorc) computes the sum of squares for the contrast in the levels of the factor given in the character variable Factor with contrast coefficients given in vectorc.

Contrast will only work if there is an active ANOVA model. If the model is unbalanced, the contrast sum of squares will be corrected for all terms that precede the first appearance of Factor in the model.

Example: anova("y=a+b+a.b")
contrast("b",cat(-1,1,0)) (assuming b has 3 levels)

cumbin(Val,N,P) computes the probability that a binomial random variable with N trials at probability P would be less than or equal to Val.

N must be positive but less than 1000 and P must be between zero and one.

cumchi(Val,df) computes the probability that a chi squared random variable with df degrees of freedom would be less than Val.

The degrees of freedom must be positive. Upper tail areas of chi squared can be computed as one minus cumchi.

cumF(Val,df1,df2) computes the probability that an F random variable with df1 and df2 degrees of freedom would be less than Val.

The degrees of freedom must be positive. Upper tail areas of F can be computed as one minus cumF.

cumnor(Val) computes the probability that a standard normal (mean zero, variance one) random variable would be less than Val.

Upper tail areas of the normal can be computed as one minus cumnor.

cumpoi(Val,lambda) computes the probability that a poisson random variable with mean lambda would be less than or equal to Val.

lambda must be positive.

cumstu(Val,df) computes the probability that a t random variable with df degrees of freedom would be less than Val.

The degrees of freedom must be positive. Upper tail areas of t can be computed as one minus cumstu. Two tailed p-values for the t can be computed with the macro twotailt.

See also twotailt.

delete(var1,var2,...,vark) deletes the variables given as arguments and frees the memory they used for other purposes.

See also list.

describe(Data) computes descriptive statistics for the vector of numbers given in Data.

The value returned is a structure with eight components: n, min, q1, median, q3, max, mean, and var, where q1 and q3 are the lower and upper quartiles.

factor(x) creates a vector with contents identical to x except that the new vector is marked as a factor.

The contents of x must be positive integers. The number of classes in the factor will be the largest integer in x. Thus both the x vectors 1,2,4 and 1,2,3,4 will produce factors with four classes, though only three of the classes are present in the first example.

A factor indicates classes or categories, while a variate indicates quantities.

Help(Topic) prints helpful information about the topic given in the character variable Topic.

For example, Help("Help") will produce this text. Help() with no argument will

print a list of available help topics.

list() lists the name, type, and dimensions of all variables currently active in computer memory.

See also delete.

macro(Text) creates a macro from the commands contained in the character variable Text.

In the body of Text, references to $1, $2, and so on will be replaced (as character strings) by the first, second, and so on arguments when the macro is called. Arguments in a macro call are ALWAYS character strings.

Example: mymacro<-macro("cat($1,$2)")
mymacro("1","4")
has as output the vector 1,4.

makestr(var1,var2,....,vark) creates a structure with components named var1, var2, ..., vark. The values of the components are equal to var1, etc.

matrix(datav,rowdim) makes a matrix with rowdim rows containing the data in datav.

rowdim must exactly divide the length of the data vector. The data are entered with the row dimension (leftmost subscript) changing fastest.

See also array().

plot(x,y) makes a scatterplot of the data in x and y.

On a Macintosh, the plot may be printed by pressing Command-shift-4, or saved to the clipboard by selecting Copy from the Edit menu.

print(Name) prints the variable whose name is given in the character variable Var.

This function is most useful inside macros and compound lines and for printing macros

rankits(x) computes rankits (normal scores) for the data in the vector x.

plot(rankits(x),x) produces a rankit plot of x.

read("fname") reads numerical data from the file with name fname and creates a vector containing the data.

Missing values (represented by a question mark ?) are allowed in the file.

See also readcols.

readcols("fname","ncols","var1name","var2name",...,"varncolsname") (a macro) reads numerical data from the file fname and puts the data into ncols variables with names given in the third and following arguments.

The number of data values in the file must be divisible by ncols, ncols must be less than or equal to 10, and there must be ncols variable names given.

See also macro, read.

regress(Model) fits the regression model given in the character variable Model.

Model is of the form response = variable 1 + variable 2 + ... + variable k, and all X variables are assumed to be variates. Output includes coefficients, standard errors, and an ANOVA table. The variables RESIDUALS, HII, SS, DF, TERMNAMES, STRMODEL, REGCOEF, and XTXINV are produced as side effects. Regression through the origin is specified by including a "-1" in the model. No more than 10 X variables are allowed.

Example: regress("y = x1 + x2 + x3")

See also anova.

restore(Name) deletes all current variables and restores the variables saved into the file with name given in the character variable Name.

The command save(Name) must have been given prior using restore(Name).

See also save.

save(Name) saves the current set of variables into a file with name given in the character variable Name.

See also restore.

sort(x) sorts the data in x into ascending order.

split(Data,Factor) splits the values in Data into separate components of a structure according to the values of Factor (which must be a factor).

For example, split(cat(1,2,3,4),factor(cat(1,2,1,2))) will produce a structure with two components. The first, named comp1, is a vector containing 1 and 3; the second, named comp2, is a vector containing 2 and 4.

stemleaf(var) produces a stem and leaf diagram of the data in var.

t2int(var1,var2,Cover) computes a (two sided) t confidence interval with coverage rate Cover for the mean of the data in var1 minus the mean of the data in var2.

Cover must be between zero and one. The value is a vector of length two giving the lower and upper endpoints of the interval.

t2int computes a pooled estimate of the standard error of the difference.

See also t2val, tval, and t2val.

t2val(var1,var2) computes the two sample Student's t statistic for testing the null hypothesis that the data in var1 and var2 have the same mean.

If the null hypothesis difference (mean1 - mean2) is something other than zero, say delta, then t2val(var1-delta,var2) will produce the correct t value for that null hypothesis. For example, if delta is 3, use t2val(var1-3,var2).

P-values may be computed using the function cumstu or the macro twotailt.

t2val computes a pooled estimate of the standard error of the difference.

See also tval, tint, t2int, cumstu, and twotailt.

tint(var,Cover) computes a (two sided) t confidence interval with coverage rate Cover for the mean of the data in var.

Cover must be between zero and one. The value is a vector of length two giving the lower and upper endpoints of the interval.

See also tval, t2val, and t2int.

transformations log(x), log10(x), exp(x), and abs(x) produce the natural log, log base 10, exponential, and absolute value of the data in x.

tval(var) computes the Student's t statistic for testing the null hypothesis that the data in var have mean zero.

If the null hypothesis mean is something other than zero, say delta, then tval(var-delta) will produce the correct t value for that null hypothesis. For example, if delta is 3, use tval(var-3).

P-values may be computed using the function cumstu or the macro twotailt.

See also t2val, tint, t2int, cumstu, and twotailt.

twotailt("T","df") is a macro which computes the two tailed p-value for a t value of T with df degrees of freedom.

Recall that the arguments to a macro must be character variables, for example, twotailt("2.5","21").

See also macro and cumstu.

