MacAnova Reference Manual

Version 4.06

Technical Report #618
School of Statistics

University of Minnesota

Christopher Bingham Gary W. Oehlert

September 1997, Revised May 1998

Contents

Chapter 1

Introduction

This document is provided as a detailed reference to the commands and functions distributed with MacAnova. The sections here are slightly reformatted versions of the material that is available on-line in MacAnova using the `help()` command. Thus, for example, `help(anova)` in MacAnova prints the same material that is given here.

The MacAnova distribution includes three help files: MacAnova.hlp, Design.hlp, and Userfun.hlp. Help topics from these files are arranged in three sections below. Some topics in MacAnova.hlp have not been included here as they are primarily concerned with news about changes in MacAnova; these sections are: `macanova3`, `news`, and `updates`.

Note that help topics with names including an underscore character _ (for example, `data_files` or `c_macros`) have had the underscore changed to a space in this document for formatting purposes.

All MacAnova help files can contain search keys for finding commands. For example, "Plotting" is a search key that can be used to find help topics related to plotting via the command `help(key:"plotting")`. Chapter 5 contains tables for each help file that list the help topics related to each search key.

This document is not an introduction to or explanation of how MacAnova itself works. For that, consult the *MacAnova User's Guide* by Oehlert and Bingham.

Chapter 2

MacAnova Help File

This Chapter contains MacAnova help topics that are in the standard MacAnova help file.

2.1 addchars

Keywords: plotting
Usage: addchars([Graph,] x,y,[,c] [,lines:T,impulse:T] [, graphics keyword phrases])

addchars(x,y,c) is equivalent to chplot(x,y,c,add:T). It adds character labeled points to the plot in LASTPLOT, displays the plot, and updates LASTPLOT with the new information.

Arguments x, y, and c are as for chplot() and the points are labeled the same way as is done by chplot(). When c is ommitted, the same default is used as for chplot().

It is an error if LASTPLOT does not exist.

addchars(Graph,x,y,c) or chplot(Graph,x,y,c,add:T) displays GRAPH variable Graph with the addition of character labeled points, saving the modified plot in LASTPLOT. Graph is not changed (unless it is LASTPLOT).

addchars(x,y,c,keep:F) suppresses any change to LASTPLOT.

addchars(x,y,c,show:F) suppresses immediate display of the modified graph but updates LASTPLOT. This is useful when you are building a complex graph in stages using addchars(), adchars(), addstrings(), or addpoints(). When you are done, simply type showplot(). You can't use both show:F and keep:F.

Keywords 'dumb', 'lines', 'xmin', 'xmax', 'ymin', 'ymax', 'xlab', 'ylab', 'title', 'xaxis', 'yaxis', 'height', 'width' and 'pause' may be used as for chplot().

If option 'dumbplot' has been set False (see setoptions()), the plot

will be a low resolution plot unless 'dumb:F' is an argument.

A value of MISSING for any of xmin, xmax, ymin or ymax (for example, xmin:?) forces determination of a value from the current data and the extreme value of data already in the graph. Alternatively, you can force recomputation of both xmin and xmax (ymin and ymax) by keyword phrases xmin:0, xmax:0 (ymin:0, ymax:0).

New labels and title may be set only by keywords 'xlab', 'ylab' and 'title'.

See topic graphs for information on keywords.

2.2 addlines

Keywords: plotting
Usage: addlines([Graph,] x,y [,linetype:PosInt,thickness:PosReal] [,impulse:T] [,other graphics keyword phrases])

addlines(x,y) is equivalent to lineplot(x,y,add:T). It displays the plot in LASTPLOT, adding lines such as are produced by lineplot(), and updates LASTPLOT with the new information.

Arguments x, and y are as for lineplot(). They can be replaced by a structure with at least two REAL components. Any components beyond the first two are ignored.

It is an error if LASTPLOT does not exist.

If option 'dumbplot' has been set False (see setoptions()), the plot will be a low resolution plot unless 'dumb:F' is an argument.

addlines(Graph,x,y) or lineplot(Graph,x,y,add:T) displays GRAPH variable Graph with the addition of lines connectings points specified by x and y, saving the modified plot in LASTPLOT. Graph is not changed (unless it is LASTPLOT).

addlines(x,y,keep:F) (or lineplot(x,y,keep:F,add:T)) suppresses any change to LASTPLOT.

addlines(x,y,show:F) (or lineplot(x,y,show:F,add:T)) suppresses immediate display of the modified graph but updates LASTPLOT. This is useful when you are building a complex graph in stages using addlines(), adchars(), addstrings(), or addpoints(). When you are done, simply type showplot(). You can't use both show:F and keep:F.

You can use keywords 'linetype' and 'thickness' to control the type of lines used (solid, dashed, etc.). See graphs.

Keywords 'dumb', 'xmin', 'xmax', 'ymin', 'ymax', 'xlab', 'ylab', 'title', 'xaxis', 'yaxis', 'height', 'width' and 'pause' may be used as

for other plotting commands. See graphs.

A value of MISSING for any of xmin, xmax, ymin or ymax (for example, xmin:?) forces determination of a value from the current data and the extreme value of data already in the graph. Alternatively, you can force recomputation of both xmin and xmax (ymin and ymax) by keyword phrases xmin:0, xmax:0 (ymin:0, ymax:0).

New labels and title may be set only by keywords 'xlab', 'ylab' and 'title'.

See topic graphs for more information on keywords and LASTPLOT.

2.3 adddatapath

Keywords: input, files
Usage: adddatapath(dirName [,T]), dirName a quoted string or CHARACTER vector specifying one or more additional directory or folder names to search when attempting to read a file

adddatapath(DirName) adds the directory or folder name specified by quoted string or CHARACTER scalar DirName to the CHARACTER vector DATAPATHS. On any commands which read a file, if MacAnova does not find the file in the default directory (see files), it then searches for it in the folders or directories whose names are in DATAPATHS. DirName is added at the beginning, so that the folder or directory will be searched before any other files listed in DATAPATHS.

adddatapath(DirName,T) does the same except DirName is added at the end of DATAPATHS so the directory or folder will be searched last.

For both usages, DirName can also be a CHARACTER vector, each element of which specifies a directory or folder to be searched.

If DATAPATHS does not already exist, adddatapaths(DirName) creates it.

Example:

```
adddatapath("MyDisk:Survey Folder:") # on Macintosh
adddatapath("D:/SURVEY.DIR/") # on DOS/Windows
```

See also matread(), vecread(), macroread(), inforead(), files

2.4 addmacrofile

Keywords: macros, files
Usage: addmacrofile(fileName [,T]), fileName a quoted string or CHARACTER vector specifying one or more additional files to be searched by getmacros.

addmacrofile(fileName) adds the file name specified by quoted string or CHARACTER scalar fileName to CHARACTER vector MACROFILES which specifies files to be searched for macros by pre-defined macro getmacros. The file name are added at the beginning, so the file will be searched before any other files listed in MACROFILES.

addmacrofile(fileName,T) does the same except the file name is added at the end of MACROFILES so the file will be searched last.

If fileName is a simple file name containing no directory or folder information (for example, "mydata.dat" but not "./mydata.dat" or ":mydata.dat"), the file is first assumed to be in the default directory. If not found there, MacAnova looks for it in the folders or directories listed in variable DATAPATHS. See topics files, macro files, adddatapath.

For both usages, fileName can also be a CHARACTER vector, each element of which specifies a file to be searched.

If MACROFILES does not already exist, addmacrofiles(fileName) creates it.

Example:

```
addmacrofile("survival.mac")
addmacrofile("C:/mvmacros/survival.mac",T) # for DOS/Windows
addmacrofile("MyDisk:MVMacros:Survival.mac",T) # on a Mac
```

See also getmacros, macros.

2.5 addpoints

Keywords: plotting
Usage: addpoints([Graph,] x,y [,lines:T,impulse:T][, graphics keyword phrases])

addpoints(x,y) is equivalent to plot(x,y,add:T). It displays the plot in LASTPLOT, adding points such as are produced by plot(), and updates LASTPLOT with the new information. It is an error if LASTPLOT does not exist.

Arguments x and y are as in plot(). They can be replaced by a structure with at least two components. Any components beyond the first two are ignored.

addpoints(Graph,x,y) displays GRAPH variable Graph with the addition of points such as are produced by plot(), and saves the plot in LASTPLOT. Graph is not changed (unless it is LASTPLOT).

addpoints(x,y,keep:F) suppresses any change to LASTPLOT.

addpoints(x,y,show:F) suppresses immediate display of the modified graph but updates LASTPLOT. This is useful when you are building a complex graph in stages using addlines(), adchars(), addstrings(), or addpoints(). When you are done, simply type showplot(). You can't use both show:F and keep:F.

Keywords 'lines', 'xmin', 'xmax', 'ymin', 'ymax', 'xlab', 'ylab', 'title', 'xaxis', 'yaxis', and 'pause' may be used as for other plotting commands.

A value of MISSING for any of xmin, xmax, ymin or ymax (for example, xmin:?) forces determination of a value from the current data and the extreme value of data already in the graph. Alternatively, you can force recomputation of both xmin and xmax (ymin and ymax) by keyword phrases xmin:0, xmax:0 (ymin:0, ymax:0).

New labels and title may be set only by keywords 'xlab', 'ylab' and 'title'.

See topic graphs for more information on keywords and LASTPLOT.

2.6 addstrings

Keywords: plotting
Usage: addstrings([Graph,] x,y,charVec,[, graphics keyword phrases])

addstrings(x,y,charVec) displays the plot in LASTPLOT and then writes the i-th element of CHARACTER vector charVec at position (x[i], y[i]), updating LASTPLOT to include the new information.

If option 'dumbplot' has been set False (see setoptions()), the plot will be a low resolution plot unless 'dumb:F' is an argument.

addstrings(Graph,x,y,charVec), displays GRAPH variable Graph, adding the string or strings in charVec, and saves the modified plot in LASTPLOT. Graph is not changed (unless it is LASTPLOT).

In contrast with other plotting commands, both x and y must be vectors of the same length. The most usual use is when both x and y are REAL scalars and charVec is a quoted string or CHARACTER scalar to be written at coordinates (x,y) (addstrings(110,20,"Frequency 1 cycle/week")).

By default, each string is written centered at (x[i], y[i]). However, if 'justify:"l"' or 'justify:"r"' is an argument following charVec, each

string will be left or right justified.

addstrings(x,y,charVec,keep:F) suppresses any change to LASTPLOT.

addstrings(x,y,charVec,show:F) suppresses immediate display of the modified graph but updates LASTPLOT. This is useful when you are building a complex graph in stages using addlines(), adchars(), addpoints(), or addstrings(). When you are done, simply type showplot(). You can't use both show:F and keep:F.

Keywords 'xmin', 'xmax', 'ymin', 'ymax', 'xlab', 'ylab', 'title', 'xaxis', 'yaxis', and 'pause' may be used as for other plotting commands.

A value of MISSING for any of xmin, xmax, ymin or ymax (for example, xmin:?) forces determination of a value from the current data and the extreme value of data already in the graph. Alternatively, you can force recomputation of both xmin and xmax (ymin and ymax) by keyword phrases xmin:0, xmax:0 (ymin:0, ymax:0).

New labels and title may be set only by keywords 'xlab', 'ylab' and 'title'.

See topic graphs for more information on keywords and LASTPLOT.

2.7 alltrue

Keywords: logical variables, syntax
Usage: alltrue(arg1,arg2,...,argm), all arguments LOGICAL scalars

alltrue(a1,a2,...,am) is equivalent to a1 && a2 && ... && am, except that no arguments are evaluated unnecessarily, that is, it evaluates no arguments after the first false one. All arguments must be LOGICAL scalars.

Example:

```
if(!alltrue(isscalar(x),isreal(x),x > 0, x == floor(x))){
   error("ERROR: x is not positive integer")
}
```

The apparently more natural way to do the same thing

```
if(!(isscalar(x) && isreal(x) && x > 0 && x == floor(x))){
   error("ERROR: x is not positive integer")
}
```

would not do what you want for a non-REAL x since an attempt would be made to evaluate floor(x), which is illegal for non-REALs.

alltrue is implemented as a pre-defined macro.

See also logic, anytrue.

2.8 anova

Keywords: glm, anova
Usage: anova([Model] [,print:F or silent:T, fstats:T, pvals:T, coefs:F, unbalanced:T, marginal:T])

anova(Model) computes and prints an ANOVA table for the linear model in the CHARACTER variable Model.

Examples (y a REAL vector, a and b factors, x a variate):

anova("y = a")	One-way ANOVA of y
anova("y = a+b")	Two-way ANOVA of y with no interaction
anova("y = x+a+b+a.b")	Two-way analysis of covariance of y with interaction and covariate x
anova("{log10(y)} = {sqrt(x)}+a")	One-way analysis of covariance of log10(y) with covariate sqrt(x)

See topic models for more information on how to specify Model.

anova(Model,weights:Wts) does an analysis using weighted least squares. Wts must be a REAL vector with no negative elements, with the same length as the response vector.

When you omit Model (anova() or anova(,...)), the model used by the most recent GLM command such as anova(), regress() or poisson() is used. If the previous GLM command was regress(), no new computations are done. The ANOVA table is based on what was previously computed. When there haven't been previous GLM commands, but CHARACTER variable STRMODEL exists, anova() uses STRMODEL as Model. See glm.

Side effect variables created are RESIDUALS, HII, DF, SS, DEPVNAME, TERMNAMES, and STRMODEL. When weights are specified, RESIDUALS = Response - Fitted and WTDRESIDUALS = sqrt(Wts)*RESIDUALS is an additional side effect vector. You should use WTDRESIDUALS rather than RESIDUALS in residual plots or other diagnostic procedures.

When there are no variates in Model and MacAnova recognizes that the data are from a balanced design, the analysis is by a fast method which uses marginal totals, quite analogous to the usual hand computations for a balanced analysis of variance. Otherwise, the analysis is done by explicitly constructing the design matrix and doing modified Gram-Schmidt orthogonalization. If weights are supplied, the design is never considered to be balanced. See below for what designs are considered to be balanced. You can force an unbalanced computation by 'unalanced:T'.

Other Keywords

Keyword phrase	Default	Meaning
unbalanced:T	F	Forces use of the computational method used

		for unbalanced designs even when the model is balanced. This allows full use of modelinfo() after a balanced analysis.
print:F	T	Suppress all output except warning and error messages. Side effect variables are set.
silent:T	F	Suppress all output except error messages. Side effect variables are set.
fstats:T	F	Print F-statistics and P values. The denominator for each F is the mean square from the next following term with name "ERROR1", "ERROR2", ...
pvals:T	F	Print P values based on F-distribution; default is T with fstats:T
coefs:F	T	Suppresses the computation of coefficients and quantities needed for their standard errors. After use of coefs:F, coefs() and secoefs() are disabled. coefs:T is not legal with marginal:T
marginal:T	F	Specifies that SS are computed marginally. When there are no empty cells, and sometimes when there are, the computed SS are equivalent to SAS Type III SS. See topic glm for details. marginal:T is not legal with coefs:F.

For non-balanced designs, unless marginal:T is an argument, sums of squares are computed sequentially and an advisory message to that effect is printed. This means that, in an unbalanced ANOVA, to get all the sums of squares useful for testing hypotheses, you may need to run anova() several times, with the terms in the model in different orders. For example, the A main effect sum of squares in a two way unbalanced ANOVA is the sum of squares for 'a' from anova("y=b+a") and the B main effect sum of squares is the sum of squares for 'b' from anova("y=a+b").

In many cases, use of marginal:T can simplify things. For example the A and B sums of squares produced by anova("y=a+b",marginal:T) are the A sums of squares from anova("y=b+a") and the B sums of squares from anova("y=a+b").

If the previous GLM command was regress() the behavior of anova() with Model missing is slightly modified -- it uses the results from the previous computation instead of computing things afresh, even if the variables in the previous model have been changed or deleted. Specifically, any factors in the model are treated as variates and, if the previous GLM command was regress() with weights specified by keyword 'weights' or 'wts', the entries in the ANOVA table pertain to the weighted regression. Thus, for example, even if a, b, and c are factors, the commands

```
Cmd> regress("y=a+b+c",weights:w); anova()
```

print a summary of the weighted multiple regression, followed by an weighted regression ANOVA table with 1 degree of freedom for each of a, b and c. Contrast this with anova("y=a+b+c") which computes an unweighted factorial ANOVA with no interactions.

MacAnova recognizes balance only in two cases: (1) the design is completely balanced, that is, all cells have the same number of cases; and (2) the design is a balanced main effect design such as a Latin square. If there are any missing values, the design is treated as unbalanced, even if the non-missing cases represent a balanced design.

See also coefs(), cellstats(), contrast(), factor(), fastanova(), modelinfo(), predtable(), regress(), secoefs(), xvariables().