6. Examples

Introduction. This section illustrates some of the commands and possible models that can be used in MacAnova. It is **not** a complete inventory of models and the examples should **not** be construed as complete or recommended analyses. Many procedures (residual plots, for example) should be used in nearly every example, but are only used once or twice in this section to save space. The sample data sets are taken from Montgomery (1984) (M) or Devore and Peck (1986) (DP). Comments are entered in this typeface.

Simple descriptive statistics. This is Example 19, page 95 of DP. The rem variable is the REM period latency and secretion indicates normal (1) or hyper (2) cortisol secretion.

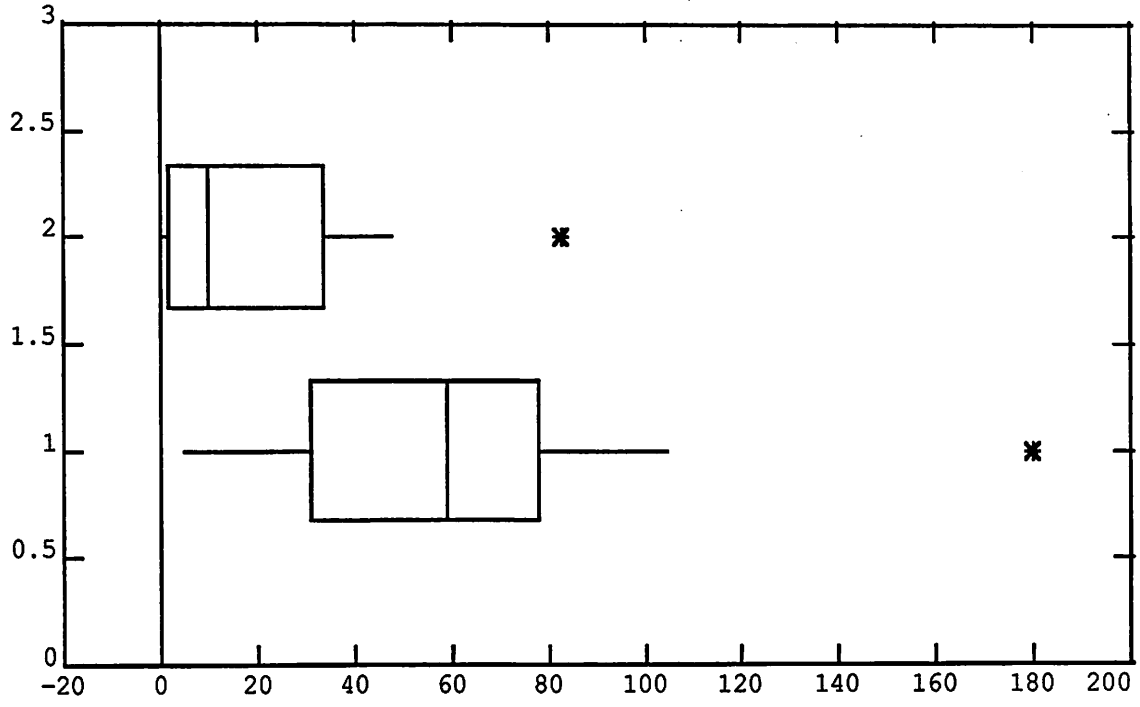
```
Cmd> rem <- cat(.5,1,2.4,5,15,19,48,83,5,5.5,6.7,13.5,31,40,47,47,59,62,68,72,
78,84,89,105,180)
```

Enter the data

```
Cmd> secretion <-
factor(cat(2,2,2,2,2,2,2,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1))
```

```
Cmd> boxplot(split(rem,secretion))
```

Box plots separately by group




```
Cmd> stemleaf(rem[secretion = 2])

  4              0 | 1125
  4             10 | 59
  2             20 |
  2             30 |
  2             40 | 8
```

Could do stem and leaf (hyper secretion)

```
High:             83

Cmd> describe(rem[secretion = 1])
component: n
(1)               17
component: min
(1)                5
component: q1
(1)               31
component: median
(1)               59
component: q3
(1)               78
component: max
(1)              180
component: mean
(1)           58.394
component: var
(1)             1932

Cmd> t2val(rem[secretion = 1],rem[secretion = 2])
(1)           2.1333

Cmd> 1-cumstu(2.1333,23)
(1)         0.021896
```

or get simple descriptive statistics (normal secretion)

Two sample t test

Upper tail p-value.

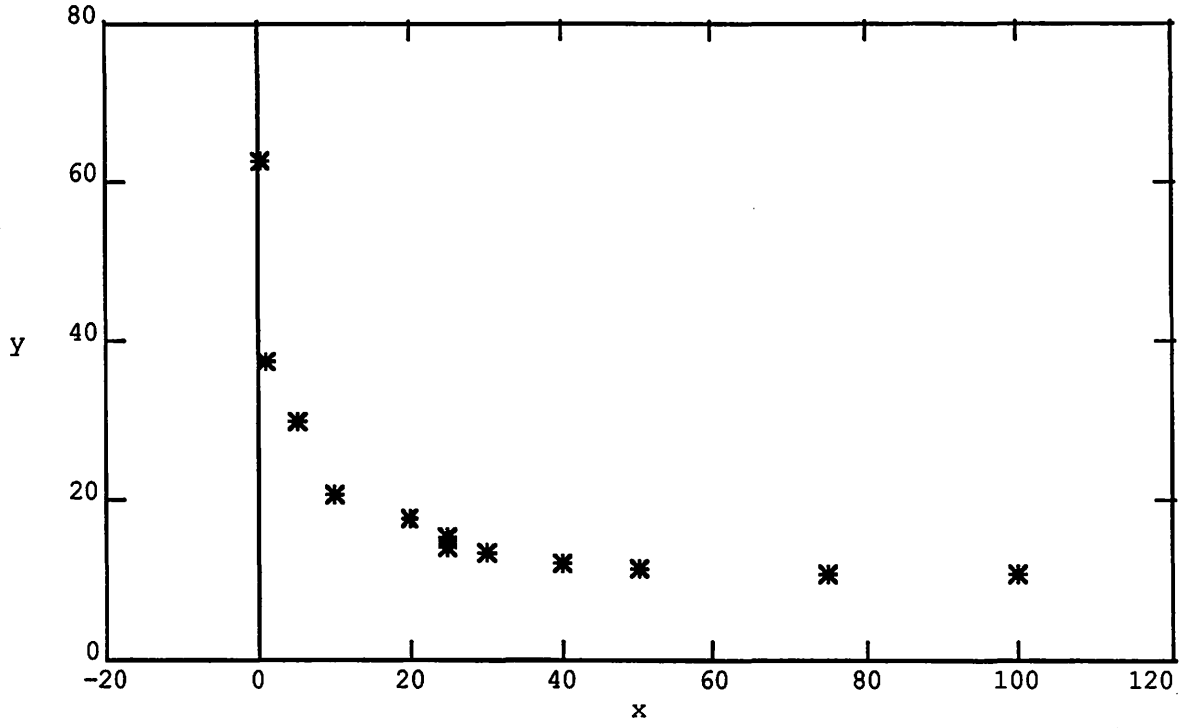
Simple regression. This is example 18 from page 153 of DP. The x is the distance in meters from a highway and the y is the lead content of the soil in ppm.

```
Cmd> x<-cat(.3,1,5,10,20,25,25,30,40,50,75,100)
```

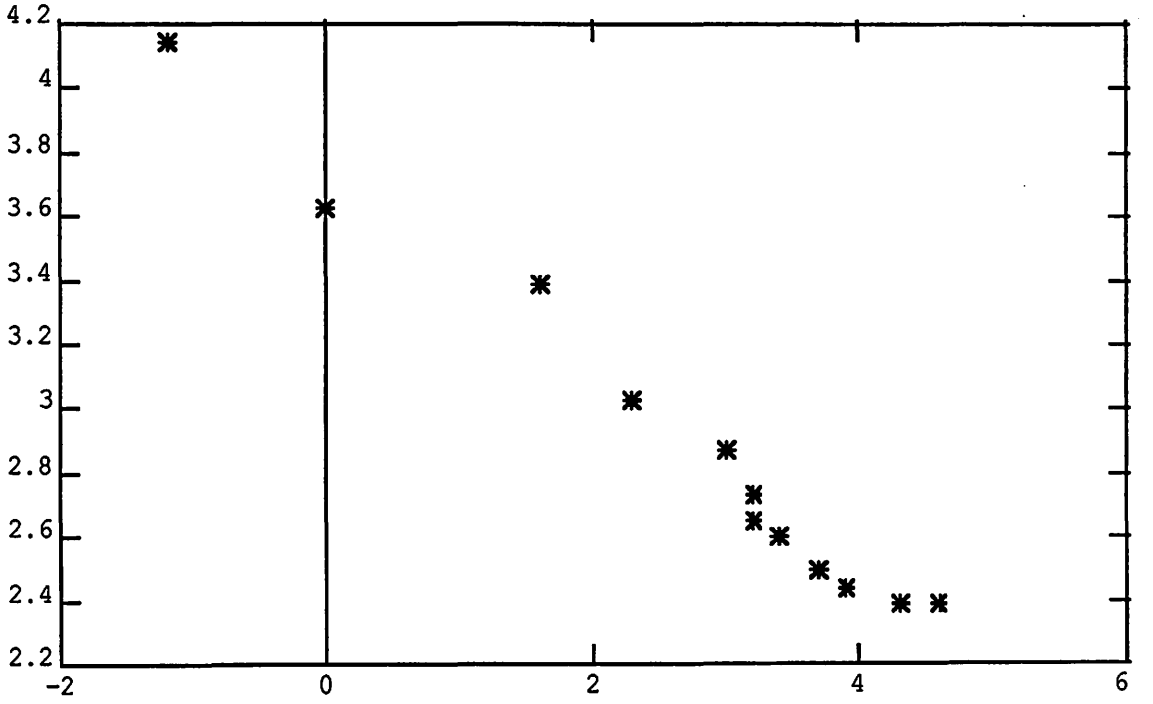
The data

```
Cmd> y<-cat(62.75,37.51,29.70,20.71,17.65,15.41,14.15,13.50,12.11,11.40,
10.85,10.85)

Cmd> plot(x,y)
```

Looks pretty curved




```
Cmd> plot(log(x),log(y))
```
Looks better



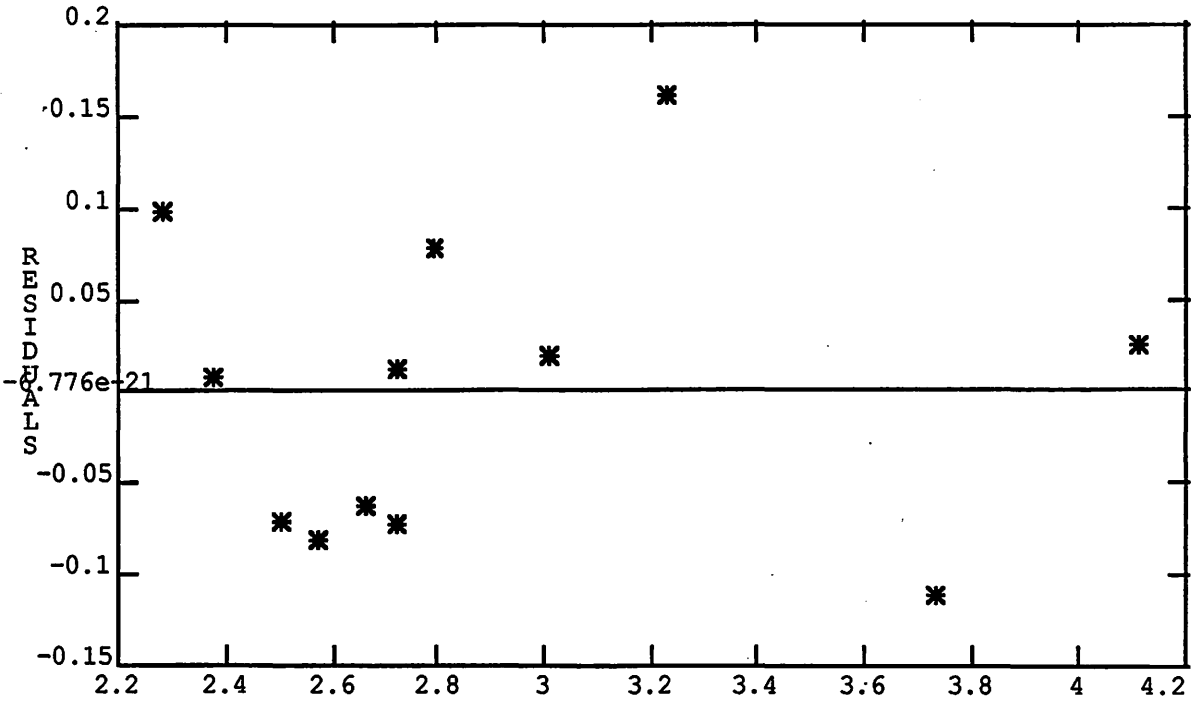

```
Cmd> logy <- log(y)

Cmd> logx <- log(x)

Cmd> regress("logy = logx")
```
regression on transformed data

	Coef	sd	t
CONSTANT	3.736	0.047431	78.767

```
logx                    -0.31472      0.015039      -20.928

N: 12   MSE:     0.007617  DF: 10  R^2:      0.97768

Sequential sums of squares

                                          DF          SS          MS
CONSTANT                                   1      100.57      100.57
logx                                       1       3.336       3.336
ERROR1                                    10     0.07617    0.007617
```