2.9 anovapred

Keywords: glm, anova
Usage: anovapred(a,b,...), a, b, ... all the factors in STRMODEL

anovapred(a,b,...), where a, b, ... are all the factors in the most recent GLM model, computes the fitted (predicted) value, the standard error of estimation, and the standard error of prediction for each cell. The result is a structure with components 'estimate', 'SEest' and 'SEpred', each of which is a vector, matrix, or array with dimensions derived from the number of levels of a, b, ... It uses side effect variables DEPVNAME, RESIDUALS, and HII.

If the most recent GLM model was manova() with a p-dimensional dependent variable, each component will have an extra dimension of size p.

If the most recent GLM model included variates (non-factors), or if you do not include all factors in the argument list, the results will probably be wrong, although no warning message will be printed.

anovapred is implemented as a pre-defined macro.

See also predtable(), regpred(), glm.

2.10 anymissing

Keywords: missing values, null variables
Usage: anymissing(x), x REAL, LOGICAL, or CHARACTER, returns True or False

anymissing(x) returns the value True if x contains any missing values

```
and the value False otherwise.  x must be a vector, matrix, or array.
If x is CHAR, anymissing(x) is True if and only if any string in x is
empty ("").

If x is a NULL variable, anymissing(x) returns the value False.

anymissing(Str), where Str is a structure, returns a structure whose
non-structure components parallel those of Str, but are LOGICAL scalars,
indicating whether the corresponding component of Str contains any
missing values.  To test whether any component of a structure Str
contains any missing values, use if(sum(vector(anymissing(Str))) !=
0){...}.

Examples:
   anymissing(vector(1,3,5,?)) and anymissing(vector("A", "B", ""))
both return True.
   anymissing(vector(1,3,5,7)) and anymissing(vector("A", "B", "C"))
both return False.
   anymissing(structure(a1:vector(1,?),a2:"Hello")
returns structure(a1:T,a2:F).

See also ismissing().
```

2.11 anytrue

Keywords: logical variables, syntax
Usage: anytrue(arg1,arg2,...,argm), all arguments LOGICAL scalars

```
anytrue(a1,a2,...,am) is equivalent to a1 || a2 || ... || am, except
that no arguments are evaluated unnecessarily, that is, it evaluates no
arguments after the first true one.  All arguments must be LOGICAL
scalars.

Example:
   if(anytrue(!isscalar(x),!isreal(x),x <= 0, x != floor(x))){
      error("ERROR: x is not positive integer")
   }

The apparently more natural way to do the same thing
   if(!isscalar(x) || !isreal(x) || x <= 0 || x != floor(x)){
      error("ERROR: x is not positive integer")
   }
would not do what you want for a non-REAL x since an attempt would be
made to evaluate floor(x), which is illegal for non-REALs.

anytrue is implemented as a pre-defined macro.

See also logic, alltrue
```

2.12 arginfo fun

Keywords: general, control
Usage: Type help(file:"Userfun.hlp",user fun) for information on the structure of user functions. Type help(file:"Userfun.hlp",callback fun) for information on the structure of user functions making "call backs" to MacAnova. Type help(file:"Userfun.hlp",arginfo fun) for information on how to enable automatic checking of arguments to a user function.

```
This topic is now in file Userfun.hlp.  Type
  help(file:"Userfun.hlp", arginfo fun)

It provides a brief introduction to the form of an arginfo function,
that is, an externally compiled function to be called by MacAnova to
obtain information about the arguments expected by a user function.

Some other useful entries in Userfun.hlp are arginfo fun and
callback fun.  Type
  help(file:"Userfun.hlp","**")
for a complete list of entries.
```

2.13 arithmetic

Keywords: syntax, operations, missing values
Usage: a + b, a - b, a * b, a / b, a %% b, a^b, -a, +a
a <-+ b, a <-- b, a <-* b, a <-/ b, a <-%% b, a <-^ b

Arithmetic operators	Precedence	Meaning
a + b	9	Addition (sum of a and b)
a - b	9	Subtraction (difference of a and b)
a * b	10	Multiplication (product of a and b)
a / b	10	Division (a divided by b)
a %% b	10	Modular division (see below)
-a	12	Unary minus (negative of a)
a ^ b or a ** b	13	Exponentiation (a to the b-th power)

```
(Level 11 is the precedence level of matrix multiplication; see
matrices.)

These are all element-wise operations: If a and b are vectors, matrices,
or arrays with identical dimensions, then c <- a OP b, computes a result
of the same size and shape such that c[i,j,...] is a[i,j,...] OP
b[i,j,...], where OP is one of these operators.  Similarly (-a)[i,j,...]
is -(a[i,j,...]).  See below for how they operate when a and b do not
have identical dimensions.

Modular division x %% y computes the non-integral part of x / y or zero
if y exactly divides x.  It is implemented as x %% y = x - y*floor(x/y).
```

In particular, 17 %% 4 is 1, -17 %% 4 is 3, and -17 %% -4 is -1. a %% 0 is always MISSING.

When a is not zero, a / 0 yields has value MISSING. However, 0 / 0 has value 0. This can be a useful convention when a and b have the same pattern of zero elements.

If p is not an integer, a^p (or a**p) is defined to be sign(a)*abs(a)^p and a warning message is printed when a < 0. 0^p is zero when p > 0 or MISSING when p < 0. a^0 is always 1, even when a = 0.

If either any elements of a and/or b are MISSING, so is the corresponding element of a OP b and a warning message is printed.

If the result is too large a number to be represented in the computer (for example, 1e300/1e-300), the result is set to MISSING and a warning message printed.

Binary operators '+', '-', '*', '/', and '%%' associate from left to right. For example, a - b - c means (a - b) - c and a/b/c means (a/b)/c. Exponentiation ('^' or '**') associates from right to left, that is a^b^c is a^(b^c), not (a^b)^c. The arithmetic assignment operators such as '<-+' also associate from right to left, that is, a <-+ b <-+ c is interpreted as a <-+ (b <-+ c).

When there is more than one operator in an expression, operators with higher precedence as specified in the table above are evaluated before operators with lower precedence. You may use parentheses to group terms and change the order of evaluation. See topics logic, matrices and bit operations for precedence levels for other operations.

Examples:

Expression	Interpretation	Value	Explanation
4*3-2	(4*3) - 2	12 - 2 = 10	* has higher precedence than -
3*2^4	3*(2^4)	3*16 = 48	^ has higher precedence than *
(3*2)^4		6^4 = 1296	Parentheses change evaluation order
-2^4	-(2^4)	-16	^ has higher precedence than prefix -
(-2)^4		16	Parentheses change evaluation order
30/5/2	(30/5)/2	3	/ associates to left
3^2^4	3^(2^4)	43046721	^ associates to right

See topic logic for information on comparison operators '==', '<', '>', '<=', and '>=' and logical operators '&&', '||', and '!'.

See topic bit operations for information on operators '%&', '%|', '%^', and '%!' whose operands are integers considered as sets of bits.

Behavior of arithmetic, logical, and bit operations when operands differ in size.

Scalar operand:
A scalar operand (single number, all dimensions 1) is combined or compared with all the elements of the other operand. For example, x - 2 subtracts 2 from each element of x and x == 2 compares every element with 2.

Column vector operand and matrix operand:
When a column vector of length m (m by 1 matrix or vector of length m) is combined or compared with a m by n matrix, it is combined or compared with each column of the matrix to yield a m by n matrix. For example, if a is 3 by 6, run(3) + a adds 1 to row 1, 2 to row 2, and 3 to row 3, and a != vector(1,1,2) compares elements in rows 1 and 2 with 1 and elements in row 3 with 2.

Row vector operand and matrix operand:
When a 1 by n matrix (a row vector) is combined or compared with a m by n matrix, the row vector is combined or compared with each row of the matrix. For example, if x is a matrix

```
xbar <- sum(x)/nrows(x); resid <- x - xbar
```

subtracts the average of the rows of x from every row, since sum(x) computes a row vector with the same number of columns as x. This would not work if xbar were computed as xbar <- describe(x,mean:T) because describe() computes it as a vector, not a row vector. However, in this case, resid <- x - xbar' would work.

Row vector operand and column vector operand:
When a column vector of length m is combined or compared with a row vector of length n, the result is the m by n matrix obtained by combining each element of the column vector with each element of the row vector, what might be called an outer product, outer sum, etc. Thus run(2)/run(3)' is the matrix

```
                1.000 0.500 .3333
                2.000 1.000 .6667
```

and run(2) <= run(3)' is the matrix

```
                True  True True
                False True True
```

Missing values in arithmetic and logical expressions yield missing results and a warning message is printed (but see setoptions() option 'warnings').

LOGICAL variables and constants may be used in arithmetic expressions and comparisons with True and False equivalent to 1 and 0, respectively. In particular 1*w converts a LOGICAL vector, matrix, or array w to a numerical vector, matrix or array.

If one of the operands is a structure, each of its components is combined with the other argument, following the same rules of compatibility as above, producing a structure with the same shape as the structure argument. If both arguments are structures, they must have the same shape and the corresponding components are combined.

There are also arithmetic assignment operators '<-+', '<--', '<-*',

'<-/', '<-^', '<-**', and '<-%%' that are useful for modifying a variable. They are best illustrated by example.

```
a <-+ 3      is equivalent to a <- a + 3
a <-%% b     is equivalent to a <- a %% b
a <-^ -1     is equivalent to a <- a^(-1)
```

To avoid syntactic ambiguities, '<-+' and '<--' must be followed by at least one space so that, for example 'a <-- 3' means 'a <- a - 3' instead of 'a <- -3'. These operators may not be used to modify parts of a matrix. For example, a[1,2] <-/ 3 is illegal. Use a[1,2] <- a[1,2]/3.

See also logic, structures, syntax, bit operations.

2.14 array

Keywords: variables, combining variables, character variables

Usage: array(x,n1,n2,...[,KeyPhrases]) or array(x,dimVec [,KeyPhrases]), x REAL, LOGICAL or CHARACTER, n1, n2, ... positive integers or dimVec a vector of positive integers, KeyPhrases can be labels:structure(lab1,lab2,...) and/or silent:T, where lab1, lab2, ... are CHARACTER scalars or vectors.

array(x,dimVec) takes the data in x, ignoring the dimensions of x, and makes an array containing that data with dimensions as given in the REAL vector dimVec, all of whose elements must be positive integers. x can be REAL, LOGICAL, or CHARACTER.

array(x,n1,n2,...) is equivalent to array(x,vector(n1,n2,...)), if n1, n2, ... are REAL scalars or vectors. Most usually n1, n2, ... are scalars.

There must be exactly as many elements in x as the product of the elements of dimVec or of vector(n1, n2, ...). The data from x are entered with the leftmost dimensions varying fastest and the rightmost varying slowest. Thus array(run(20),vector(5,4)) and array(run(20),5,4) are equivalent to matrix(run(20),5).

If x is a matrix or array, its dimensions are ignored, that is, array(x,dimVec) is identical to array(vector(x),dimVec).

array(x) is equivalent to array(x,dim(x)), that is, it returns a variable identical to x.

You can specify coordinate labels for the output using keywords labels. See topic labels for details.

See also matrix(), matrices, subscripts.

2.15 asciisave

Keywords: files, general, output
Usage: asciisave(FileName [,all:T, v335:T, nulls:F, options:F]) asciisave() repeats previous save() or asciisave() with same options

asciisave(FileName) saves the MacAnova "workspace", that is, all the current variables and option values, in a file with name given in the quoted string or CHARACTER variable FileName. On versions with windows (Macintosh, Windows, Motif), FileName can be "", in which case you will be prompted for the file name. The file written is an ASCII coded text file which should be readable by restore() on any computer on which MacAnova runs.

If FileName is omitted and a previous asciisave() or save() was executed, the same file will be used as before. Moreover, if the previous save() specified a older version (v24:T, v31:T, or v335:T), the same option will be used, unless explicitly changed. If there was no previous save() or asciisave(), omitting the file name is an error.

asciisave(FileName, var1, var2, ...) saves only variables or macros var1, var2, ... on the file. If any of the variables saved is specified in keyword form, the keyword is used for the name. The items saved can be restored without deleting everything by restore(FileName,delete:F).

asciisave() also saves the current time and date. These are reported by restore() when the workspace or variables are restored.

In addition, asciisave() saves information about the computer MacAnova is running on and the compiler it was compiled with as well as information about the internal representation of linear models. This information is used by restore() to determine what information is safe to restore and what is not.

asciisave(FileName,ascii:F) is equivalent to save(FileName), that is the file written will have a binary format.

asciisave(FileName,v24:T [, var ...]) or asciisave(FileName,old:T [, var ...]) saves in the format recognized by versions 2.4x and earlier of MacAnova. Keywords all and options are ignored and options are not saved.

asciisave(FileName,v31:T [, var ...]) saves in the format recognized by versions 3.0 and 3.1x of MacAnova.

asciisave(FileName,v335:T [, var ...]) saves in the format recognized by version 3.35.

See save() for information on keywords 'all', 'null' and 'options'.

asciisave() differs from save() in that asciisave() saves the information in the form of a "text" file that can be transferred between

different types of computers. Files created by asciisave() are often bigger than the corresponding file created by save(). On a Macintosh, the actual type is 'Sasc' rather than 'TEXT'.

The file produced by asciisave() consists of many short lines. All the characters written are printable ASCII characters (CR and space through ~), with any other characters in escaped octal format ('\t' for TAB). The file can be printed, viewed in an editor, or sent by E-mail. It cannot be edited safely without specialized knowledge of the actual format used.

See also restore(), files

2.16 asLong

Keywords: transformations, variables
Usage: asLong(x), x REAL with no MISSING values and with integer values between -2147483647 and 2147483647.

asLong(x), where x is REAL returns a LONG variable the same size and shape as x, but with all of its elements represented as integers instead of floating point values. All the elements of REAL scalar, vector, matrix or array x must be exact integers with values between -2147483647 and 2147483647 = 2^31-1.

The only use at present for asLong() is to create a long integer argument to a user function called by User(). When the argument is returned it is "coerced" to an equivalent REAL variable. Thus, for example, User("foo", result:asLong(20)) will return a REAL integer scalar value.

asLong(x) is also legal as an argument to print() and write(). For example, print(asLong(vector(1,3,5,2))) produces the same output as print(vector(1,3,5,2)).

When assigned (y <- asLong(x)), a LONG variable is "coerced" to a ordinary REAL variable. For example, a <- asLong(vector(1,3,5,2)) has the same effect as a <- vector(1,3,5,2).

See also User(), variables.

2.17 assignment

Keywords: syntax
Usage: a <- b assigns value of b to a. a <-+ b assigns a+b to a and similarly for a <-- b, a <-* b, a <-/ b, a <-%% and a <-^ b.

You can assign values to a variable using the left pointing arrow '<-' made up of the two characters "less than" and "minus". For example, 'foo <- 5' assigns the value 5 to the variable foo. If foo did not previously exist, it is created; otherwise, its previous value is discarded and foo is re-defined. An expression of the form 'x <-3', say, is always interpreted as 'x <- 3' rather than as 'x < -3'. If you want the latter,. be sure to put a space before '-3'.

The value of an assignment command is the value of the variable after the assignment. For example, 'exp(x <- 4)' yields the value exp(4) at the same time as assigning 4 to variable x and 'x <- y <- 4' assigns 4 to both x and y.

If no assignment is made in a command, the value, if any, of the command will be printed except when it is part of a compound command or unless its value is "invisible". Thus simply typing 'foo' prints the value of variable foo while typing '_foo' ("invisible" variable) prints nothing

Because the value of a compound command is the value of its last command, that value is printed unless the compound command is itself nested in a larger compound command. Thus, because an assignment command has a value, '{x <- 3}' not only assigns the value 3 to x put also prints the number 3, although 'x <- 3' by itself prints nothing. For this reason, it is a often a good idea to terminate each compound command with ';;', as in '{x <- 3;;}'. Of course, this is a bad idea if you want the final value to be printed or if you are assigning the value of the entire compound statement to a variable.

There are also several arithmetic assignment operators: <-+, <--, <-*, <-/, <-%% and <-^. For example, a <-* b is equivalent to a <- a*b and a <-^ b is equivalent to a <- a^b. '<--' and '<-+' require a following space. See arithmetic.

2.18 atan

Keywords: transformations

Usage: atan(x) or atan(x,y), x and y REAL or structures with REAL components, y the same size and shape as x; value in radians (default), cycles, or degrees as set by option "angles"

atan(x) transforms the elements of REAL vector, matrix, or array x to arctangents. If x is a structure with components x1,...,xm, atan(x) is a structure with components atan(x1),...,atan(xm).

atan(x,y) computes theta = arctan(x/y), with the result in the appropriate quadrant, where x and y must be REAL vectors, matrices, or arrays with the same dimensions. Specifically, theta is chosen so that sin(theta) has the same sign as x, cos(theta) has the same sign as y and tan(theta) = x/y. The values of theta are in radians, degrees, and cycles as specified by option 'angles' (default is radians).

atan(x,y) is also defined when x and y are both structures with the same number of components, say x is structure(x1,...,xm) and y is structure(y1,..., ym) . The result is what would be produced by structure(atan(x1,y1),...,atan(xm,ym)).

atan(x) can also be used when x is a CHARACTER variable and atan(x,y) can be used when both x and y are CHARACTER variables with matching dimensions. In that case the result is a CHARACTER variable describing the transformation of the arguments. For example, atan(vector("X1", "X2")) returns vector("atan(X1)","atan(X2)") and atan("X","Y") returns "atan(X,Y)". This feature may be useful in creating new labels for a transformed variable.