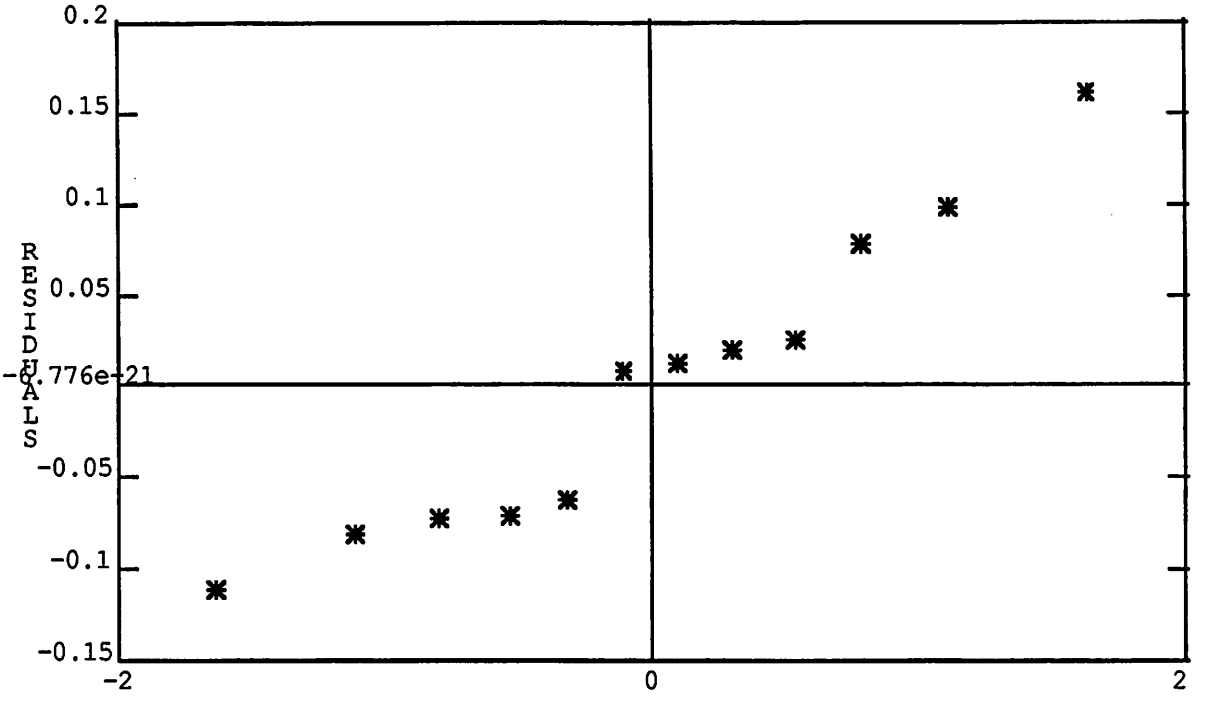
```
Cmd> plot(logy-RESIDUALS,RESIDUALS)
```

plot of residuals versus predicted




```
Cmd> plot(rankits(RESIDUALS),RESIDUALS)
```

rankit plot of residuals



Multiple regression. This data is Example 6, page 509 of DP. The two X variables are extractable iron and extractable aluminum; the Y variable is a phosphate adsorption index.

```
Cmd> iron <- cat(61,175,111,124,130,173,169,169,160,244,257,333,199)

Cmd> aluminum <- cat(13,21,24,23,64,38,33,61,39,71,112,88,54)

Cmd> adsorption <- cat(4,18,14,18,26,26,21,30,28,36,65,62,40)

Cmd> regress("adsorption = iron + aluminum")
                              Coef          sd           t
CONSTANT                   -7.3507      3.4847     -2.1094
iron                       0.11273    0.029691      3.7969
aluminum                     0.349    0.071306      4.8944

N: 13   MSE:      19.179  DF: 10  R^2:     0.94847

Sequential sums of squares

                                          DF          SS          MS
CONSTANT                                   1       11580       11580
iron                                       1      3070.5      3070.5
aluminum                                   1      459.43      459.43
ERROR1                                    10      191.79      19.179

Cmd> HII
(1)          0.30754     0.27815     0.15449     0.14506     0.35893
(6)          0.10313     0.12684     0.13338     0.08757     0.15238
(11)          0.53456     0.53144    0.086537
```

Cases 11 and 12 have fairly high leverage

```
Cmd> studres <- RESIDUALS/(19.179*(1-HII))^0.5
```

```
Cmd> studres
```
internally studentized residuals

```
(1)        -0.017302    -0.45867     0.11455     0.82601     -1.0383
(6)          0.14126    -0.54205    -0.73346     0.88505     -2.2161
(11)           1.4359     0.36646      1.4504
```

```
Cmd> tres <- studres*( (13-3-1)/(13-3-studres*studres) )^0.5
```

```
Cmd> tres
```
externally studentized residuals (t type)

```
(1)        -0.016414    -0.43978     0.10875      0.8118     -1.0428
(6)          0.13414    -0.52196    -0.71533     0.87459     -2.9471
(11)           1.529     0.35002      1.5484
```

Case 10 fits the model poorly

```
Cmd> CooksD <- studres*studres*HII/(1-HII)/3
```

```
Cmd> CooksD
```
Cook's distance

```
(1)        4.432e-005    0.027022  0.00079927    0.038588     0.20119
(6)        0.00076483    0.014227    0.027598    0.025059     0.29429
(11)          0.78937    0.050772     0.06643
```

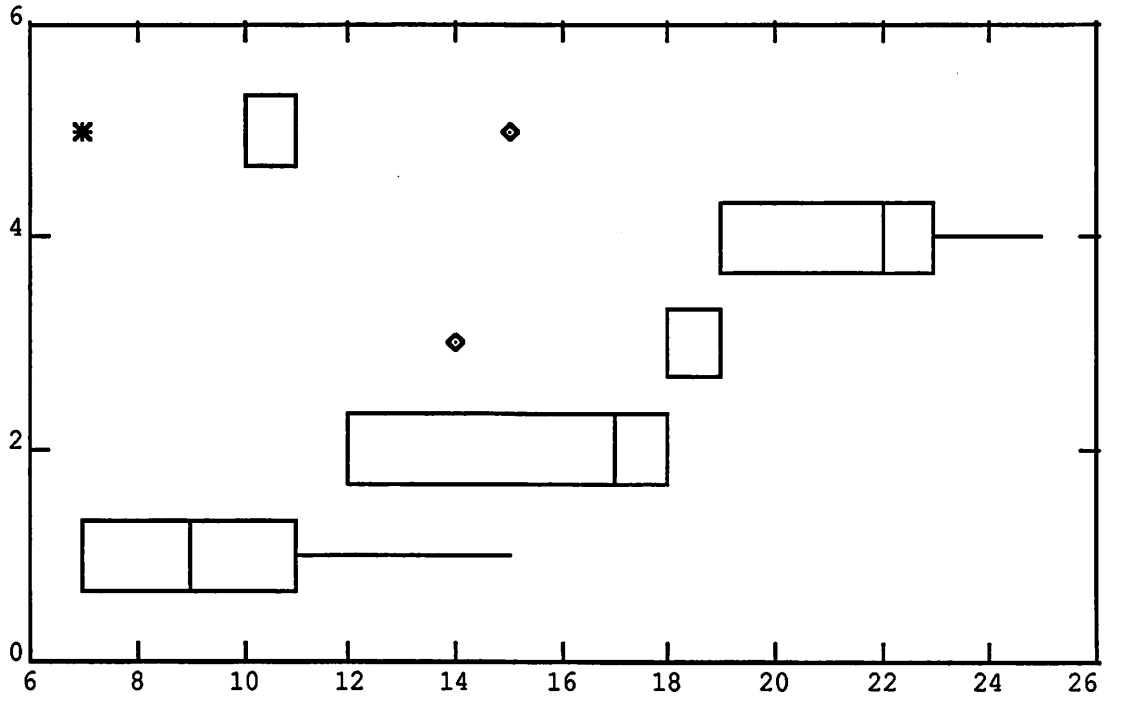
Deletion of case 11 would most influence the analysis. See Weisberg (1985) for a discussion of residuals and Cook's distance.

One way ANOVA. This is example 3-1 of M. The response is the tensile strength of a synthetic fiber; the experimental treatment is the percent of cotton: 15, 20, 25, 30 or 35. There are five observations for each treatment.

```
Cmd> strength <-
cat(7,7,15,11,9,12,17,12,18,18,14,18,18,19,19,19,25,22,19,23,7,10,11,15,11)
```

```
Cmd> grp <- factor(cat(1,1,1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4,5,5,5,5,5))
```

```
Cmd> boxplot(split(strength,grp))
```
Clear differences in strength as percent cotton varies.




```
Cmd> anova("strength=grp")
```

One-way ANOVA with 5 groups

	DF	SS	MS
CONSTANT	1	5655	5655
grp	4	475.76	118.94
ERROR1	20	161.2	8.06

```
Cmd> contrast("grp",cat(-2,-1,0,1,2))
(1)              33.62
```

SS for linear effect of percent (compare with polynomial model below)

```
Cmd> cellstats("grp")
```

Cell by cell statistics

```
component: mean
(1)               9.8        15.4        17.6        21.6        10.8
component: var
(1)              11.2         9.8         4.3         6.8         8.2
component: count
(1)                 5           5           5           5           5
```

```
Cmd> percent <- 5*grp+10
```

Set up polynomials in percent cotton

```
Cmd> percent2<-percent*percent

Cmd> percent3<-percent2*percent

Cmd> percent4<-percent3*percent

Cmd> anova("strength=percent+percent2+percent3+percent4")
WARNING: unbalanced data so SS are sequential
```

	DF	SS	MS
CONSTANT	1	5655	5655
percent	1	33.62	33.62
percent2	1	343.21	343.21
percent3	1	64.98	64.98
percent4	1	33.946	33.946

```
ERROR1                                    20          161.2          8.06

Cmd> 1-cumF(33.95/8.06,1,20)
(1)          0.053455
```

Strict believers in 5% significance (as M apparantly is) would conclude the cubic term is significant and the quartic term is not.

```
Cmd> 1-cumF(64.98/8.06,1,20)
(1)          0.010133

Cmd> regress("strength=percent+percent2+percent3")
```

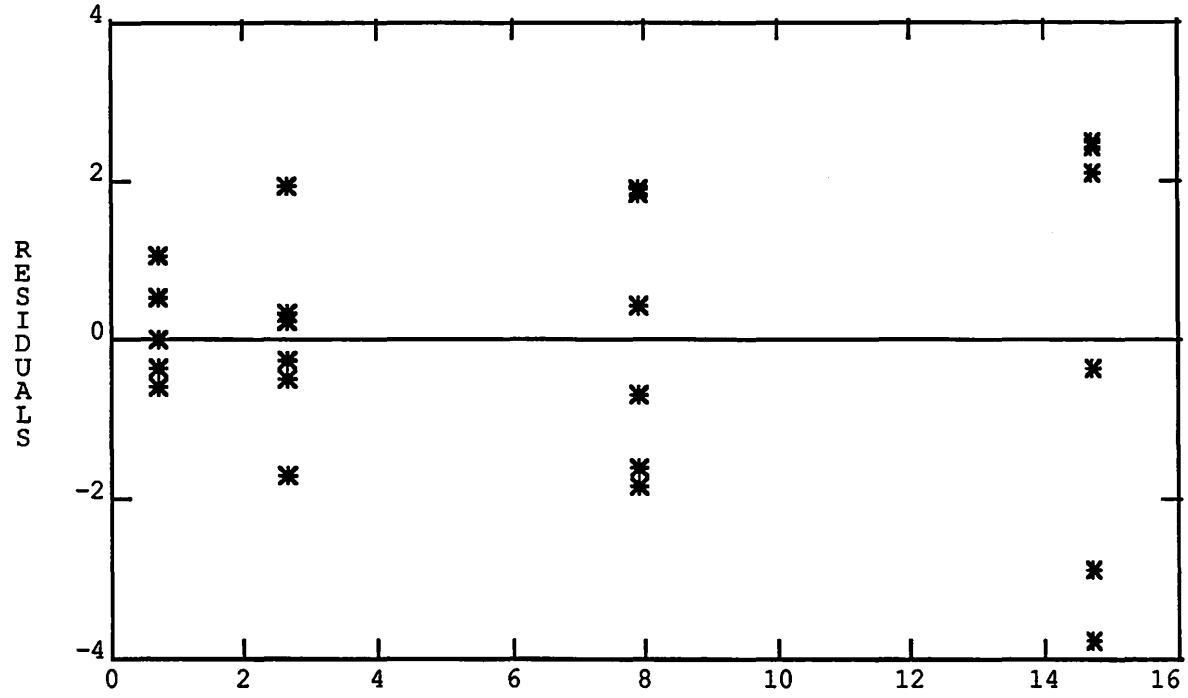
polynomial regression

```
                              Coef           sd            t
CONSTANT                    62.611       39.757       1.5748
percent                    -9.0114       5.1966      -1.7341
percent2                   0.48143      0.21605       2.2284
percent3                   -0.0076     0.002874      -2.6444

N: 25   MSE:       9.2927  DF: 21  R^2:      0.69363

Sequential sums of squares

                                          DF           SS           MS
CONSTANT                                   1         5655         5655
percent                                    1        33.62        33.62
percent2                                   1       343.21       343.21
percent3                                   1        64.98        64.98
ERROR1                                    21       195.15       9.2927
```

Variance stabilizing transformations. This is example 4-2, page 94 from M. The responses are estimates of peak discharge during flood flow when using four different estimation techniques.

```
Cmd> discharge<-cat(.34,.12,1.23,.7,1.75,.12,.91,2.94,2.14,2.36,2.86,4.55,
6.31,8.37,9.75,6.09,9.82,7.24,17.15,11.82,10.95,17.2,14.35,16.82)

Cmd> method<-factor(cat(1,1,1,1,1,1,2,2,2,2,2,2,3,3,3,3,3,3,4,4,4,4,4,4))

Cmd> anova("discharge=method")
                                          DF           SS           MS
CONSTANT                                   1       1012.6       1012.6
method                                     3       708.35       236.12
ERROR1                                    20       62.081       3.1041

Cmd> plot(discharge-RESIDUALS,RESIDUALS)
```