See also transformations, setoptions(), structures, labels.

2.19 autoreg

Keywords: time series
Usage: autoreg(Phi,A [,reverse:T, limits:vector(i1,i2), start:startVals]), REAL vector Phi, REAL vector or matrix A

autoreg(phi,a) applies the autoregressive operators specified by the columns of the REAL matrix phi to the columns of the REAL matrix a. If ncols(phi) = 1, phi is applied to every column of a and if ncols(a) = 1, each column of phi is applied to a. The result is a matrix with nrows(a) rows and max(ncols(phi), ncols(a)) columns. If both phi and a have more than one column, they must both have the same number of columns.

Specifically, assuming for simplicity that both phi and a are vectors so that the result x is a vector,

```
    x[i] = a[i] + sum(phi[k]*x[i-k],1<=k<=nrows(phi)),
```

with x[1] taken to be 0 for 1 < 1.

A common usage is autoreg(1,x), where x is a vector or matrix. This computes the partial sums x[1,], x[1,]+x[2,], ..., sum(x). A useful macro might be defined by

```
    parsum <- matrix("autoreg(1,$1)")
```

autoreg(phi,a,reverse:T) applies the autoregressive operator in reverse:

```
    x[i] = a[i] + sum(phi[k]*x[i+k],1<=k<=nrows(phi)),
```

with x[1] = 0 for 1 > nrows(a).

autoreg(phi,a,limits:vector(i1,i2),start:StartVals [,reverse:T]) is the same except that x[i] is computed as just described only for i1 <= i <= i2, with the remaining values copied before the computation from rows 1 to i1-1 and rows i2+1 to nrows(a) of matrix StartVals. Thus the values in rows 1 through i1-1 of StartVals serve as "starting values" for the autoregressive operator. If reverse:T is an argument, rows nrows(a)

through i2+1 serve as starting values. StartVals must be the same size and shape as a. This feature is useful for generating out of sample forecasts or "backcasts".

Examples:
```
autoreg(phi,rnorm(400))[-run(100)] generates an autoregressive series
  with normal innovations, discarding the first 100 values to avoid
  transients.
autoreg(phi,matrix(rnorm(4000),400)[-run(100),] generates 10
  independent autoregressive series at once.
autoreg(hconcat(phi1,phi2),rnorm(400))[-run(100)] generates 2
  autoregressive series with different coefficients but the same
  innovations
autoreg(vector(1,1),padto(1,20)) computes the first 20 Fibonacci
  numbers.
```

autoreg() is the inverse of movavg() and vice versa, in that
`autoreg(phi,movavg(phi,x))` and `movavg(phi,autoreg(phi,x))`
both reproduce x, except for rounding error.

See also movavg().

2.20 batch

Keywords: syntax, control, files
Usage: `batch(fileName [,echo:T or F, prompt:string])`,
`CHARACTER scalars fileName and string`

batch(FileName) executes the commands in the file with name given in the quoted string or CHARACTER variable FileName. It must be the last command in a line or a compound command surrounded by '{' and '}' and must not be in a loop.

In a version with windows (Macintosh, Windows, Motif), if FileName is "" you will be prompted to enter the file name in a dialog box.

Lines of the file are read sequentially and executed as if they were typed at the keyboard. Normally, each line is printed with the file name as prompt before it is executed. You can suppress this by using keyword phrase echo:F (see below), or by previously executing setoptions(batchecho:F) (see setoptions()).

The batch file can contain any sequence of MacAnova commands, including other batch() commands unless they attempt to read from a batch file currently in use. Here is an example of a short batch file designed to do cubic regression of variable y on variable x and do a plot of residuals against x (see also regress() and plot()):

```
xsq <- x
xcub <- x * xsq
regress("y=x + xsq + xcub")
```

```
plot(x, RESIDUALS, title:"Cubic regression residuals vs x")
```

When an error occurs, the default behavior is to terminate all current batch() commands. You can use setoptions(errors:N), where N is a positive integer to increase the number of errors tolerated before termination. See setoptions() for details.

batch(FileName,echo:F) works the same as batch(FileName) except the prompts and lines read from the file are not printed. This status is inherited by batch files invoked from within a batch file. You can use setoptions(batchecho:F) to set the default behavior of batch() so that lines will not be echoed.

batch(FileName,echo:T) forces the printing of prompts and commands, even if option 'batchecho' has been set False (see setoptions()).

batch(FileName,prompt:Prompt), where Prompt is a quoted string or CHARACTER scalar, forces echoing, with command lines starting with Prompt instead of the file name. If the batch file contains a setoptions(prompt:newPrompt) command, newPrompt overrides Prompt. A subsequent setoptions(default:T) in the file, restores Prompt.

On a Macintosh, selecting item Open Batch File on File menu is equivalent to typing 'batch("")' except that it first erases everything after the prompt.

See also launching.

2.21 bcprd

Keywords: matrix algebra, glm

Usage: bcprd(x), REAL matrix x bcprd(x1, x2, ...), x1, x2, ... REAL matrices with the same number of rows.

bcprd(x) where x is a matrix computes a "bordered" cross product matrix containing the means of the columns of x and mean-corrected sums of squares and products of the columns of x.

bcprd(x1,x2,...,xm) yields the same result as bcprd(hconcat(x1,x2,..., xm)) when x1, x2, ... are all REAL matrices with the same number of rows.

Specifically, when x is an n by p matrix, bcp <- bcprd(x) sets bcp to a p+1 by p+1 matrix, where

```
  bcp[1,1]   = 1/n
  bcp[-1,1]  = a column vector containing the sample mean xbar
  bcp[1,-1]  = a row vector containing xbar'
  bcp[-1,-1] = the p by p matrix of mean-corrected sums of squares and
               products of the columns of x
```

bcprd(x) is mathematically equivalent to

{@TMP <- hconcat(rep(1,nrows(x)),x); swp(@TMP %c% @TMP,1)}.
However, the use of bcrpd() is preferred to the illustrated use of swp() since it uses a numerically stable algorithm to compute the corrected sums of squares and products.

If all the arguments of bcprd() have labels, so will the result of bcprd() with both row and column labels taken from the column labels of the arguments.

See also swp().

2.22 bin

Keywords: categorical data, summary statistics
Usage: bin(x,Bnds [,silent:T,leftendin:T]), x a REAL matrix, Bnds REAL vector, Bnds[k] < Bnds[k+1]
bin(x,vector(binEdge,binWidth) [,leftendin:T]), binEdge and binWidth > 0 REAL scalars
bin(x,nbins, [leftendin:T]), nbins positive integer.
bin(x [,leftendin:T])

bin(x,Bnds), where x is a REAL vector or matrix and Bnds is a REAL vector of class interval boundaries, counts the number of values v in each column of x in each class interval, that is the number such that Bnds[k] < v <= Bnds[k+1], k = 1, ..., length(Bnds)-1. This means any value equal to the right end of an interval is counted in that interval. It is an error if length(Bnds) = 1 or Bnds[k] >= Bnds[k+1] for any case. The case where length(Bnd) = 2 is treated specially; see below.

The value of bin(x,Bnds) is a structure with components 'boundaries' and 'counts'. If length(Bnds) > 2, then bin(x,Bnds)$boundaries is identical to Bnds, and bin(x,Bnds)$counts is a vector of length nbins = length(Bnds) - 1 or a nbins by ncols(x) matrix with counts[i,j] = (number of values in column j in class interval i).

Note that bins() counts each column of x separately using the same boundaries for each column. If any element of x is <= Bnds[1] or > Bnds[nbins+1], it is not included in the count and a warning message is printed.

bin(x,Bnds,silent:T) does the same except any warning messages are suppressed.

bin(x,Bnds,leftendin:T [,silent:T]) does the same except a value v is counted in interval k if Bnds[k] <= v < Bnds[k+1], that is, a value equal to the left end of an interval is counted as being in the interval. If v < Bnds[1] or v >= Bnds[nbins+1], it is not counted.

bin(x,vector(b,width) [,leftendin:T]), where b and width are scalars, does the same, except vector Bnds is computed so that its values are

```
equally spaced with the form b + k*width, where k is an integer, and
such that Bnds[1] < min(vector(x)) and Bnds[nbins+1] >= max(vector(x)).
For example, bin(x,vector(.5,1)) would use bins of width 1 with
boundaries at half integers, that is centered at integers.

bin(x,nbins [,leftendin:T]) does the same, except that vector Bnds of
length nbins+1 is computed such that the values are equally spaced and
Bnds[1] < min(vector(x)) and Bnds[nbins+1] >= max(vector(x)).

bin(x) is equivalent to bin(x, ceiling(log(nrows(x))/log(2))+1), that is
the number of bins is approximately log2(nrows(x)).

Among other things, the output of bin() can be used to draw a histogram
or to compute a chi-squared test of goodness of fit of a sample to a
theoretical distribution.
```

2.23 bit operations

Keywords: operations, glm, missing values

Usage: `a %| b, a %^ b, a %& b and %! a, where a and b are REAL or structures with REAL components with integer elements >= 0 and <= 4294967295, nbits(x)`

```
There are 4 operators for working with integers considered as the sets
of 32 bits specified by their binary representations.

Bit Operation  Precedence  Meaning
    a %| b         1       Bitwise Or (OR)
    a %^ b         2       Bitwise Exclusive Or (XOR)
    a %& b         3       Bitwise And (AND)
      %!a          4       Bitwise Complement (COMPL)

If an operand x is not an integer or x < 0 or x > 4294967295 = 2^32-1,
the result of any of these operators is MISSING.

For '%&', a bit of the result is 1 if and only if the corresponding bits
in the operands are both 1.
  Example: 25 %& 19 is 17 because 11001b AND 10011b is 10001b

For '%|', a bit of the result is 1 if and only if at least 1 of the
corresponding bits in the operands is 1.
  Example: 25 %| 19 is 27  because 11001b OR 10011b is 11011b

For '%^', a bit of the result is 1 if and only if exactly 1 of the
corresponding bits in the operands are 1, that is, if the corresponding
bits differ.
  Example: 25 %^ 19 is 10  because 11001b XOR 10011b is 01010b

Operator '%!' operates on the immediately following variable considered
as a collection of 32 bits, changing 1's to 0's and 0's to 1's.
  Examples:
```

```
    %! 25 is 4294967270 since COMPL(00000000000000000000000000011001b)
       is 11111111111111111111111111100110b
    %! 0  is 4294967295 since COMPL(00000000000000000000000000000000b)
       is 11111111111111111111111111111111b
```

If an operand is LOGICAL, it is treated as having value 0 (F) or 1 (T). The result is always REAL.

If any operand is MISSING, so is the result.

These operators were introduced to be useful with the output of modelinfo(bitmodel:T). For example, 2^(i-1) %& modelinfo(bitmodel:T)[j] is non-zero if and only if the j-th term of the model contains the i-th factor or variate (assuming i <= 32).

The operators were listed above in increasing order of precedence. Moreover, they have lower precedence that all arithmetic, comparison, or logical operators.

Examples:

Expression	Interpretation	Value
17 %| 29 %^ 91 %& 11	17 %| (29 %^ (91 %& 11))	23
%!21 %| 97 %& %! 33	(%!21) %| (97 %& (%!33))	4294967274
1 %& 3 + 4	1 %& (3+4)	1
3 %^ 5 != 6	3 %^ (5 != 6)	2
1 %| 2 == 3	1 %| (2 == 3)	1
%!0 == 4294967295	%!(0 == 4294967295)	4294967295

To understand the last three examples, note that 5 != 6 is True and is interpreted as 1, and that 2 == 3 and 0 == 4294967295 are both False and are interpreted as 0. See topics arithmetic and logic.

See topic arithmetic for a description of the "shape" of the result when operands are not scalars.

See also modelinfo(), nbits().

2.24 boxcox

Keywords: transformations
Usage: boxcox(x,power), x a REAL vector or matrix, power a REAL scalar

boxcox(var,Pow) computes the Box-Cox transformation of the data in vector or matrix var. If var is a matrix, the transformation is applied to each column separately. If GM is the geometric mean of the values in a vector, boxcox(y,Pow) computes (y^Pow-1)/(Pow*(GM)^(Pow-1)) when Pow != 0, and GM*log(y) when Pow == 0. Boxcox is implemented as a macro.

See also macro and transformations.

2.25 boxplot

Keywords: plotting, descriptive statistics
Usage: boxplot(x1,x2,...,xk [,vertical:T, graphics keyword phrases]), arguments REAL vectors
boxplot(Struc, [, vertical:T, graphics keyword phrases]), Struc a structure with REAL vector components

boxplot(var1, var2, ... , vark) produces parallel Tukey boxplots for the vectors var1 through vark.

boxplot(Struc) produces parallel box plots for the components of structure Struc, all of which must be vectors.

By default boxplots are aligned horizontally.

boxplot(var1, var2, ... , vark,vertical:T) and boxplot(Struc,vertical:T) do the same except the boxplots are aligned vertically. Pre-defined macro vboxplot() which is used identically to boxplot(), makes use of the feature to make vertical boxplots.

boxplot(split(y,a) [,vertical:T]) draws parallel box plots of the data in vector y classified according to levels of factor a. See split().

boxplot(split(y) [,vertical:T]) draws parallel box plots of the data in each column of matrix y.

If option 'dumbplot' has been set False (see setoptions()), the plot will be a low resolution plot unless 'dumb:F' is an argument.

You can use keywords 'title', 'xlab', and 'ylab' to override the default labeling information. You can override the default tick positions and tick lengths using keywords 'xticks', 'yticks', 'xticklen' and 'yticklen'. You can use 'dumb:T' to specify the plot be constructed of printable characters. See topic graph keys.

See topic graph keys for information on graphics keywords.

See topic graph files for information on how to save a boxplot in a file using keywords 'file', 'new, 'ps', 'screendump', and 'epsf'.

See topics macintosh and wx for information on how to print graphs in windowed versions (Macintosh, Windows, Motif).

See also showplot(), structures, vt, tek.

2.26 break

Keywords: control, syntax
Usage: `for(i,run(n))if(x[i] < 0)break ....`
`for(i,run(n))for(j,run(m))if(x[i,j] < 0)break 2`

break is used in 'while' and 'for' loops to exit prematurely from the loop, perhaps because an error has been found. Execution resumes immediately after the '}' terminating the current 'while' or 'for' loop.

break n, where n is a positive integer exits from n enclosing loops. Thus 'break 1' is equivalent to 'break' and will exit the current loop, and 'break 2' will exit the current loop and the loop enclosing it, etc. n must be a literal integer ('1', '2', ...) and not a variable with integer value.

It is an error to use break outside of a loop or to use break n when not enclosed in at least n loops.

Examples:

```
  for(i,run(100)){... compute x ...;if(x<0){print("x < 0");break;};...;}
```

If x ever becomes negative, the 'for' loop is terminated.

```
  for(i,run(10)){for(j,run(5)){...;if(x<0){break 2}}}
```

If x ever becomes negative, both 'for' loops are terminated.

Inside a macro, break n with the appropriate value of n should always be used instead of breakall.

See also if, for, while, breakall, batch().

2.27 breakall

Keywords: control, syntax
Usage: `for(i,run(n))if(x[i] < 0)breakall ....`
`for(i,run(n))for(j,run(m))if(x[i,j] < 0)breakall`

breakall is used in 'while' and 'for' loops to exit prematurely from any and all "enclosing" 'while' or 'for' loops. Execution resumes immediately after the '}' terminating the most inclusive 'while' or 'for' loop currently in effect. breakall should normally not be used inside a macro, since if such a macro were invoked in a loop at the prompt level, breakall would exit from that loop as well as any loops in the macro. This would seldom be what you want. Instead, use break n, where n is a positive integer specifying the number of loops to exit.

It is an error to use breakall outside of a loop.