evidence that variance increases with mean response

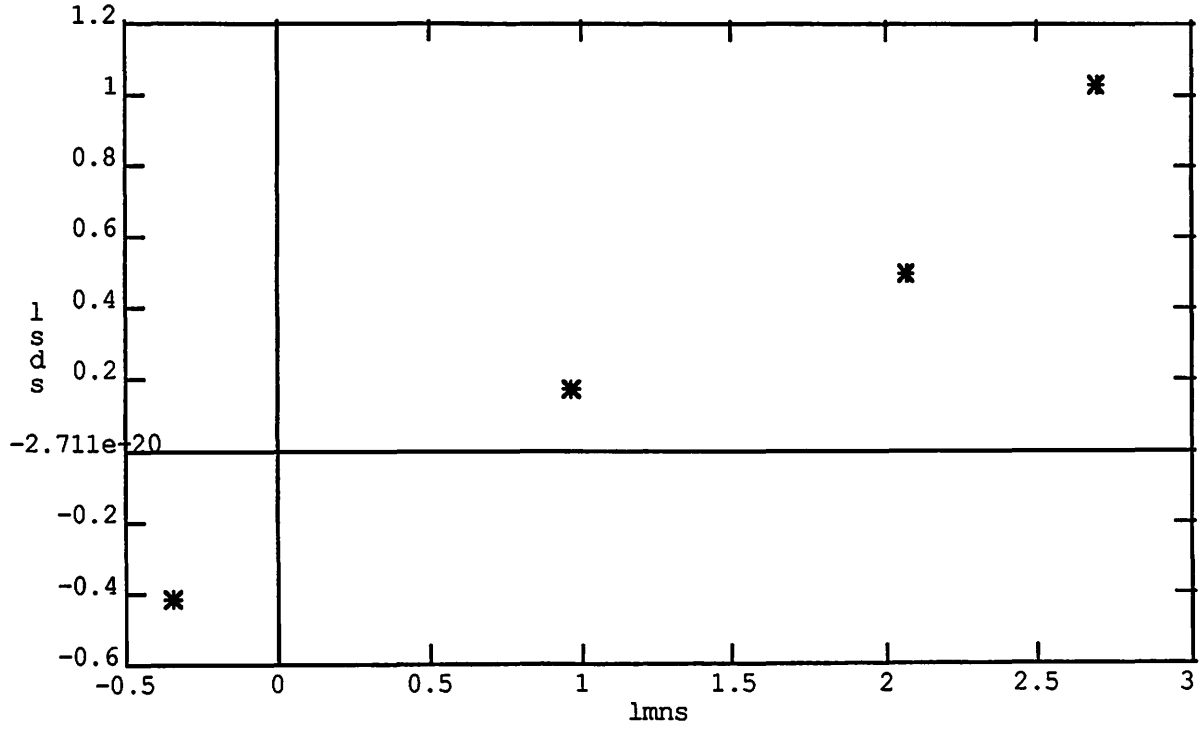



```
Cmd> stats<-cellstats("method")
```

```
Cmd> lmns<-log(stats$mean)
```
log cell means

```
Cmd> lsds<-log((stats$var)^0.5)
```
log cell standard deviations

```
Cmd> plot(lmns,lsds)
```



```
Cmd> regress("lsds=lmns")
                          Coef          sd           t
CONSTANT               -0.27809     0.10031     -2.7724
lmns                    0.44647    0.056589      7.8897

N: 4   MSE:     0.017049  DF: 2  R^2:      0.96887

Sequential sums of squares

                                        DF          SS          MS
CONSTANT                                 1     0.41659     0.41659
lmns                                     1      1.0613      1.0613
ERROR1                                   2    0.034098    0.017049
```

1 minus the regression coefficient is the suggested transformation. 1-.44 is approximately a square root

```
Cmd> rtdischarge<-discharge^0.5

Cmd> anova("rtdischarge=method")
                                        DF          SS          MS
CONSTANT                                 1      120.52      120.52
method                                   3      32.684      10.895
ERROR1                                  20      2.6884     0.13442
```

```
Cmd> cellstats("method")
```

better on this scale

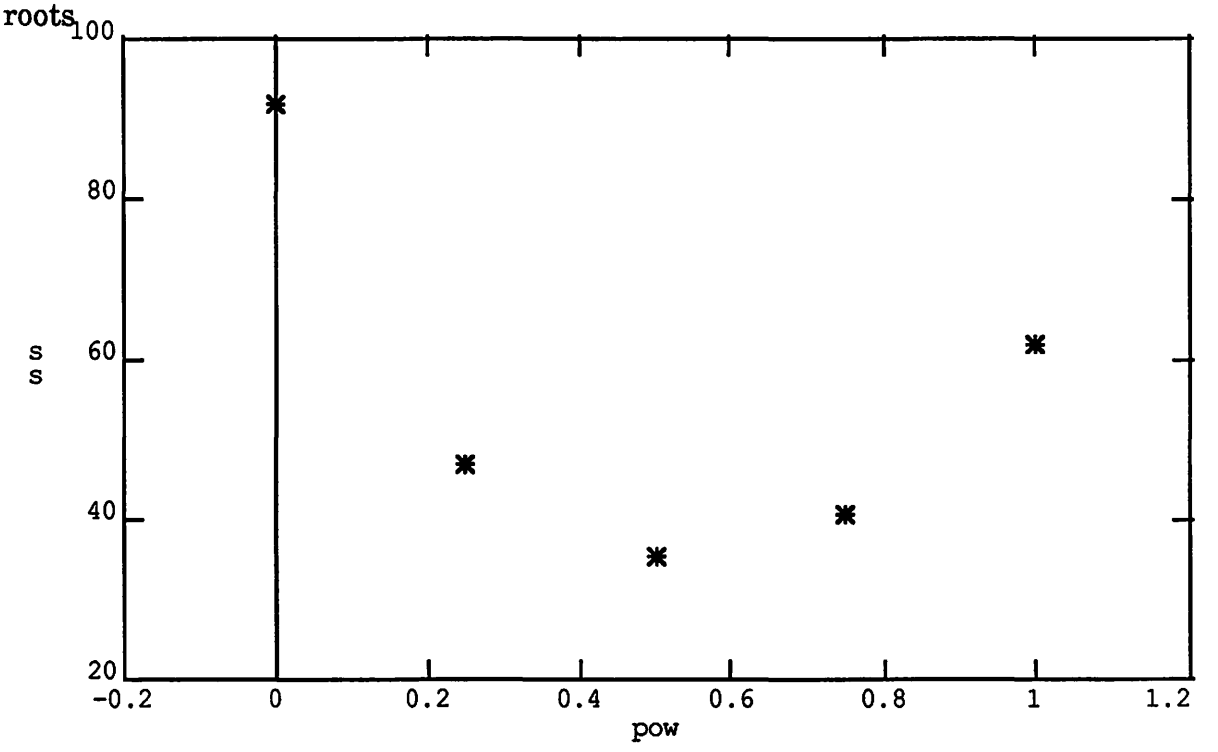
```
component: mean
(1)          0.75742       1.582      2.8033      3.8208
component: var
(1)          0.16358     0.14879    0.085844     0.13947
component: count
(1)                6           6           6           6
```