Example:

```
  Cmd> for(i,run(m)){
          for(j,run(n)){... compute x ...;if(x<0){breakall};}}
```

```
If x ever becomes negative, both 'for' loops are immediately terminated.
Using break instead of breakall would mean that only the inner loop
(for(j,run(n)){...}) would be terminated.

See also if, for, while, break.
```

2.28 breakif

Keywords: control, syntax
Usage: `for(i,run(n))breakif(x[i] < 0) ....`
`for(i,run(n))for(j,run(m))breakif(x[i,j] < 0, 2) ...`

```
breakif(Logical) is equivalent to if(Logical){break;} .
breakif(Logical,n) is equivalent to if(Logical){break n;} .

It is implemented as a pre-defined macro.  It can be used only inside a
loop and n must not exceed the number of containing loops.

Example:
  Cmd> for(i,run(length(x))){breakif(abs(x[i]) > 3)}
computes the index i of the first element on vector x to exceed 3 in
absolute value.

See also break.
```

2.29 callback fun

Keywords: general, control
Usage: Type help(file:"Userfun.hlp",user fun) for information on the structure of user functions. Type help(file:"Userfun.hlp",callback fun) for information on the structure of user functions making "call backs" to MacAnova. Type help(file:"Userfun.hlp",arginfo fun) for information on how to enable automatic checking of arguments to a user function.

```
This topic is in file Userfun.hlp.  Type
  help(file:"Userfun.hlp", callback fun)

It provides a brief introduction to the form of a user function that
makes "call backs" (executes functions internal to MacAnova).

Some other useful entries in Userfun.hlp are arginfo fun and user fun.
Type
  help(file:"Userfun.hlp","**")
for a complete list of entries.
```

2.30 cat

Keywords: variables, combining variables, character variables, null variables
Usage: `cat(x1,x2,...,xk [,KeyPhrases])` where x1, x2, ... all have the same type, REAL, LOGICAL, or CHARACTER, or are structures with components all of the same type KeyPhrases can be labels:lab and/or silent:T, where lab is a CHARACTER scalar or vector.

cat(x1, x2, ..., xk) combines (conCATenates) scalars x1, x2, ... xk into a vector of length k. Thus you can enter a small set of data by, for example,

```
x <- cat(3.5, 9.6, 2.5, 2.3, 7.7, 2.6, 6.3, 6.5, 6.6, 4.1)
```

cat() is identical to vector(). See vector() for more details on its use.

The use of cat() is deprecated -- that is, it will continue to be available for the immediate future, but at some point may be disabled. Use vector() instead.

See also vector(), vectors.

2.31 cconj

Keywords: time series, complex arithmetic
Usage: `cconj(cx)`, cx a REAL matrix representing complex data

cconj(cx) returns the complex conjugates of successive pairs of columns of the matrix cx, considered as the real and imaginary parts of complex series. The real and imaginary parts of the results are in alternating columns.

If cx has an odd number, say 2*m-1, of columns, an addional additional column of zeros is implictly added before computing the complex conjugates.

See also hconj(), hreal(), himag(), creal(), cimag().

See topic 'complex' for discussion of complex matrices in MacAnova.

2.32 ceiling

Keywords: transformations
Usage: `ceiling(x)`, x REAL

ceiling (x) rounds the elements of the REAL variable x to the next integer in the positive direction, producing a vector, matrix, or array with the same shape as x.

When x > 4503599627370495 or x < -4503599627370495, ceiling(x) is set to MISSING because of the impossibility of exact representation of integers beyond these limits. These limits may be different on some computers.

If x is a structure, so is ceiling(x). If xi is the i-th component of x, the i-th component of ceiling(x) is ceiling(xi).

Examples: ceiling(3.1416) is 4, ceiling(-3.1416) is -3 and ceiling(12) is 12.

See also floor(), round(), structures.

2.33 cellstats

Keywords: descriptive statistics, anova
Usage: cellstats(Term), Term a CHARACTER scalar of form "A.B. ...", where A, B, ... are factors in current GLM model.

cellstats(Term) computes statistics for each cell of the multiway layout indicated by the term in the CHARACTER variable Term. The term must be made of factors in the model used by the most recent GLM (generalized linear or linear model) command such as regress(), anova(), or poisson(). It omits all cases for which there is any MISSING data in the left or right hand sides of the model.

cellstats() and tabs() do almost the same thing and generally tabs() is to be preferred. If, Term is, say, "a.b.c" where a, b, and c are factors in the most recent GLM, and y is the reponse variable in the model, then cellstats(Term) is almost equivalent to tabs(y, a, b, c).

cellstats() and tabs() will differ only when (a) the response variable y is multivariate (has more than 1 column) and (b) there are MISSING data in y. cellstats() omits completely any row of y that contains any MISSING data; tabs() uses all non-MISSING data available and thus the cell count can differ among the columns of y. Except in this case, you should probably use tabs() in preference to cellstats() since tabs() has additional options and can be used independently of any GLM command. Even in this case, tabs() is preferable if you want cell statistics that use all the data.

Of course, both cellstats() and tabs() omit all cases for which any of the factors are MISSING (how could they determine the cell?).

Example:

```
  Cmd> anova("y=a+b+c+b.c") ; cellstats("b.c") # or tabs(y,b,c)
```

gives cell statistics for the b.c term.

See also glm, tabs().

2.34 cft

Keywords: time series, complex arithmetic
Usage: cft(cx [,divbyT:T]), cx a REAL matrix representing complex data

cft(cx) where cx is a REAL vector or matrix, computes the fully complex form of the discrete Fourier transforms of successive pairs of columns of cx, considered as the real and imaginary parts of complex series. The real and imaginary parts of the results are in alternating columns. Any MISSING values in cx are treated as if they were 0.

cft(cx,divbyt:T) does the same except the transform is divided by the number of rows of cx.

cconj(cft(cconj(cx),divbyt:T)) is the inverse of cft() in the sense that cx and cconj(cft(cconj(cft(cx)),divbyt:T)) are equal except for rounding error.

The largest prime factor of nrows(cx) must not exceed 29.

See topic 'complex' for discussion of complex matrices in MacAnova.

See also hft(), rft(), cconj().

2.35 changestr

Keywords: structures
Usage: changestr(Struc,name,x), Struc a structure, name a CHARACTER scalar, x a defined variable
changestr(Struct,n,x), integer n, 1 <= n <= ncomps(Struct) + 1
changestr(Struct,name:x)
changestr(Struc, -n), n a positive integer

changestr(Struc,CompName,x) makes a copy of structure Struc except that the value of component CompName is changed to x. CompName must be a quoted string or CHARACTER variable. If there is no component with name CompName, x will be added as a new component. It will have name CompName unless x is a keyword phrase

changestr(Struc,CompNumber,x), where CompNumber is a positive integer, does the same, except the component to be changed is specified by number rather than by name. If CompNumber = ncomps(Struc) + 1, a new component with value x is added. It is an error if CompNumber > ncomps(Struc)+1.

In both these usages, if x is a keyword phrase, say newName:value, the the name of the replaced or new component will be 'newName'.

changestr(Struct,name:x) does the same, replacing component 'name' with x if it exists or adding a new component 'name'.

changestr(Struc,-CompNumber) produces a new structure omitting component CompNumber. It is illegal to delete the only component of a structure.

Examples: In each of the following groupings, all the commands return the same structure:

```
  Cmd> changestr(structure(a:run(10),b:"Hello"),"a",PI)
  Cmd> changestr(structure(a:run(10),b:"Hello"),1,PI)
  Cmd> structure(a:PI,b:"Hello")

  Cmd> changestr(structure(a:run(10),b:"Hello"),"a",pi:PI)
  Cmd> changestr(structure(a:run(10),b:"Hello"),1,pi:PI)
  Cmd> structure(pi:PI,b:"Hello")

  Cmd> changestr(structure(a:run(10),b:"Hello"),3,c:"Dolly")
  Cmd> changestr(structure(a:run(10),b:"Hello"),c:"Dolly")
  Cmd> structure(a:run(10),b:"Hello",c:"Dolly")

  Cmd> changestr(structure(a:run(10),b:"Hello"),-1)
  Cmd> structure(b:"Hello").
```

See also structures, keywords, structure(), strconcat().

2.36 cholesky

Keywords: matrix algebra
Usage: cholesky(x), x a positive definite square REAL matrix with no MISSING values

cholesky(a) computes the Cholesky decomposition of the positive definite REAL symmetric matrix a, that is the upper triangular matrix r such that r' %*% r equals a. It returns a square REAL matrix of the same size as a.

cholesky(a,pivot:T) or simply cholesky(a,T) reorders the rows and columns as the computation proceeds so as to obtain the most stable computation. It returns a structure with components 'r', a REAL upper triangular matrix, and 'pivot', a REAL vector of integers describing the reordering. After result <- cholesky(a,T), result$r' %*% result$r should equal a[result$pivot, result$pivot] except for rounding error.

cholesky(a,force:Vec), where Vec is a REAL vector whose length is nrows(a), enables pivoting, but allows some control on reordering. The elements of Vec should be 1, -1, or 0, since only the signs are used. Before factoring, rows and columns of a, if any, with index j such that

Vec[j] > 0 are moved to rows and columns 1, 2, ..., (initial columns) but are not further moved. All rows and columns with Vec[j] < 0 are moved to rows and columns nrows(a), nrows(a) - 1, ..., (final columns) but are not further moved. Rows and columns, if any, with Vec[j] == 0 (pivoted columns), are free to be reordered, but will follow the initial columns and precede the final columns. Again the result is a structure with components 'r' and 'pivot'.

See also qr().

2.37 chplot

Keywords: plotting

Usage: chplot(x,y [, c] [, lines:T, impulse:T] [,graphics keyword phrases]), where x is a REAL vector or scalar, y is a REAL vector or matrix and c is a integer or CHARACTER scalar, vector, or matrix

chplot(x,y,c) makes a scatter plot of REAL vector or matrix y versus REAL vector x using plotting symbols as specified by CHARACTER or REAL vector c.

chplot(Struc,c), where Struc is a structure with at least two REAL components, is equivalent to chplot(Struc[1], Struc[2], c). Thus chplot(x,y,c) and chplot(structure(x,y),c) are equivalent. Any components beyond the first two are ignored.

chplot(graph,x,y,c) or chplot(graph,Struc,c), where graph is a GRAPH variable, draws the plot encapsulated in graph, adding to it the new information. See topic graph for details on adding information to a plot.

If c is REAL, each element c[i] must be an integer with 0 <= c[i] <= 999 and the plotting symbol will be the number centered at the plotting point.

If c is CHARACTER, up to 3 characters from each c[i] will be drawn centered at the plotting point.

Argument c may be omitted. In this case the default plotting characters are as follows:

y a vector:	The row number of an element
y a matrix with ncols(y) > 1:	The column number of an element

If keyword phrase 'lines:T' is an argument after c or y, the points will be connected by lines similarly to lineplot().

If keyword phrase 'impulses:T' is an argument after c or y, vertical lines will be drawn to the points from the x = 0 line.

If option 'dumbplot' has been set False (see setoptions()), the plot will be a low resolution plot unless 'dumb:F' is an argument.

If c has more than 1 column then you must have ncols(c) = ncols(y) and the elements in c[,j] will be used to plot y[,j], reusing the rows of c cyclically if nrows(c) < nrows(y).

If c is a vector of length ncols(y), c[j] will be used to plot all elements of the column y[,j]

Otherwise, if c is a vector with length(c) != ncols(y), c[i] will be used to plot all elements in the row y[i,], reusing the rows of c cyclically if nrows(c) < nrows(y).

If the first character of an element of a CHARACTER c has ASCII code v between 1 and 31, it designates a specially drawn character. For v = 1, 2, ..., the corresponding characters (and their codes) are diamond (1), plus sign (2), square (3), cross (4), triangle (5), star (6), dot (7), small cross (8), diamond (9), plus sign (10), and so on cyclically. You can specify these special ASCII codes using quoted strings "\1", "\2", "\3", "\4", "\5", "\6", "\7", "\10", "\11", ... ,"\17", "\20", ..., "\27", "\30", ..., "\37". The digit or digits are the octal representations of the codes. Thus, for example, "\3" represents an ASCII 3 and specifies a square, and "\10" represents an ASCII 8 and specifies a small cross.

See topic graphs for the use of a scalar or length 2 vector for x.

Use addchars(), addlines(), and addpoints() to add information to a plot.

See topic graph keys for the use of keywords title, xlab, ylab, xmin, xmax, ymin, ymax, xaxis, yaxis, dumb, height, width, add, file, linetype and thickness.

See topic graph files for information on how to save a boxplot in a file using keywords 'file', 'new, 'ps', 'screendump', and 'epsf'.

See topics macintosh and wx for information on how to print graphs in windowed versions (Macintosh, Windows, Motif).

Examples:

```
  Cmd> chplot(x,y,"*")
```

makes a plot of y vs x with "*" as plotting symbol.

Suppose x[,1] contains integers 1, 2, or 3. Then

```
  Cmd> chplot(X2:x[,2],X3:x[,3],vector("A","B","C")[x[,1] ],\
        title:"X3 vs X2")
```

and

```
  Cmd> chplot(X2:x[,2],X3:x[,3],x[,1],title:"X3 vs X2")
```

are equivalent. Both make a plot of column 3 of x against column 2, using plotting symbols "A", "B", or "C", according as the value in column 1 of x is 1, 2 or 3. Axes are labeled 'X2' and 'X3' and a title

```
is printed.
  Cmd> chplot(X:1,run(20)^(.2*run(5)'),vector(".2",".4",".6",".8","1.")',\
       ylab:"Powers of X",title:"X^.2, X^.4, X^.6, X^.8, and X",lines:T)
Here, X:1 is equivalent to X:run(20).

See also graphs, plot(), lineplot(), showplot(), addchars(), addlines(),
addpoints(), colplot, rowplot, tek, vt.
```

2.38 cimag

Keywords: time series, complex arithmetic
Usage: `cimag(cx), cx a REAL matrix representing complex data`

```
cimag(cx) computes the imaginary part of the fully complex matrix cx.
Thus cimag(matrix(run(10),5)) is vector(6,7,8,9,10).

See also hconj(), cconj(), hreal(), himag(), creal().

See topic 'complex' for discussion of complex matrices in MacAnova.
```

2.39 clipboard

Keywords: syntax, character variables, input, output
Usage:
```
CLIPBOARD <- x or x <- CLIPBOARD or
vecread(string:CLIPBOARD)
x <- fromclip([ncols]) or toclip(x), using pre-defined
macros fromclip and toclip
SELECTION <- x or x <- SELECTION (Motif only)
```

```
A special CHARACTER variable CLIPBOARD is always defined.  When used in
an expression or as an argument to a function, CLIPBOARD behaves just
like any other variable.  It can be printed, written to a file, or
assigned to a regular variable.

In the Macintosh, Windows and Motif versions, CLIPBOARD allows direct
access to the system Clipboard.  In other versions, assigning to
CLIPBOARD translates the right hand side to a CHARACTER variable but
does nothing more.

When CLIPBOARD is assigned to, its behavior is special.  When x is a
CHARACTER scalar, CLIPBOARD <- x makes CLIPBOARD identical to
x. However, when x is REAL or LOGICAL, CLIPBOARD <- x does not set to x
itself, but to a scalar CHARACTER representation of x.  Specifically,
when x is REAL or is a CHARACTER vector, matrix, or array, the value of
CLIPBOARD is what would be produced by
   CLIPBOARD <- paste(x,multiline:T,missing:"?",sep:"\t",\
                linesep:"\n", format:"0.17g")
```

After CLIPBOARD <- x, the contents of variable CLIPBOARD (and of the Clipboard in the Macintosh, Windows and Motif versions) can be summarized as follows:

If x is a vector of length N, CLIPBOARD will contain N lines, with line i containing a CHARACTER representation of x[i].

If x is a matrix, CLIPBOARD will contain nrows(x) lines, with line i containing CHARACTER representations of x[i,j], j=1,...ncols(x), with the elements of each row separated by tab characters.

If x is an array with 3 or more dimensions greater than 1, it is treated as if it were a matrix with nrows(x) = first dimension > 1.

See topic paste() for more information on paste(x,multiline:T,...).

Since MISSING is coded as '?', after CLIPBOARD <- x, where x is REAL vecread(string:CLIPBOARD), should produce vector(x), including MISSING values.

Two pre-defined macros, toclip and fromclip, are useful when working with CLIPBOARD. In particular, toclip allows for different coding of MISSING and user specified field separators. See topics fromclip and toclip.

In the Macintosh, Windows and Motif versions, the value of CLIPBOARD is whatever text the system Clipboard currently contains. This may differ from what was most recently assigned to CLIPBOARD because of the use of items on the Edit menu in MacAnova or another program. This feature allows easy importing of data from other programs, especially from spreadsheets. See fromclip.

Similarly, in the Macintosh, Windows and Motif versions you can easily export data from MacAnova to another application like a spreadsheet by assigning the value of a variable to CLIPBOARD. See toclip.

You can free up the memory used by the contents of CLIPBOARD by delete(CLIPBOARD). This has no effect on any system Clipboard.

You probably should not use CLIPBOARD as a name for a structure component or as a keyword on a computer with an actual Clipboard, as every mention of CLIPBOARD, even in contexts like str$CLIPBOARD or CLIPBOARD:T, refreshes special variable CLIPBOARD with stuff from the Clipboard.

In Motif, selecting text with the mouse provides another method of communicating between programs. Briefly, if you select text and then click in a window using the middle button on the mouse, what was selected is inserted there. The Motif version of MacAnova has special CHARACTER variable SELECTION which is connected to the current selection in the same way CLIPBOARD is connected to the clipboard. Immediately after

```
  Cmd> SELECTION <- x
```
clicking in a window with the middle button "pastes" x into the window. Similarly, if text is selected in a window,
```
  Cmd> charx <- SELECTION
```
creates a CHARACTER variable charx containing the text; if the text is numerical data,
```
  Cmd> x <- vecread(string:SELECTION)
```
creates a REAL vector x. Use of this feature is somewhat tricky, since clicking in a window can change the selection. If you assign something to SELECTION, you should retrieve it with a middle button click before doing anything else. And if you want to assign from or read from SELECTION, you should type the command without a terminating Enter, then select what you want to copy, click on the frame of the MacAnova window, and then press Ctrl+E followed by Return to execute the command.

On a Macintosh or Windows computer, putting something on the Clipboard whether by Copy or Cut on the Edit menu or by assigning to CLIPBOARD, deletes any text that was already there. In Motif, it goes at the end of a list of items previously put on the Clipboard. CLIPBOARD and Paste on the Edit menu references the last item in the list. You can edit the Clipboard and delete obsolete items with X program xclipboard.

See also vecread(), matread(), macroread().

2.40 cluster

Keywords: multivariate analysis

Usage:
```
cluster(x [, nclust:n, standard:F, method:name,
keep:charVec, print:T, tree:T or F, classes:T or
F, reorder:T]), x a REAL matrix, name a character
scalar (one of "single", "complete", "average",
"ward", "mcquitty", "centroid", or "median"), charVec
a CHARACTER vector with elements "all", "classes",
"criterion", or "distances"
cluster(dissim:d [, ...]), d a square REAL matrix
cluster(similar:s [, ...]), s a square REAL matrix
```

cluster(x) performs a hierarchical cluster analysis of cases (rows) of the data matrix x. The default method is average linkage and the default maximum number of clusters described in the output is 9. It produces a table of cluster membership with one line per case and a dendrogram, with the join points labeled with the value of the criterion used. There must be at least 2 rows in x.

Distances between cases in x are computed as squared Euclidean distance after standardization by dividing by standard deviations. Standardization can be suppressed by including 'standard:F' as an argument. NOTE: This is a change in behavior of cluster() from version 3.1 to version 3.3.

cluster(dissim:d) uses the upper triangle of the square matrix d as dissimilarity or distance measure. Matrix d must have at least 2 rows and is treated algorithmically as if it were unsquared Euclidean distance.

cluster(similar:s) uses the upper triangle of the square matrix s as a similarity matrix. Matrix sqrt(2*(max(vector(s))-s)) is used as a distance matrix. Matrix s must have at least 2 rows.

Other Keywords

Keyword phrase	Default	Meaning
method:Name	"average"	The clustering method used. Legal values are "ward", "single", "complete", "average", "mcquitty", "median" and "centroid". Name must be a quoted string or CHARACTER variable.
nclust:m	9	The number (>= 2) of clusters to be described in the output. If m > 25, the class membership table requires more than 80 columns for printing, and if m > 22 the dendrogram requires more than 80. m > 50 is illegal when either the class membership table or the dendrogram is to be printed.
standard:F	T	suppresses the standardization of the data matrix to unit standard deviations before computing distances. Not legal with 'dissim' or 'similar'.
distance:Dname	"euclid"	Specifies the distance measure used to label the dendrogram. Legal values are "euclid" and "euclidsq". It has no effect on the clustering produced. Dname must be a CHARACTER variable or quoted string. Not legal with keywords 'dissim' or 'similar'.
keep:charVec	none	Specifies which, if any, results should be returned as the value of cluster(). charVec must be a quoted string or a CHARACTER vector or scalar. Legal values for elements of charVec are "distances" (the computed distances are returned), "classes" (the computed n by nclust-1 class membership matrix is returned), "crit" (the criterion values at each of the final nclust - 1 merges are saved), and "all" (all three are returned). If only one item is to be returned, it is returned as a matrix or vector. Otherwise, items are components in a structure with names 'distances', 'classes', and 'criterion'. The use of 'keep' suppresses printing the table of class membership and the dendrogram, unless print:T, tree:T, or

		classes:T are arguments. If 'keep' is not used, cluster() has a NULL value.
print:T		Forces printing output, even when 'keep' is used. Default is F when 'keep' is used; otherwise the default is T.
tree:T		Forces printing of dendrogram.
tree:F		Suppresses printing of dendrogram. Default is F when 'keep' is used; otherwise T. Must come later than 'keep' in argument list.
classes:T		Forces printing of table of class membership.
classes:F		Suppresses printing of table of class membership. Default is F when 'keep' is used; otherwise T. Must come later than 'keep' in argument list.
reorder:T	F	Directs that the rows of the printed table of class membership be reordered so that cases in the same clusters are adjacent. It does not affect the returned value if keep:"classes" appears. The reordering is the same as that implied in the dendrogram. A warning message is printed if you use reorder:T together with classes:F.

```
Example:
  Cmd> results <- cluster(x,nclust:15,keep:vector("classes","crit"),\
       method:"median",classes:T, reorder:T)
```
computes the last 15 stages of clustering, using the so called median method, returns the class membership table and the criterion in a structure, and prints the reordered class membership table.

2.41 cmplx

Keywords: time series, complex arithmetic
Usage: `complx(Re,Im)`, Re and Im REAL matrices with same size and shape. `complx(Re)`

`cmplx(re,im)` combines matrices re and im considered as the real and imaginary parts of a complex matrix. The j-th columns of re and im become the 2j-1-th and 2j-th columns of the result. Re and im must have the same size and shape and the output has the same number of rows and twice the columns. For example, if re and im are both 5 by 2, `cmplx(re,im)` is equivalent to `hconcat(re[,1],im[,1],re[,2],im[,2])`.

`cmplx(re)` is equivalent to `cmplx(re,0*re)`, that is, it produces a complex matrix with 0 imaginary part.

See topic 'complex' for discussion of complex matrices in MacAnova.

2.42 coefs

Keywords: glm, anova, regression
Usage: coefs([Term] [, errorTerm:ErrorTerm, se:T, coefs:F, byterm:F]), Term a CHARACTER scalar, a positive integer, or a factor or variate in the current GLM model, ErrorTerm a CHARACTER scalar or positive integer. Use byterm:F only when Term and coefs:F omitted, and se:T included

coefs(Term) returns the model effects or regression coefficients for term Term in the current GLM model. These are determined from information computed by the most recent GLM (generalized linear or linear model) command such as regress(), anova(), or poisson().

Term is usually a quoted string or CHARACTER variable such as "a.b" which exactly matches a term in the most recent model, that is, "a.b" is not the same as "b.a". An interaction term produces a matrix or array with the leftmost subscript corresponding to the leftmost factor in Term. If a model term contained {expr} where expr is a MacAnova expression, '{' and '}' are part of the term name and must be included.

For a term which consists of a single factor or variate, Term can be its unquoted name.

Alternatively, Term can be a integer between 1 and the number of terms, excluding the final error term. Thus, for example, unless the model contained "-1", coefs(1) gets the estimated intercept or grand mean.

coefs() (no Term specified) computes coefficients for all terms in the model as a structure with one component for each term.

coefs(Term,se:T) and coefs(se:T) also compute standard errors and are equivalent to secoefs(Term) and secoefs(), respectively. You can also use keywords 'error' and 'byterm' in this case. See secoefs().

coefs(Term,Varno) or coefs(,Varno) computes coefficients only for variable number Varno in the case of a multivariate dependent variable. If present, Varno must be the second argument and any keywords must follow it.

Function coefs() does not work after screen() or after a GLM command with 'coefs:F' as an argument.

Example: After anova("y= a + b + a.b")
coefs(a), coefs("a"), or coefs(2) will compute the main effect coefficients for factor a
coefs("a.b") or coefs(4) will produce a matrix of the a by b interaction coefficients.

```
   coefs() will produce all coefficients, including the constant.

See also secoefs().
```

2.43 colplot

Keywords: plotting
Usage: `colplot(x [, graphics keyword phrases]), x a REAL matrix`

```
colplot(x) makes an "interaction" plot of the data in the REAL matrix x.
The plotting positions are the row numbers and the values in x.  Points
within each column are joined by lines.  Any keywords useable in chplot
may follow x.  Colplot is implemented as a pre-defined macro.

If option 'dumbplot' has been set False (see setoptions()), the plot
will be a low resolution plot unless 'dumb:F' is an argument.

Example:
  Cmd> colplot(run(20)^(.2*run(5)'),xlab:"X",\
       title:"X^.2, X^.4, X^.6, X^.8, X")

See also rowplot.
```

2.44 comments

Keywords: syntax
Usage: `[command1; command2 ...]  # comment which will be ignored`

```
Anything following a '#' on a line is ignored unless it is part of a
string quoted with '"'.  You can use this feature to add comments to
spooled output or usage information to macros.  The '#' and everything
following up to the end of the line or a terminating '\' are skipped.

You can use function macrousage() to print any comment lines (lines
starting with "#") in a macro.  It is good practice to include comment
lines describing the usage.  They should not be confused with header
lines starting with ')' in a file which may also give usage information.
See macrousage(), macros, files.

Example:
  Cmd> xbar <- sum(x)/nrows(x) # compute mean as a row vector
is just the same as
  Cmd> xbar <- sum(x)/nrows(x)

See also spool().
```

2.45 complex

Keywords: time series, complex arithmetic
Usage: `cmplx(re,im), hprdhj(hx1,hx2), hprdh(hx1,hx2), cprdc(cx1,cx2), cprdcj(cx1,cx2), hpolar(hx), cpolar(cx), hrect(hx), crect(cx), hreal(hx), himag(hx), creal(cx), cimag(cx)`

MacAnova stores complex matrices in two forms, fully complex and packed Hermitian.

The fully complex form has alternating columns containing the real and imaginary parts of the columns of the complex matrix represented. If such a matrix has an odd number of columns, there is an implied additional column of zeros. Thus columns 1, 3, 5, ... are the real parts of complex series and columns 2, 4, 6, ... are the corresponding imaginary parts.

The packed Hermitian form is useful only for matrices whose columns represent periodic complex frequency functions with Hermitian symmetry sampled at frequencies 0,1/n,2/m,...,(n-1)/n cycles, where n is the number of rows. Hermitian symmetry for such a function g(f) means g(-f) = g(1-f) = conj(g(f)). This implies that g(0) and g(.5) are real and g((n-i)/n) = conj(g(i/n)), and hence only n real numbers are required to represent such series. The n numbers are stored in the following order, the packed Hermitian form:
z[0],Re(z[1]),...,Re(z[(n-1)/2]),{z[n/2]},Im(z[(n-1)/2]),...,Im(z[1]),
where z[i] = g(i/n). z[n/2] is omitted if n is odd.

You can create a fully complex matrix from its real and imaginary parts Re and Im using cmplx(Re,Im).

You can extract the real and imaginary parts and compute the complex conjugate of a fully complex matrix cx by creal(cx), cimag(cx), and cconj(cx).

You can do the same for a packed Hermitian matrix hx by hreal(hx), himag(hx), and hconj(hx).

You can switch between the two representations of a periodic Hermitan series by htoc(hx) and ctoh(cx).

You can transform both types of complex matrices to and from polar form (with the modulus being stored as the real part and the argument as the imaginary part of fully complex or packed hermitian matrices) by cpolar(cx), crect(cx), hpolar(hx), and hrect(hx).

You can multiply them column by column by cprdc(cx1,cx2), cprdcj(cx1, cx2), hprdh(hx1,hx2), and hprdhj(hx1,hx2).

Matrix multiplication is not possible with complex matrices in MacAnova.

2.46 compnames

Keywords: structures, character variables
Usage: `compnames(S), S a structure`

compnames(struc) returns a CHARACTER vector containing the names of the components of STRUCTURE struc.

```
compnames(structure(a:1,b:2,c:17)) has value vector("a","b","c")
```

See also structure(), strconcat(), changestr(), structures, nameof(), varnames()

2.47 console

Keywords: files, input
Usage: `y <- console()`

y <- console() uses a pre-defined macro to read a vector using variable CONSOLE as the filename. This results in a request to type the data in, terminating it with '!'. This has the advantage over y <- vector(1.3, 4.5, 6.7, 2.5, ...), say, in that the commas do not need to be typed. Variable CONSOLE can be used with any operation that reads or writes a file. Its value is ignored.

Command vecread() is used by console so you can use '?' to specify missing data.

On a Macintosh, you enter data into a dialog box. Return or the OK button ends a line. The Done button terminates the data.

y <- console(echo:F) suppresses any echoing of lines read. Such echoing is the default behavior on a Macintosh, or when using console in a batch file on any system.

See vecread().

2.48 contrast

Keywords: glm, anova, comparisons
Usage: `contrast(Term,Coefs [,Byvar] [,errorTerm:ErrorTerm]),`
Term a factor in the most recent GLM model or a character SCALAR or positive integer specifying a term and ErrorTerm a CHARACTER scalar or positive integer, Coefs REAL, Byvar a factor in the most recent GLM or a CHARACTER scalar specifying such a factor

contrast(Term,Coefs) computes the estimated value, sum of squares, and standard error for the contrast in the levels of the term given in the quoted string or CHARACTER variable Term with contrast coefficients given in Coefs. Term must be made up exclusively of factor variables in the model used by the most recent GLM (generalized linear or linear model) command such as anova() or poisson(). The result is a structure with components 'estimate', 'ss', and 'se'.

If any of the variables in the model were of the form {expr}, where expr is a MacAnova expression, you must specify the variable in the same way. See models.

Term can also be a factor in the model, that is, say, after anova("y=a+b"), contrast(a,c) is equivalent to contrast("a",c).

Term may also be a positive integer, in which case contrast(Term,Coefs) is equivalent to contrast(TERMNAMES[Term],Coefs).

If Term contains more than one factor (for example "a.b"), Coefs must be an array with dimensions matching the number of levels in the factors of Term. The contrast coefficients must sum to zero, but MacAnova does not check to see if the contrast lies in any particular subspace.

contrast(Term,Coefs,Byvar) computes the contrast, sum of squares, and standard error separately for each level of the factor variable (the "byvariable") given in the CHARACTER or quoted string variable Byvar.

Alternatively, Byvar can be a factor in the model but cannot be specified by a positive integer as can Term.

For both forms, an additional optional argument of the form errorterm:ErrTerm or errorterm:ErrTermNo, where ErrTerm is a CHARACTER variable or quoted string specifying a term in the model and ErrTermNo is a positive integer, specifies that the MS from the indicated term is to be used in computing standard errors.

Function contrast() does not work after screen() or after any GLM command with 'coefs:F' as an argument.

For unbalanced models, contrasts are computed as follows.

No byvariable:

If Term corresponds to a term in the current anova model, the sum of squares is that for removal of the contrast degree of freedom from the complete model. It is actually computed as a linear combination of the non-aliased coefficients associated with the term.

If Term is not present in the model, the sum of squares is the incremental sum of squares obtained when adding the contrast df to the complete model, that is, the contrast is adjusted for all terms in the model.

Byvariable specified:

```
  The contrast is unadjusted for any other terms in the model, that is,
  it is computed as if the contrast degree of freedom at that level of
  the byvariable is the only degree of freedom in the model).

In all three cases, the MSE used in computing the standard error is the
error mean square (or other mean square as specified by 'errorterm')
from the most recent GLM command.

After logistic(), probit(), poisson(), and other GLMs fit iteratively by
glmfit(), contrast() computes the estimated value and the deviance
associated with the contrast based on full model weights.  Key word
'byvar' may not be used, but deviances are otherwise computed as after
anova().  Standard errors are computed using a scale parameter of 1, or
the value specified by keyword 'scale' on the GLM command..

After robust(), the contrast value is computed based on coefficients
from the full model.  The "deviance" computed is the square of the ratio
of the estimated contrast to its estimated standard error multiplied by
the pseudo MSE used in computing the standard error.  The latter is
printed in an advisory message.  Keyword 'byvar' cannot be used.

Contrast() does not work following fastanova(), ipf(), or screen() or
after a GLM command with coefs:F.

Example:
  Cmd> anova("y=a+b+a.b")
  Cmd> contrast("b",vector(-1,1,0)) # assumes b has 3 levels
  Cmd> contrast("b",vector(-1,1,0),"a") # a is byvariable
  Cmd> contrast("a.b",matrix(vector(-1,1,0,0,1,-1),2))# assumes 2 by 3
```

2.49 convolve

Keywords: time series

Usage: convolve(wts, x [, reverse:T, decimate:n]), wts a REAL vector, x a REAL vector or matrix, n a positive integer

```
convolve(a,x) performs a circular convolution of the values in vector a
with each of the columns of vector or matrix x.  If we index rows
starting with 0, so that a contains elements a[0], a[1], ..., a[p-1],
and a column of x contains x[0], x[1], ..., x[n-1], the corresponding
column of the result is computed as follows:

  d[k] = sum(a[j]*x[k-j],j=0,min(k,p-1)) + sum(a[j]*x[k-j+n],j=k+1,p-1),

where the second sum is omitted when k >= p - 1.

convolve(a,x,reverse:T) computes sums of circularly lagged products of
the elements of a and each column of x.  Thus, with the same indexing,

 d[k] = sum(a[j]*x[j-k+n],j=0,min(k-1,p-1)) + sum(a[j]*x[j-k],j=k,p-1)
```

where the first sum is omitted when k = 0 and the second sum is omitted when k >= p.

convolve(a,x,decimate:m) and convolve(a,x,reverse:T,decimate:m) d[k] only for k = 1, 1+decimate, 1+2*decimate , ..., where decimate is a positive integer. The number of rows in the results is floor((nrows(x)-1)/decimate) + 1. This option is useful if a is the impulse reponse function of a smoothing filter.