As an alternative, lets try Box-Cox transformations

```
Cmd> discharge0<-boxcox("discharge","0")

Cmd> discharge25<-boxcox("discharge","0.25")

Cmd> discharge50<-boxcox("discharge","0.50")

Cmd> discharge75<-boxcox("discharge","0.75")

Cmd> anova("discharge0=method")
      (                                 DF          SS          MS
CONSTANT                                 1      370.12      370.12
method                                   3      461.16      153.72
ERROR1                                  20      91.958      4.5979

Cmd> anova("discharge25=method")
                                        DF          SS          MS
CONSTANT                                 1      423.29      423.29
method                                   3      406.96      135.65
ERROR1                                  20      46.989      2.3494

Cmd> anova("discharge50=method")
                                        DF          SS          MS
CONSTANT                                 1      486.93      486.93
method                                   3      430.66      143.55
ERROR1                                  20      35.424      1.7712
```

```
Cmd> anova("discharge75=method")
                                        DF          SS          MS
CONSTANT                                 1       579.9       579.9
method                                   3      524.17      174.72
ERROR1                                  20      40.607      2.0304

Cmd> ss<-cat(91.96,46.99,35.43,40.61,62.08)

Cmd> pow<-cat(0,.25,.5,.75,1)

Cmd> plot(pow,ss)
```

Ploting error SS against BC parameter also suggests square roots



Randomized complete block. This is example 5-1, page 129 of M. The response is the depth of a depression made in metal when a tip is pressed with a standard force into the metal. We wish to see if there are any differences in readings between four types of tips (treatment); we use each tip once in each of four metal specimens (block).

```
Cmd>
depth<-cat(9.3,9.4,9.6,10,9.4,9.3,9.8,9.9,9.2,9.4,9.5,9.7,9.7,9.6,10,10.2)

Cmd> tip<-factor(cat(1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4))

Cmd> specimen<-factor(cat(1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4))

Cmd> anova("depth=specimen+tip")
                                        DF          SS          MS
CONSTANT                                 1      1482.2      1482.2
specimen                                 3       0.825       0.275
tip                                      3       0.385     0.12833
ERROR1                                   9        0.08   0.0088889
```

```
Cmd> anova("depth=tip+specimen")
```
Order of terms doesn't matter when balanced

	DF	SS	MS
CONSTANT	1	1482.2	1482.2
tip	3	0.385	0.12833
specimen	3	0.825	0.275
ERROR1	9	0.08	0.0088889

```
Cmd> depth[1]<-?
```
Make tip 1 block 1 missing, now unbalanced and order matters

```
Cmd> anova("depth=specimen+tip")
```
block then treatment is correct

```
WARNING: unbalanced data so SS are sequential
WARNING: cases with MISSING values deleted
```

	DF	SS	MS
CONSTANT	1	1395.9	1395.9
specimen	3	0.72567	0.24189
tip	3	0.37611	0.12537
ERROR1	8	0.075556	0.0094444

```
Cmd> anova("depth=tip+specimen")
```
treatment then block is **incorrect**

```
WARNING: unbalanced data so SS are sequential
WARNING: cases with MISSING values deleted
```

	DF	SS	MS
CONSTANT	1	1395.9	1395.9
tip	3	0.37317	0.12439
specimen	3	0.72861	0.24287
ERROR1	8	0.075556	0.0094444

Latin squares. Data from Table 5-9, page 146 of M. Blocks are batches (rows) and operators (columns), treatment is mixing formulation of dynamite (letters), and response is a measure of explosive force.

```
Cmd> force<-cat(24,20,19,24,24,17,24,30,27,36,18,38,26,27,21,26,31,26,23,22,
22,30,20,29,31)

Cmd> operator<-factor(cat(1,2,3,4,5,1,2,3,4,5,1,2,3,4,5,1,2,3,4,5,1,2,3,4,5))

Cmd> batches<-factor(cat(1,1,1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4,5,5,5,5,5))

Cmd> mix<-factor(cat(1,2,3,4,5,2,3,4,5,1,3,4,5,1,2,4,5,1,2,3,5,1,2,3,4))

Cmd> anova("force=batches+operator+mix")
WARNING: unbalanced data so SS are sequential
```

	DF	SS	MS
CONSTANT	1	16129	16129
batches	4	68	17
operator	4	150	37.5
mix	4	330	82.5
ERROR1	12	128	10.667

For a complete data Latin square, the order of the terms in the model does not matter.

Balanced incomplete blocks. The data is from Table 6-1 of M. The response is elapsed time for a reaction, the blocks are batches of raw material, and the treatments are different types of catalyst. In this example we do the intrablock analysis. Always have blocks precede treatments in the model.

```
Cmd> time<-cat(73,74,71,75,67,72,73,75,68,75,72,75)

Cmd> catalyst<-factor(cat(1,1,1,2,2,2,3,3,3,4,4,4))

Cmd> batches<-factor(cat(1,2,4,2,3,4,1,2,3,1,3,4))


Cmd> anova("time=batches+catalyst")
WARNING: unbalanced data so SS are sequential
                                        DF          SS          MS
CONSTANT                                 1       63075       63075
batches                                  3          55      18.333
catalyst                                 3       22.75      7.5833
ERROR1                                   5        3.25        0.65

Cmd> contrast("catalyst",cat(1,1,1,-3))
(1)          22.222
```

The catalyst four versus average of the first three catalysts contrast SS indicates that catalyst four must be separated from the other three.

Analysis of covariance. This is example 16-1, page 480 of M. The response is breaking strength of a fiber, the covariate is diameter of the fiber, and there are three different machines (treatments) making the fibers that we would like to compare.

```
Cmd> strength<-cat(36,41,39,42,49,40,48,39,45,44,35,37,42,34,32)

Cmd> diam<-cat(20,25,24,25,32,22,28,22,30,28,21,23,26,21,15)

Cmd> machine<-factor(cat(1,1,1,1,1,2,2,2,2,2,3,3,3,3,3))

Cmd> anova("strength=machine")   analysis ignoring covariate
                                        DF          SS          MS
CONSTANT                                 1       24241       24241
machine                                  2       140.4        70.2
ERROR1                                  12         206      17.167
```

```
Cmd> anova("strength=diam+machine")   Analysis with covariate (parallel lines
model). Notice that the machine effect is much smaller after allowing for the
covariate.
WARNING: unbalanced data so SS are sequential
                                        DF          SS          MS
CONSTANT                                 1       24241       24241
diam                                     1      305.13      305.13
machine                                  2      13.284      6.6419
ERROR1                                  11      27.986      2.5442

Cmd> anovacoefs("diam")   Slope of covariate.
(1)          0.95399

Cmd> anovacoefs("machine")
(1)          MISSING      0.18241       1.2192
```

```
Cmd> -(.1824+1.2192)
(1)          -1.4016
```
(Adjusted group effects are .18, 1.22, and -1.40)

```
Cmd> anova("strength=diam+machine.diam")
WARNING: unbalanced data so SS are sequential
```
Separate slopes model.