See also autoreg(), movavg().

2.50 copyright

Keywords: general
Usage: Copyright (C) 1994 - 1998 by Gary W. Oehlert and Christopher Bingham. Type help(copyright) for fuller information and acknowledgments.

MacAnova is conceived and programmed by Gary W. Oehlert and Christopher Bingham, Department of Applied Statistics, University of Minnesota, and is Copyright (C) 1994 - 1998 by them. Their e-mail addresses are kb@stat.umn.edu and gary@stat.umn.edu.

MacAnova is distributed under the terms of the GNU Public License, Version 2 (see file COPYING distributed with MacAnova).

Briefly, this means that MacAnova may be freely copied and distributed and the source is available. The source will be available by anonymous ftp or other equivalent retrieval from stat.umn.edu. Any changes others make to the source must be clearly marked.

There is no warranty of any kind for MacAnova, either expressed or implied. MacAnova is distributed "as is". See file COPYING for a more complete statement.

The MacAnova WWW home page is
http://www.stat.umn.edu/~gary/macanova/macanova.home.html
Executable versions of the Macintosh, DOS and Windows versions are available there, along with source and Portable Document Format (PDF) versions of the User's Guide and other documentation. An up-to-date mirror of these files is maintained by statlib at
http://lib.stat.cmu.edu/

Reports of bugs should be emailed to kb@stat.umn.edu.

The Macintosh version uses TransSkel 3.12, a transportable Macintosh application skeleton placed in the public domain by Paul Dubois (dubois@primate.wisc.edu).

The extended memory MSDOS version (DJGPP) is compiled using a version of

Gnu gcc developed and copyrighted by D. J. Delorie (DJGPP) and distributed under the terms of the GNU Public License. Starting with MacAnova 4.04, version 2 of this compiler has been used. Source and executable for DJGPP can be retrieved via anonymous ftp from oak.oakland.edu (in pub/simtelnet/gnu/djgpp); modifications to the DJGPP library for its use in MacAnova here can be retrieved via anonymous ftp from umnstat.stat.umn.edu.

The Windows/Motif versions make use of the wxWin cross-platform windowing interface developed Dr. Julian Smart, Artifical Intelligence Applications Institute, The University of Edinburgh. wxWin is Copyright (c) 1995 Artificial Intelligence Applications Institute. The WxWin home page is
http://web.ukonline.co.uk/julian.smart/wxwin/ .

Currently both the Windows and Motif versions are based on version 1.68 of wxWin, a copy of which is available through the MacAnova home page.

Plotting is done using a modification of GNUplot, Copyright (C) 1986, 1987 Thomas Williams, Colin Kelley.

The Unix version and the extended memory DOS version (DJGPP) allow command line editing and history maintainance using the GNU Readline Library, Copyright (C) 1988, 1991 Free Software Foundation, Inc., distributed under the terms of the GNU public license. Source for this is available in many places. Version 2.0 is available by ftp from ftp.cis.ohio-state.edu as file /gnu/mirror/readline-2.0.tar.gz. The version used in the DOS DJGPP version was included with the source for gdb4.12 found on ftp://oak.oakland.edu/ which has been reorganized since we retrieved it.

Included in MacAnova's distribution are modified translations from Fortran to C of the following programs written by others.

Program screen and related subroutines for computing regressions by leaps and bounds by G.M.Furnival and R.W.Wilson supplied by Sanford Weisberg.

Subroutines rebak, reduc, rsg, tql2, tqlrat, tred1, tred2, svd, tridib, and tinvit from the Eispack library.

Subroutines dchdc, dgeco, dgedi, dgefa, dgesl, and dqrdc from the Linpack library.

Subroutines for computing mixed radix fast Fourier transforms written by Gordon Sande at the University of Chicago circa 1968.

Program hc and related subroutines for computing hierarchical cluster analysis by F. Murtagh, retrieved from statlib.

Subroutines for making stem and leaf displays from the book ABCs of EDA by David Hoaglin and Paul Velleman, Duxbury 1981.

Subroutines to compute the roots of real polynomials from Algorithm 493 published in TOMS retrieved from netlib.

Code to compute the cumulative normal adapted from W. J. Kennedy and J. E. Gentle, Statistical Computing, Marcel Dekker, 1980, pp 90-92, which is based on W. J. Cody, Rational Chebyshev approximations for the error function, Math. Comp 23 (1969) 631-637.

Code to compute the inverse of a normal distribution from Algorithm AS 111 by J.D. Beasley and S. G. Springer, Appl. Statist. 26 (1977), 118-121 retrieved from statlib.

Code to compute the inverse Student's t-distribution frin CACM Algorithm 396, by G. W. Hill retrieved from netlib.

Code to compute the (central) Beta distribution from a subroutine of W. Fullerton, Los Alamos, based on Bosten and Battiste, Remark on Algorithm 179, CACM 17 (1974) p. 153

Code to compute the inverse Beta distribution from Algorithm AS 109 by G. W. Cran, K. J. Martin and G. E. Thomas, Appl. Statist. 26 (1977), 111-114 retrieved from statlib.

Code to compute the non-central Beta distribution from Algorithm AS 226 by R. V. Lenth, Appl. Statist. 36 (1987) 241-244, incorporating changes by H. Frick, Appl. Statist. 39 (1990) 311-12, retrieved from statlib

Code to compute the gamma and chi-squared cumulative distributions from Algorithm AS 91 by D. J. Best and D. E. Roberts, Appl. Statist. 24 (1975), 385-388, incorporating revisions by B. L. Shea, Appl. Statist. 40 (1991), 233-235), retrieved from statlib.

Code to compute the non-central chi-squared cumulative distribution from Algorithm AS 275 by Cherng G. Ding, Appl. Statist. 24 (1992), 478-482, retrieved from statlib.

Code to compute the non-central Student's t cumulative distribution from Algorithm AS 243 by Russell V. Lenth, Appl. Statist. 38 (1989), 185-189, retrieved from statlib.

Code to compute the cumulative distribution function and its inverse for the Studentized range from Algorithm AS 190 by R. E. Lund and J. R. Lund, Appl. Statist. 32 (1983) 204-210, incorporating corrections by Lund and Lund, Appl. Statist. 34 (1985) 104 and I. D. Hill, Appl. Statist. 36 (1987) 119, retrieved from statlib.

Code for a combined uniform pseudo-random number generator for 32 bit machines in P. L'Ecuyer 1988 Comm. ACM, retrieved from netlib.

Code implementing the Singleton quicksort algorithm (Comm. ACM Algorithm 347) adapted from ssort.f in cmlib.

Code computing the cumulative distribution for Dunnett's t was adapted from Algorithm AS 251 by C. W. Dunnett , Appl. Statist. 38 (1989) 564-579 incorporating a correction by C. W. Dunnett, Appl. Statist. 42 (1993) p. 709, and subroutine mvstud, also by Dunnett, that is part of the AS 251 distribution from statlib.

Code generating a pseudo-random Poisson variable adapted from a Fortran program in C. D. Kemp and W. A. Kemp, Poisson random variate generation, Appl. Statist. 40 (1991) 143-158.

Code generating a pseudo-random binomial variable adapted from Algorith 678, Transactions on Math. Software 15, 394-397 by Voratas Kachitvichyanukul and Bruce Schmeiser.

Code implementing varimax rotation from subroutine varmx supplied by Douglas Hawkins (doug@stat.umn.edu).

Code implementing k-means clustering from subroutine trwcla supplied by Douglas Hawkins (doug@stat.umn.edu).

Code used to compute inverses to cumulative distributions from subroutine fsolve supplied by Douglas Hawkins (doug@stat.umn.edu). It is used by invchi() to compute the inverse of non-central chi-squared and by invdunnett() to compute probability points of Dunnett's t.

2.51 cor

Keywords: descriptive statistics
Usage: cor(x1 [,x2,...]), x1, x2, ... REAL vectors or matrices all with the same number of rows

cor(x) computes a correlation matrix for data in REAL matrix x. Any row of x containing missing data is entirely omitted. It is an error to have missing data in all rows.

cor(a,b,c,...) is equivalent to cor(hconcat(a,b,c,...)). All arguments must be vectors or matrices with the same number of rows.

If any column in the input is constant (all values the same), the entire corresponding row and column of the output is set MISSING, including the diagonal, and a warning message is printed. In particular this occurs when the number of rows in the input is 1.

All non-MISSING elements of the diagonal are always exactly 1.

2.52 cpolar

Keywords: time series, complex arithmetic
Usage: `cpolar(hx [,unwind:F or crit:val]), hx a REAL matrix representing complex data, val a REAL scalar, 0.5 < val <= 1`

cpolar(cx) computes the polar form of the fully complex matrix cx, storing it in pseudo fully complex form, with the modulus as the real part and the phase (argument) as imaginary part. Thus creal(cpolar(cx)) returns a REAL matrix whose columns are the moduli of the complex series represented by pairs of columns of cx and cimag(cpolar(cx)) returns the phases. By default the latter are "unwound" so as to minimize discontinuities arising from wrap-around.

cpolar(cx,crit:val) changes the criterion controlling "unwinding". See unwind() for details.

cpolar(cx,unwind:F) suppresses the unwinding.

See also hpolar(), crect(), hrect().

See topic 'complex' for discussion of complex matrices in MacAnova.

2.53 cprdc

Keywords: time series, complex arithmetic
Usage: `cprdc(cx1 [, cx2]), cx1 and cx2 REAL matrices representing complex data`

cprdc(cx1, cx2) computes the element wise complex multiplication of fully complex (pairs of columns consitute real and imaginary parts) matrices cx1 and cx2. If cx1 or cx2 has an odd number of columns, it is augmented with a column of zeros before multiplication.

cprdc(cx) is equivalent to cprdc(cx,cx).

If cx1 or cx2 represents a single complex series (has 1 or 2 columns), that series is multiplied by all the series in the other arguments. Thus for example, if cx1 and cx2 have 2 and 6 columns, respectively, and nrows(cx1) = nrows(cx2), cprdc(cx1,cx2) is equivalent to cprdc(cx1[,vector(1,2,1,2,1,2)], cx2).

See also cprdcj(), hprdh(), hprdhj().

See topic 'complex' for discussion of complex matrices in MacAnova.

2.54 cprdcj

Keywords: time series, complex arithmetic
Usage: `cprdcj(cx1 [, cx2]), cx1 and cx2 REAL matrices representing complex data`

cprdcj(cx1, cx2) computes the element wise complex multiplication of fully complex (pairs of columns consitute real and imaginary parts) matrices cx1 and cconj(cx2). If cx1 or cx2 has an odd number of columns, it is augmented with a column of zeros before multiplication.

cprdcj(cx) is equivalent to cprdcj(cx,cx) and returns as output a matrix with the squared moduli of the elements of cx, considered as complex numbers, in the odd columns (real part) of the result, with zero's in the even columns (imaginary part).

If cx1 or cx2 represents a single complex series (has 1 or 2 columns), that series is multiplied by all the series in the other arguments. Thus for example, if cx1 is m by 2 and cx2 is m by 6, cprdcj(cx1,cx2) is equivalent to cprdcj(cx1[,vector(1,2,1,2,1,2)],cx2).

See also cprdc(), hprdh(), hprdhj(), cconj().

See topic 'complex' for discussion of complex matrices in MacAnova.

2.55 creal

Keywords: complex arithmetic, time series
Usage: `creal(cx), cx a REAL matrix representing complex data`

creal(cx) computes the real part of the fully complex matrix cx. Thus creal(matrix(run(10),5)) is vector(1,2,3,4,5).

See also hconj(), cconj(), hreal(), himag(), cimag().

See topic 'complex' for discussion of complex matrices in MacAnova.

2.56 crect

Keywords: time series, complex arithmetic
Usage: `crect(cx), cx a REAL matrix representing complex data`

crect(cx) is the inverse operation to cpolar(). Matrix cx is assumed to represent the polar form of a fully complex series, with moduli in the real part and phases or arguments in the imaginary part. The result contains the real and imaginary parts of that series in fully complex form.

See also cpolar(), hpolar(), hrect().

See topic 'complex' for discussion of complex matrices in MacAnova.

2.57 ctoh

Keywords: time series, complex arithmetic
Usage: ctoh(cx), cx a REAL matrix representing complex data

ctoh(cx) returns the packed Hermitian symmetrized form of the REAL matrix cx, considering its columns in pairs as representing unrestricted Complex series. If cx is m by 2*n-1, or m by 2*n, ctoh(cx) is m by n, with column i containing the Hermitian symmetrized form of columns 2*i-1 and 2*i. If ncols(cx) is odd, the final complex series is assumed to have imaginary part zero.

If cx actually has Hermitian symmetry, then ctoh(cx) represents the same matrix in packed form. If cx does not have such symmetry, ctoh(cx) represents the symmetrized matrix obtained by averaging elements that should be equal and discarding the imaginary parts of elements that should be real.

See topic 'complex' for discussion of complex matrices in MacAnova.

See also htoc(), cconj(), hconj(), hreal(), himag(), creal(), cimag().

2.58 cumbeta

Keywords: probabilities
Usage: cumbeta (x,alpha,beta[,lam]), x, alpha, beta, and lam REAL, elements of alpha, beta > 0, lam >= 0

cumbeta(Val,a,b) computes the probabilities that a beta random variable with parameters a and b would be less than the elements of the vector, matrix, or array Val.

cumbeta(Val,a,b,lam) computes similar probabilities for non-central beta with noncentrality parameter lam.

Any of Val, a, b, or lam that are not scalars (single numbers) must be vectors, matrices, or arrays with the same size and shape which will also be the size and shape of the result.

Parameters a and b must be positive real numbers. Upper tail areas of beta can be computed as 1 - cumbeta().

Example:

```
Cmd> 1 - cumbeta(.83,1.5,2.5)  # a = 1.5, b = 2.5
```

See also cumF(), invF(), invbeta().

2.59 cumbin

Keywords: probabilities
Usage: cumbin(x,N,P), x, N and P REAL, elements of N integers > 0, elements of P between 0 and 1

cumbin(Val,N,P) computes the probabilities that a binomial random variable with N trials at probability P would be less than or equal to the elements of the vector, matrix or array Val. If Val is not integral, the result is the same as for argument floor(Val).

Any of Val, N, and P that are not scalars (single numbers) must be vectors, matrices, or arrays with the same size and shape which will also be the size and shape of the result. The elements of N must be positive integers less than 1000 and the elements of P must be between zero and one.

Probabilities that a binomial random variable is greater than or equal to Val can be computed as 1 - cumbin(Val-1,N,P).

Example:
```
  Cmd> 1 - cumbin(7,13,.25)  # n = 13, p = .25
```

2.60 cumchi

Keywords: probabilities
Usage: cumchi(x,df), x and df REAL, elements of df > 0
cumchi(x,df,lam), same x, df, lam REAL with 0 <= lam[i] < 1419.56542578676

cumchi(Val,df) computes P(x <= Val) where x is a chi-square random variable with df degrees of freedom. If Val is a vector, matrix or array, the probablities are computed for each element. If Val and df are both not scalars, they must be the same size and shape which will also be the size and shape of the result.

The elements of df must be positive (fractional degrees of freedom are allowed). Upper tail areas of chi squared can be computed as 1 - cumchi(Val, df).

Example:
```
  Cmd> 1 - cumchi(sum((obs-e)^2/e), nrows(obs) - 1) # P value
```

cumchi(Val,df,lam) computes P(x <= Val) where x is a non-central chi-square random variable with df degrees of freedom and non-centrality parameter lam. All non-scalar arguments must be the same size and shape

```
which will also be the size and shape of the result.  The elements of
Val must be non-negative and less than 1419.56542578676.

Example:
  Cmd> 1 - cumchi(invchi(1-.05,df),df,n*sum((p1-p0)^2/p0)) # power

See also cumgamma(), invchi(), invgamma().
```

2.61 cumdunnett

Keywords: probabilities, comparisons

Usage: cumdunnett(x, ngroup, errorDf [,groupSizes][,onesided:T][,epsilon:ep
x REAL, elements of ngroup integers >= 2, elements of
errorDf >= 1, elements of groupsizes >= 0, eps > 0,
default = .00001.

```
cumdunnett(x, K, Df) computes the probability that Tmax <= x, Tmax =
max(abs(t21), abs(t31), ..., abs(tK1)), where t21, t31, ..., tK1 are K-1
t-statistics of the form tI1 = (xbarI-xbar1)/ stderr(xbarI-xbar1), I =
2,...,K.  xbar1, xbar2, ..., xbarK are the means of independent normal
random samples of the same size with identical population means and
variances, and the standard errors are computed using an independent
estimate of error variance with Df degrees of freedom.  The value is 0
for any x <= 0.  For x >= 0, when K = 2 the value is the same as
2*cumstu(x,Df) - 1.  See cumstu().

cumdunnett(x, K, Df, onesided:T) computes the probability that Tmax<=x,
where Tmax is now the maximum of t21, t31, ..., tK1, not of their
absolute values.  When K = 2 the value is the same as cumstu(x,Df).

See below for computing probabilities when the sample sizes differ.

x, K and Df must be REAL.  The elements of K must be integers >= 2, and
the elements of Df must be >= 1, not necessarily integers.

Any of the arguments x, K or Df that are not scalars must all be
vectors, matrices or arrays of the same size and shape; the value has
the same size and shape.

cumdunnett(x, K, Df [, onesided:T], epsilon:eps), where eps is a small
positive number (default .00001) which controls the accuracy to which
the probability is computed; the computed probability should be no
farther than eps from the true probability.

cumdunnett() is primarily used to compute P values for a multiple
comparisons procedure due to C. W. Dunnett wherein a control group
(group 1) is compared to K-1 other treatment groups using K-1 t-tests.
For a completely randomized design with k treatment groups of size n,
the P value is computed as 1-cumdunnett(maxt, k, k*n - k [,onesided:T]).
See invdunnett() for computing critical values of the Dunnett test.
```

Caution: cumdunnett() is very computation intensive and may be unacceptably slow on an older computer. On one Macintosh 68000 computer with no math coprocessor, a single value took about 7 minutes to compute.

Example:

```
  Cmd> cumdunnett(4.1, 5, 5*8 - 5)
```

computes P(Tmax > 4.1) for a completely randomized design with 5 groups, all with sample size 8.

cumdunnett(x, K, Df, groupSizes [,onsided:T, epsilon:eps]) computes probabilities for Tmax, with REAL argument groupSizes specifying the sample sizes. In the simplest usage, groupSizes is a vector (ndims(groupSizes) = 1), with elements >= 0. If groupSizes is a matrix or array (ndims(groupSizes) > 1), it is treated as if it were a vector, matrix or array, with one less dimension, each of whose elements is a vector with length = last dimension of groupSizes. The first ndims(groupSizes) - 1 dimensions of groupSizes must match the dimensions of any of x, K, or DF which is not a scalar. In particular, a m by 1 matrix, which is treated as a vector of length m by most MacAnova functions, is interpreted by cumdunnett() as a set of m vectors of length 1.

In computing an element of the result based on a vector of group sizes (either all of groupSizes when it is a vector, or a row or "slice" of groupSizes when ndims(groupSizes) > 1), cumdunnett() uses up to k of the non-zero leading values in the vector, where k is the corresponding element of K. If there are fewer than k non-zero values, the last one is replicated as many times as needed. It is an error to have a zero value followed by a nonzero value or to have all values zero.

If there is only 1 non-zero value in a row or "slice" of groupSize, the replication of this element means the group sizes are assumed to be equal. In particular, this is the interpretation when groupSizes is a scalar or a m by 1 matrix.

Examples:

```
  Cmd> cumdunnett(4.1, 4, 12 - 4, vector(6,2,2,2))
```

computes P(Tmax <- 4.1) for a completely randomized design with 4 groups and sample sizes 6, 2, 2 and 2.

```
  Cmd> cumdunnett(4.1, vector(3,4), vector(12 - 3, 12 - 4),\
        matrix(vector(6,3,3,0, 6,2,2,2),4)')
```

computes P(Tmax < 4.1) for two completely randomized designs, one with 3 groups and sample sizes 6, 3, and 3, the other with 4 groups with sample sizes 6, 2, 2, and 2. Because trailing values in in the rows of groupSizes are replicated, matrix(vector(6,3, 6,2),2)' would be an equivalent way to specify the group sizes.

```
  Cmd> cumdunnett(4.1, 4, 12 - 4, 3)
```

computes the same result as cumdunnett(4.1, 4, 12 - 4), because groupSizes is a scalar.

Only the ratios of non-zero elements of groupSizes are relevant. Thus

for 5 groups (K=5), the following groupSizes are equivalent:
vector(1,2,3,3,3), vector(1,2,3), vector(1,2,3,0,0), vector(2,4,6).
vector(1,2,3,0,3) and vector(1,2,3,0,0,1) would be errors because a non-zero value follows a zero.

Caution: cumdunnett() is somewhat computation intensive. On a slow computer you may have to wait several seconds for a result. Using a somewhat larger value for epsilon, for example, epsilon:.0001, may speed up the calculation at the cost of loss of accuracy.

See also invdunnett(), cumstudrng().

2.62 cumF

Keywords: probabilities
Usage: cumF(x,df1,df2 [,lam]), x, df1, df2 and lam REAL, elements of df1 and > df2 0 and lam >= 0

cumF(Val,df1,df2) computes the probabilities that an F random variable with df1 and df2 degrees of freedom would be less than the elements of the vector, matrix, or array Val.

cumF(Val,df1,df2,lam) computes similar probabilities for non-central F with noncentrality parameter lam.

Any of Val, df1, df2, or lam that are not scalars (single numbers) must be vectors, matrices, or arrays with the same size and shape which will also be the size and shape of the result.

The degrees of freedom must be positive REAL numbers (not necessarily integers). Upper tail areas of F can be computed as 1 - cumF().

Example:
```
  Cmd> 1 - cumF((SS[2]/DF[2])/(SS[5]/DF[5]), DF[2], DF[5])
```

See also invF(), cumbeta(), invbeta().

2.63 cumgamma

Keywords: probabilities
Usage: cumgamma(x,alpha), x and alpha REAL, elements of alpha > 0

cumgamma(Val,alpha) computes the probabilities that a gamma random variable with shape parameter alpha would be less than the elements of the vector, matrix, or array Val.

If Val and alpha are both not scalars, they must be vectors, matrices,

```
   x <- fromclip(5)
will create a 10 by 5 REAL matrix consisting of the data copied.

fromclip is implemented as a pre-defined macro.
```

2.99 frowplot

Keywords: plotting, files
Usage: frowplot(fileName, x [, graphics keyword phrases]), fileName a CHARACTER scalar, x a REAL matrix

```
frowplot(FileName,x [,...]) is equivalent to rowplot(x,file:FileName
[,...]).  The use of frowplot is discouraged; use rowplot instead.

See macro rowplot for details on arguments and keywords.
```

2.100 fwrite

Keywords: output, files
Usage: fwrite(fileName, a, b, ...[,format:Fmt or nsig:m, header:F, labels:F, missing:missStr, width:w, height:h]), Fmt, missStr, fileName CHARACTER scalars, m > 0, w >= 30, h >= 12 integers

```
fwrite(name,a,b,...) is equivalent to write(a, b, ..., file:name).  The
use of fwrite() is discouraged.  Use write() instead.

See write() and print() for details on keyword use.

See also matprint(), macroprint().
```

2.101 getdata

Keywords: files, variables, input
Usage: y <- getdata(setName), where setName is the unquoted name of a data set on file specified by variable DATAFILE

```
getdata is a pre-defined macro to retrieve named data sets from a file
whose name is in CHARACTER variable DATAFILE.  Specifically,
    y <- getdata(DataName)
is equivalent to
    y <- matread(DATAFILE,"DataName")

The data set name should not be enclosed in quotes or be a CHARACTER
```

2.96 fplot

Keywords: plotting, files
Usage: fplot(fileName, x,y [,impulse:T, lines:T] [,graphics keyword phrases]), where fileName is a CHARACTER scalar, x is a REAL vector or scalar, y is a real vector or matrix

fplot(FileName,x,y [,...]) is equivalent to plot(x,y,file:FileName, [,...]). The use of fplot() is discouraged; use plot() instead.

See plot() for details on arguments and keywords.

2.97 fprint

Keywords: output, files
Usage: fprint(fileName, a, b, ...[,format:Fmt or nsig:m, header:F, labels:F, missing:missStr, height:h, width:w]), Fmt, missStr, fileName CHARACTER scalars, m > 0, h >= 12 integer, w >= 30 integer

fprint(name,a,b,...) is equivalent to print(a, b, ...,file:name). The use of fprint() is discouraged. Use print() instead.

See print() and write() for details on keyword use.

See also write(), matprint(), matwrite(), macrowrite().

2.98 fromclip

Keywords: character variables, input
Usage: fromclip([ncol]), ncol > 0 an integer

y <- fromclip() does a vecread() from CLIPBOARD, creating a REAL vector y.

y <- fromclip(ncol) is equivalent to matrix(fromclip(),ncol)', where ncol is an integer. This creates a REAL matrix with ncol columns and makes most sense if the contents of CLIPBOARD are arranged in ncol columns as they would be after CLIPBOARD <- x, where x is a matrix with ncol columns.

In the Macintosh, Windows and Motif versions, fromclip allows easy importing of data from other applications such as a spreadsheet. Thus, if you have Copied to the Clipboard a 10 by 5 rectangle of spreadsheet cells containing numbers, or 10 lines of a 5 column numerical table in a word processor or editor,

```
the i-th component of floor(x) is floor(xi).

Examples: floor(vector(3.1416, -3.1416, 12)) is vector(3, -4, 12)

See also ceiling(), round(), structures.
```

2.95 for

Keywords: syntax, control
Usage: `for(i,vec))command1;command2; ..., vec a REAL vector for(i,NULL)... for(i,start,end [, incr])command1;command2;..., start, end and incr REAL scalars`

```
for(Index,Range){statement1;statement2;... } where Range is a REAL
vector of length N repeats the statements in {...} N times with Index
taking the values Range[1], Range[2], ..., Range[N] successively.
Unless the last statement in {...} is null (';;') its value may be
printed on each loop.  The most common form for Range is 'run(n)' which
results in the statements in {...} being repeated n times, with variable
Index taking values 1, 2, ..., n.

for(Index,NULL){...} skips the compound statement {...}.  An example
might be
  Cmd> for(i, run(length(x))[x>10]){print(paste("x[",i,"] > 10"))}
when max(x) < 10.

for(Index,i1,i2){...} is equivalent to for(Index,run(i1,i2)){...}.

for(Index,i1,i2,incr){...} is equivalent to for(Index,
run(i1,i2,incr)){...}.

A 'for' statement does not have a value.  Hence such constructs as
  yyy <- for(i,run(3)){i+2} or 4 + for(i,run(3)){i+2}
are illegal.

Example:
  Cmd> @n <- length(a);@s <- 0;for(i,run(@n)){@s <- @s+(a[i]-1)^2;;}; @s
is essentially equivalent to sum((vector(a)-1)^2).  'for(i,run(@n))'
could be replaced by 'for(i,1,@n)' or 'for(i,1,@n,1)'.

The following might be better:
  Cmd> @s <- 0; for(@ai,vector(a)){@s <- @s + (@ai - 1)^2;;}; @s

The opening '{' after 'for(...)' must be on the same line as 'for'.

See also while, break, breakall.
```

See batch() for information on the format of batch files.

Files written by asciisave() are ASCII files but are not meant to be read or edited by humans and their format is arcane and subject to change. The asciisave() format is the same across computer types, so that, for example, a DOS machine can read a Macintosh asciisave() file and vice versa. asciisave() files can also be emailed as is, without encoding.

Files written by save() are binary files with formats that may differ among computer types.

The default help file, usually "macanova.hlp" in the same directory as MacAnova, is a text file in a special format. The format is described near the start of the file. If you develop a file of macros, you could use this information to write a special help file. You can change the file used by help() by means of keywords 'file', 'orig' and 'alt'. See help().

2.93 flineplot

Keywords: plotting, files
Usage: flineplot(fileName, x,y [,linetype:m][,other graphics keyword phrases]), where fileName is a CHARACTER scalar, x is a REAL vector or scalar, y is a real vector or matrix, and m >= 0 is an integer

flineplot(FileName,x,y [,...]) is equivalent to lineplot(x,y, file:FileName [,...]). The use of flineplot() is discouraged; use lineplot() instead.

See lineplot() for details on arguments and keywords.

2.94 floor

Keywords: transformations
Usage: floor(x), x REAL

floor(x) rounds the elements of the REAL variable x to the next integer in the negative direction, producing a vector, matrix, or array with the same shape as x.

When x > 4503599627370495 or x < -4503599627370495, floor(x) is set to MISSING because of the impossibility of exact representation of integers beyond these limits. These limits may be different on some computers.

If x is a structure, so is floor(x). If xi is the i-th component of x,

You cannot change the default directory under Unix.

In the limited memory version (BCPP), you change the default directory by, for example,

```
  Cmd> shell("cd e:\\classes\\stat1001",interact:T)
```

or

```
  Cmd> !cd e:\classes\stat1001
```

This does not currently work in the extended memory version (DJPP) or Windows version. See shell(). In both forms, '\' is required; '/' won't do.

On a Macintosh, the default folder initially is the MacAnova directory, but is changed whenever you use the scrolling dialog box to select a file in a different folder. Thus the first time you read a file from a given folder, you should use "" as file name. If you want, you can then read other files in the same folder by specifying their names. This feature means you never need specify a path on the Macintosh.

If your file name includes a path, it can be relative or absolute. In the next paragraphs, for a Macintosh, 'directory' should be taken to mean 'folder'.

Under DOS/Windows or Unix, a file name specifying a relative path might be "../data.dir/mydata" or "tseries.dir/tseries.dat" (under DOS/Windows you could replace '/' by '\\'). On a Macintosh, a relative path starts with ":" Examples would be "::data.dir:mydata" and ":tseries.dir:tseries.dat". On all three systems, these examples specify either a file 'mydata' in a directory 'data.dir' in the "parent" directory of the default directory, or a file 'tseries.dat' in a sub-directory 'tseries.dir' in the default directory. On a Macintosh, "directory" should be interpreted as "folder."

On Unix "/usr/local/data/survey.dat" is a typical absolute path name. It must start with '/'. On DOS/Windows an absolute path name starts with '/' or 'X:/' where X is the designator of a disk drive. For example, if the current DOS default drive is drive 'C', both "C:/data.dir/survey.dat" and "/data.dir/survey.dat" specify the same file. On a Macintosh, an absolute path name starts with a disk name and contains at least one ":", for example, "MyDisk:data.dir:survey.dat". The variable HOME described previously should normally contain the absolute path name of a directory.

File formats

On all systems, data, macro and batch files must be plain text (ASCII) files. If you create them using a word processor, be sure to specify text format when you save them.

See topics data files for information on the format of files to be read by vecread() and matread().

See topics macro files for information on the format of files to be read by macroread().

On any command that involves reading a file, if MacAnova cannot find the file you specify in the default directory or folder (see below), it searches for it in the directories or folders whose names are in CHARACTER vector DATAPATHS. Each element of DATAPATHS must be the complete name (path) of a directory. No special search is done if the name you specify is not a pure file name, that is, it includes any of the special characters use to build "paths" (':' on a Macintosh, ':', '/' or '\' on DOS/Windows, and '/' on Unix). You can add directories or folders to DATAPATHS using macro adddatapath.

For example, on DOS/Windows, if DATAPATHS is vector("C:/MACANOVA", "D:/SURVEY"), matread("mydata.dat") attempts to read MYDATA.DAT in the default directory and then, if not successful, it tries to read C:\MACANOVA\MYDATA.DAT or D:\SURVEY\MYDATA.DAT. On a Macintosh, DATAPATHS might be vector("MyDisk:MacAnova:", "MyDisk:Survey:") and the files searched for would be MyDisk:MacAnova:mydata.mac and MyDisk:Survey.mydata.mac.

On a Macintosh or DOS/Windows computer, DATAPATHS is pre-defined to the name of the Macintosh folder or directory (the folder or directory where the executable file is located). On Unix its value is the name of the directory where the standard macro files are located. See topic customize for information on changing this default.

You may be able to use a file name of the abbreviated form "~/Name" (DOS/Windows and Unix) or "~:Name" (Macintosh), where "Name" is the name of a file. For this to work, a CHARACTER variable HOME must exist and contain the complete name (path) of a directory. Under DOS, for example, if HOME is "C:/SURVEY", "~/mydata.dat" becomes "C:/SURVEY/mydata.dat". On a Macintosh, if HOME is "MyDisk:Survey", "~:mydata.dat" is equivalent to "MyDisk:Survey:mydata.dat".

On a Macintosh or DOS/Windows computer, HOME is pre-defined as the full name of the directory where the executable Macintosh program is located. On Unix, HOME is predefined to be the users home directory (environmental variable $HOME). On Unix and DOS/Windows you can override the default using command line option -home. See launching. See topic customize for information on changing the default.

Most commands writing to a file recognize keyword phrase 'new:T' as specifying that any information already in the file is to be discarded. On these commands, omitting 'new:T' or using 'new:F' results in information being added to the file after whatever is there.

Default Directories or Folders

Generally MacAnova first looks for the named file in the current default directory or folder, unless the supplied file name is a "path" name, specifying a directory or folder and a file.

On DOS/Windows and Unix, the initial default directory is the default directory at the time MacAnova is started up (on DOS, this might be different if MacAnova is started by a *.BAT file that includes a CD command).

fchplot(FileName,x,y [,...]) is equivalent to chplot(x,y,file:FileName [,...]). The use of fcolplot is discouraged; use colplot instead.

See chplot() for details on arguments and keywords.