	DF	SS	MS
CONSTANT	1	24241	24241
diam	1	305.13	305.13
diam.machine	2	13.357	6.6784
ERROR1	11	27.913	2.5375

Factorial models. Example 7-4, page 225 of M. The response is (a transformation of) volume of beverage packaged; treatments are percent carbonation, pressure, and speed.

```
Cmd> volume<-cat(-3,-1,0,1,5,4,-1,0,2,1,7,6,-1,0,2,3,7,9,1,1,6,5,10,11)

Cmd> percent<-factor(cat(1,1,2,2,3,3,1,1,2,2,3,3,1,1,2,2,3,3,1,1,2,2,3,3))

Cmd> psi<-factor(cat(1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2))

Cmd> speed<-factor(cat(1,1,1,1,1,1,2,2,2,2,2,2,1,1,1,1,1,1,2,2,2,2,2,2))

Cmd> anova("volume=percent*psi*speed")
```
Full factorial model

	DF	SS	MS
CONSTANT	1	234.38	234.38
percent	2	252.75	126.38
psi	1	45.375	45.375
percent.psi	2	5.25	2.625
speed	1	22.042	22.042
percent.speed	2	0.58333	0.29167
psi.speed	1	1.0417	1.0417
percent.psi.speed	2	1.0833	0.54167
ERROR1	12	8.5	0.70833

```
Cmd> anova("volume=percent*psi*speed-percent.psi.speed")
```
Remove the three way interaction from the model and lump it into error.

	DF	SS	MS
CONSTANT	1	234.38	234.38
percent	2	252.75	126.38
psi	1	45.375	45.375
percent.psi	2	5.25	2.625
speed	1	22.042	22.042
percent.speed	2	0.58333	0.29167
psi.speed	1	1.0417	1.0417
ERROR1	14	9.5833	0.68452

Lets get the effects for the percent.psi interaction.

```
Cmd> anova("volume=percent+psi")
```

	DF	SS	MS
CONSTANT	1	234.38	234.38
percent	2	252.75	126.38
psi	1	45.375	45.375
ERROR1	20	38.5	1.925

```
Cmd> res<-RESIDUALS

Cmd> anova("res=percent*psi")
```

	DF	SS	MS
CONSTANT	1	0	0
percent	2	0	0
psi	1	0	0
percent.psi	2	5.25	2.625
ERROR1	18	33.25	1.8472

```
Cmd> cellstats("percent.psi")$mean
(1,1)            0.625      -0.625
(2,1)           -0.125       0.125
(3,1)             -0.5         0.5
```

Confounding. The basic rule for computing ANOVA tables for confounded factorial designs is to include a factor for blocks in the model preceding the factorial terms. This works for complete or partial confounding. The degrees of freedom between blocks may be broken up into replicates and blocks within replicates if so desired.

This is example 10-3 page 319 of M. It is a partially confounded 2^3 factorial in two reps, where ABC is confounded in rep 1 and AB is confounded in rep 2.

```
Cmd> y<-cat(-3,2,2,1,0,-1,-1,6,-1,0,3,5,1,0,1,1)

Cmd> a<-factor(cat(1,2,2,1,2,1,1,2,1,1,2,2,2,1,2,1))

Cmd> b<-factor(cat(1,2,1,2,1,2,1,2,1,1,2,2,1,2,1,2))

Cmd> c<-factor(cat(1,1,2,2,1,1,2,2,1,2,1,2,1,1,2,2))

Cmd> rep<-factor(cat(1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2))

Cmd> blk<-factor(cat(1,1,1,1,2,2,2,2,1,1,1,1,2,2,2,2))
```

`Cmd> anova("y=rep/blk+a*b*c")` Here we have blocks nested within replicates plus the factorial terms. Of course, we only have half information on the partially confounded terms.

```
WARNING: unbalanced data so SS are sequential
```

	DF	SS	MS
CONSTANT	1	16	16
rep	1	1	1
rep.blk	2	2.5	1.25
a	1	36	36
b	1	20.25	20.25
a.b	1	0.5	0.5
c	1	12.25	12.25
a.c	1	0.25	0.25
b.c	1	1	1
a.b.c	1	0.5	0.5
ERROR1	5	3.75	0.75

Fractional factorials. Fractional factorial designs may be analyzed just like an ordinary factorial. Effects aliased to preceding effects will have zero degrees of freedom.

The following data are from Table 11-3 of M and are from a 2^{4-1} fractional factorial with defining relation I=ABCD.

```
Cmd> a<-factor(cat(1,2,1,2,1,2,1,2))
```

```
Cmd> b<-factor(cat(1,1,2,2,1,1,2,2))

Cmd> c<-factor(cat(1,1,1,1,2,2,2,2))

Cmd> d<-factor(cat(1,2,2,1,2,1,1,2))

Cmd> y<-cat(45,100,45,65,75,60,80,96)

Cmd> anova("y=a+b+c+d")
WARNING: unbalanced data so SS are sequential
                                        DF          SS          MS
CONSTANT                                 1       40044       40044
a                                        1         722         722
b                                        1         4.5         4.5
c                                        1         392         392
d                                        1       544.5       544.5
ERROR1                                   3      1408.5       469.5

Cmd> anova("y=d+a*b*c")
WARNING: unbalanced data so SS are sequential
                                        DF          SS          MS
CONSTANT                                 1       40044       40044
d                                        1       544.5       544.5
a                                        1         722         722
b                                        1         4.5         4.5
a.b                                      1           2           2
c                                        1         392         392
a.c                                      1       684.5       684.5
b.c                                      1         722         722
a.b.c                                    0           0           0
ERROR1                                   0    1.56e-033           0
```

Split plots. Split plots are experiments with a randomization restriction wherein one of the factors is confounded with a blocking factor. In this sense, they are similar to a confounded factorial. Computationally, there is no difference between a confounded factorial and a split plot. However, it is conventional to label the replicates by whole plot treatment interaction as an error term, since that interaction is the appropriate denominator when testing the whole plot treatment.

The example data are a subset of Table 13-7 of M. There are two replicates, a whole plot treatment with three levels, and a split plot treatment with four levels.

```
Cmd>
y<-cat(30,35,37,36,34,41,38,42,29,26,33,36,28,32,40,41,31,36,42,40,31,30,32,40
)

Cmd> rep<-factor(cat(1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2))

Cmd> spt<-factor(cat(1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4))

Cmd> wpt<-factor(cat(1,1,1,1,2,2,2,2,3,3,3,3,1,1,1,1,2,2,2,2,3,3,3,3))

Cmd> anova("y=rep+wpt+E(rep.wpt)+spt+spt.wpt")
                                        DF          SS          MS
CONSTANT                                 1       29400       29400
rep                                      1         1.5         1.5
wpt                                      2      138.25      69.125
ERROR1                                   2       14.25       7.125
```

spt	3	266.33	88.778
wpt.spt	6	58.417	9.7361
ERROR2	9	53.25	5.9167

Error1 and Error2 are the whole plot and split plot errors respectively.

7. References

Aitkin, M. et al. (1985). The Generalized Linear Interactive Modelling System. Oxford: Numerical Algorithms Group.

Becker, Richard A. and John M. Chambers (1984) S: an Interactive Environment for Data Analysis and Graphics Belmont, CA: Wadsworth.

Devore, Jay and Roxy Peck (1986). Statistics: the Exploration and Analysis of Data St. Paul, MN: West.

Kernighan, Brian W. and Rob Pike (1984). The UNIX Programming Environment. Englewood Cliffs, NJ: Prentice-Hall.

Montgomery, Douglas C. (1984). Design and Analysis of Experiments, 2nd Ed. New York: Wiley.

Weisberg, Sanford (1985). Applied Linear Regression, 2nd Ed. New York: Wiley.