2.91 fcolplot

Keywords: plotting, files
Usage: fcolplot(fileName, x [, graphics keyword phrases]), fileName a CHARACTER scalar, x a REAL matrix

fcolplot(FileName,x [,...]) is equivalent to colplot(x,file:FileName [,...]). The use of fcolplot is discouraged; use colplot instead.

See macro colplot for details on arguments and keywords.

2.92 files

Keywords: files, input, output, missing values
Usage: Type help(files) for information on file names, abbreviated names of the form "~/fileName" and default directories or folders.

This topic describes the use of files. Much of it is fairly technical and not of great interest to the casual user. It has sections on file names and default directories or folders and a little information on file format. See topics data files and macro files for descriptions of formats for data files and macro files.

File Names

Most commands and macros that read or write information from or to files require that the file name be specified by a quoted string or CHARACTER variable such as "macanova.dat". The file name you provide must be a legal file name for your system. If MacAnova decides it is not a legal name it prints the message

ERROR: improper file name xxxxxxx.

On some systems, the checks exclude some technically legal file names. For example, on Unix, "-savefile" is not recognized as legal.

The Windows version understands long file names when running under Windows 95 and Windows NT, but only classic MSDOS file names when it is running under Windows.

In versions with windows (Macintosh, Windows, Motif), you can always use the null file name "". This brings up a dialog box allowing you to select a folder and a file.

fastanova(Model [,...], silent:T) does computations, creating side effect variables, but prints nothing except actual error messages.

Keyword phrase 'coefs:F' cannot be used with fastanova().

See topic 'models' for information on specifying Model.

Coefficients may be retrieved by coefs(); standard errors are not available.

The contrast() function does not work properly after fastanova(). coefs() may or may not give the correct answers; fastanova() will warn you when coefs() will fail. cellstats() results include the estimated values for the missing data.

The iterative fitting method used by fastanova() is faster than Gram-Schmidt for large unbalanced data sets. The time used is roughly proportional to length(y)*nterms, where y is the response variable, and nterms is the number of terms in the model (the not the model degrees of freedom). Thus, fastanova() gives greatest speed improvements for models with relatively few terms, each with relatively many degrees of freedom. fastanova() is least effective for models with many terms, each with few degrees of freedom. In fact, it may be slower than anova() for such models.

2.89 fboxplot

Keywords: plotting, files
Usage: fboxplot(fileName,x1,x2,...,xk [,graphics keyword phrases]), fileName a CHARACTER scalar, xi's REAL vectors
fboxplot(fileName, Struc [,graphics keyword phrases]), Struc a structure with REAL vector components

fboxplot(FileName,arg1,[,...]) is the same asboxplot(arg1,file:FileName [,...]). The use of fboxplot() is discouraged; use boxplot() instead.

See boxplot() for details on arguments and keywords.

2.90 fchplot

Keywords: plotting, files
Usage: fchplot(fileName,x,y [, c] [, graphics keyword phrases]), fileName a CHARACTER scalar, x a REAL vector or scalar, y a real vector or matrix and c a integer or CHARACTER scalar, vector, or matrix

A[J] is also a factor with k levels, even if max(A[J]) < k.

If A is a factor with k levels, A[j] <- newvalue is legal only if newvalue is an integer between 1 and k. The number of levels associated with A will not change even if max(A) < k after the replacement.

In both these last two situations, it is possible to create a factor whose actual maximum level is less than k. However, the actual maximum factor level will be used in any analysis.

See also makefactor, models.

2.88 fastanova

Keywords: glm, anova
Usage: fastanova([Model] [,print:F or silent:T,fstats:T,pvals:T])

fastanova(Model) computes the analysis of variance table for the model given in the CHARACTER variable Model. No variates (only factors) can be in the model. It uses an iterative algorithm rather than the modified Gram-Schmidt used by anova().

Caution: If there are empty cells in the design, the degrees of freedom and hence the mean squares may be in error.

fastanova(Model [,...], fstats:T) prints F-statistics and P values.

fastanova(Model [,...], pvals:T) prints P values

fastanova(Model [,...], fstats:T,pvals:F) prints F-statistics.

If option 'fstats' has value True, F-statistics will be printed unless 'fstats:F' is an argument. See setoptions() and glm.

If option 'pvals' has value True, P values will be printed unless 'pvals:F' is an argument. See setoptions() and glm.

fastanova(), with no Model specified, uses the model from the most recent GLM command such as anova() or poisson() or the model in STRMODEL.

fastanova(Model,maxiter:N,epsilon:eps) where N is a positive integer and eps is a small REAL scalar, iterates no more than N times (default is 25) on each fit and uses the value of eps (default is 1e-6) to determine when convergence has occurred. Either keyword can appear without the other.

fastanova(Model [,...], print:F) is the same as fastanova(Model [,...]) except that most printing is suppressed and the only result is to set the side effect variables.

```
Cmd> i <- 0; cmd <- "i <- i+1;print(i);evaluate(cmd)";evaluate(cmd)
```

will be 49, followed by an error message.

See also macros.

2.87 factor

Keywords: glm, anova
Usage: factor(n1 [, n2, ...]) where n1, n2, ... are REAL scalars or vectors, all of whose elements are positive integers.

factor(a), where a is a REAL vector creates a vector with contents identical to a except that the new vector is marked as a "factor". If a is not a vector, all dimensions beyond the first must be 1 and the result has the same dimensions.

The contents of a must be positive integers <= 32767. The number of classes in the factor will be the largest integer in a. Thus both factor(vector(1,2,4)) and factor(vector(1,2,3,4)) will produce factors with four classes, although only three of the classes are present in factor(vector(1,2,4)).

If a is a LOGICAL vector, factor(a) is equivalent to factor(a+1), that is, False and True are translated to levels 1 and 2, respectively. The number of levels will always be taken to be 2, even if a consists of all False.

factor(vec1, vec2, ... veck) is equivalent to factor(vector(vec1, vec2, ... veck)) where vec1, ..., veck are all vectors.

When a factor is included in a model for a non-regression GLM (generalized linear or linear model) command, for example, anova() or poisson(), its values are taken to specify levels of a categorical (non-quantitative) variable, that is, classes or categories. A vector in a model that has not been turned into a factor using factor() is called a "variate" and its values are taken to specify quantities, even if they are all positive integers. In regress(), and screen() factors are treated the same as variates -- that is the levels are viewed as quantitative.

A common mistake in using GLM commands is to forget to use factor() to turn vectors of factor levels into factors. This error results in their being treated as variates with single degrees of freedom. In a model which includes both factors and variates, the variates are often referred to as "covariates".

If A is a factor with k levels and J is an appropriate subscript for A (for example, J might be A != 3, vector(1,run(3,length(A))) or -2), then

```
Example:
  Cmd> names <- enterchars(Henry Susan Bill Rene)

enterchars is implemented as a pre-defined macro using vecread().

See also vecread(), enter.
```

2.85 error

Keywords: output
Usage: `error(a, b, ...[,format:Fmt or nsig:m, header:F, labels:F, missing:missStr])`, Fmt and missStr CHARACTER scalars, m > 0 integer

```
error() is identical to print() except it terminates execution of the
current line or macro.  It is useful for reporting errors recognized in
a macro.

Example:
  Cmd> for(i,run(n)){if(x[i] >= 0){y[i] <- sqrt(x[i]);;}else{
       error("ERROR: attempt to take square root of negative number")}

See also print(), write(), paste().
```

2.86 evaluate

Keywords: general, syntax, macros, control
Usage: `evaluate(cmds)`, cmds a quoted string or CHARACTER scalar.

```
evaluate(Cmds), where Cmds is a quoted string or CHARACTER variable
consisting of one or more MacAnova commands, executes the commands and
returns the value of the last one.  Unlike macro expansion, symbols
starting with '$' in Cmds have no special significance.
  Cmd> b <- evaluate("a <- PI;sqrt(a)");print(a,b)
  a:
  (1)       3.1416
  b:
  (1)       1.7725

In most situations, including the example just given, <<Cmds>> is
equivalent to evaluate(Cmds).  See syntax.

evaluate() can be used recursively, in the sense that among the commands
it evaluates is another use of evaluate().  The depth of such a
recursion + the number of out-of-line macros currently being evaluated
must be no more than 49.  Thus, if there are no out-of-line macros in
use, the last value of i printed by
```

2.81 else

Keywords: syntax, control
Usage: `if (Logical1)command1;command2;... [elseif (Logical2)...[else...]]`

See 'if'

2.82 elseif

Keywords: syntax, control
Usage: `if (Logical1)command1;command2;... [elseif (Logical2)...[else...]]`

See 'if'

2.83 enter

Keywords: input
Usage: `x <- enter(val1 val2 val3 ...)` , valI a number or ?, no separating commas

x <- enter(val1 val2 ...) is equivalent to x <- vector(val1, val2, ...) allowing easy entry of data without the need to type commas. val1, val2, ... should be numbers or ?.

Example:

```
  Cmd> wts <- enter(145.2 162.1 133.5 121.9 99.8 188.9)
```

enter is implemented as a pre-defined macro using vecread().

See also vecread(), enterchars.

2.84 enterchars

Keywords: input
Usage: `x <- enterchars(str1 str2 str3 ...)` , strI a non-quoted sequence of visible characters, separated by spaces.

x <- enterchars(str1 str2 ...) is equivalent to x <- vector("str1", "str2", ...) allowing easy entry of CHARACTER data without the need to type quotes (") or commas. str1, str2, ... are sequences of visible characters, separated by spaces. You cannot use enterchars to enter CHARACTER elements containing spaces, tabs or other invisible characters.

Macro edit is pre-defined only in the Unix and extended memory DOS (DJGPP) versions of MacAnova. The Unix version is included in file MacAnova.mac.

To edit a macro, say mymacro, in a windowed version (Macintosh, Windows Motif), print it to the command/output window as follows:

```
Cmd> macrowrite(CONSOLE, mymacro)
```

Then edit the macro, copy to the clipboard the entire macro including the header line (mymacro MACRO) and trailer line (%mymacro%), and then do the following:

```
Cmd> mymacro <- macroread(string:CLIPBOARD)
```

See also macrowrite(), macroread(), clipboard.

2.79 eigen

Keywords: matrix algebra
Usage: eigen(x), x a REAL symmetric matrix with no MISSING values

eigen(x) computes an eigenvector/eigenvalue decomposition of the symmetric matrix x. The result is a structure with two REAL components, 'values' and 'vectors'. Component 'values' is a vector containing the eigenvalues in decreasing order (values[i] >= values[i+1]). Component 'vectors' is a matrix whose columns are the eigenvectors of x with vectors[,j] corresponding to values[j]. The eigenvectors are orthonormal, even if there are repeated eigenvalues.

See also eigenvals(), trideigen(), releigen(), and releigenvals().

2.80 eigenvals

Keywords: matrix algebra
Usage: eigenvals(x), x a REAL symmetric matrix with no MISSING values

eigenvals(x) computes a REAL vector containing the eigenvalues of the symmetric matrix x in decreasing order.

See also det(), eigen(), trideigen(), releigen(), and releigenvals().

files, MACANOBC.EXE, MACANODJ.EXE or MACANOWX.EXE which can co-exist.

See topic customize.

Unless you use command line option -home (see launching) or include -home in environmental variable MACANOVA (see customize), MacAnova pre-defines CHARACTER variable HOME to contain the name of the Macanova directory, that is the directory where MACANOBC.EXE, MACANODJ.EXE, and/or MACANOWX.EXE is located. HOME is used by to expand file names of the form "~/path" or "~\\path" by substituting the value of HOME for '~'. This allows you to refer to files such as MACANOVA.INI as "~/macanova.ini", even if you have changed directories. If you redefine HOME, it changes the expansion of "~/" and "~\\". See topic files.

Pre-defined variables DATAPATH, and DATAPATHS are initialized with path names derived from the MacAnova directory path name unless options -dpath or -mpath are set on the command line (see launching) or environmental variable MACANOVA (see customize).

2.78 edit

Keywords: general
Usage: edit(obj [, T]), obj a macro or a REAL variable edit(0)
edit()

edit(obj) where obj is a REAL vector, matrix, array, or a macro, writes a temporary file and invokes an editor to edit that file. When editing is done, edit returns as value the edited contents of the file. Thus, for example, 'mymacro <- edit(mymacro)' allows you to change or correct macro mymacro. edit is implemented as a macro which is pre-defined only in some non-windowed versions of MacAnova. It is included in file MacAnova.mac distributed with MacAnova.

If you change the dimensions of a vector, matrix, or array, you must edit the header line to reflect the change before quitting the editor.

edit(obj,T) is equivalent to 'obj <- edit(obj)'.

mynewmacro <- edit() (with no arguments) invokes an editor on a temporary file with a header specifying a macro with a single blank line. You can enter and edit a macro, being sure to change the number on line 1 to reflect the actual number of lines in the macro.

x <- edit(0) invokes an editor on an empty temporary file and then performs vecread() on that file, under the assumption that the user has edited in data that may be read in this manner. This allows you to use an editor to enter data.

You can specify the editor to be used by creating a CHARACTER variable with name 'EDITOR' by, for example, EDITOR <- "emacs". If EDITOR has not been set, the default editor is "vi" on Unix and "edit" for DOS.

windowing library (see topic 'copyright'. It allows multiple command/output windows and high resolution graphics windows with Copy, Paste and Undo capability. It requires a 32 bit Windows system such as Windows 3.1 with Win32S, Windows 95 or Windows NT. Commands are typed immediately after the prompt in a scrollable command/output window. If you are running Windows 3.1, you will need to install Win32S. This is available free from Microsoft. See file win32s.doc distributed with MacAnova for details. Win32S is not needed for Windows 95 or Windows NT.

The executable file is usually named MACANOWX.EXE.

The WXBCPP version has File, Edit and Windows menus which are patterned after the Macintosh version but do not include all items. Item New on the Windows menu allows you to open up to eight command/output windows. You can save a command/output window to disk (Save or Save As on the File menu) or read a file, say output saved on a previous run, into a new window using Open on the File menu. The Windows clipboard is "connected" to special CHARACTER variable CLIPBOARD. See topic clipboard.

It has no intrinsic limitations on the size of variables, using RAM and virtual memory as needed.

It allows up to eight graphics windows, each of which can be printed or Copied to the clipboard. The are no panel of graph windows as on the Macintosh.

Editing of commands is done with the mouse and keyboard in a command/output window.

shell(command,interact:F) and shell(command,keep:T) do not work under Windows 3.1 and Windows 95. It is possible that they work under Windows NT, but that has not been tested. shell(command,interact:T) and lines prefixed with '!' appear to behave somewhat differently, depending on the operating system. Problems remain to be worked out. See shell().

See topic wx for more information.

Features Common to All Versions

MacAnova recognizes '/' in file names (for instance, c:/mv/macanova.dat instead of c:\mv\macanova.dat). This is desirable, since to use '\' in a quoted string it must be doubled ("c:\\mv\\macanova.dat"). See syntax.

Various command line arguments are recognized allowing automatic restoring of a workspace, suppressing the banner, changing default file names, etc. See topic launching.

MacAnova uses any default options or file or path names in environmental variable MACANOVA. See topic customize.

The startup file is MACANOVA.INI in the same directory as the executable

The executable file is usually named MACANOBC.EXE.

Version BCPP can draw high resolution plots on a variety of graphics modes (CGA, EGA, VGA, 8514 and Hercules). Keyword 'screendump' is not available on plotting commands, and hence there is no easy way to save high resolution graphs. However, when you launch BCPP MacAnova from Windows, Alt PrintScreen copies the high resolution screen to the clip board from which it can be pasted into a Word Perfect or other program's window.

It has no command editing or "history" using arrow keys.

You can execute DOS commands by prefixing the line with '!' in the first position after the prompt or by using the command shell(). However, memory limitations restrict what you can actually do. shell(cmd,keep:T) is not implemented and shell(cmd,interact:F) is the same as shell(cmd, interact:T). See shell().

Version DJGPP (extended memory)

Version DJGPP is compiled using a version of Gnu gcc developed by D. J. Delorie. Because it runs in protected mode it requires a 80386 or better processor. Because it can access all available memory and in fact uses "virtual memory" on the hard disk if necessary, the size of variables is limited only by hardware limitations.

The executable file is usually named MACANODJ.EXE.

DJGPP MacAnova works with a variety of graphic displays, including VGA. Keyword phrase screendump:fileName on plotting commands allows you to create PCX files which can be edited under Windows and included in word processor documents.

Version DJGPP has command editing implemented using the arrow keys and editor commands based on either the Emacs or Vi editor commands (Emacs and Vi are Unix editors). See topic 'unix' for details. The only difference from the Unix version is that the special file for customizing keymaps must be "INPUTRC" (not ".inputrc") in the same directory as MACANODJ.EXE.

You can use shell() and command lines starting with '!' as for BCPP, but without the memory limitations. shell(cmd,keep:T) returns output from the program executed. You must use shell(cmd,interact:T) if the program executed requires any input. When in doubt, use interact:T. This feature appears to works somewhat differently under Windows 95 as it does when running under Windows 3.1.

Macro edit (see topic edit) is predefined in the DJGPP version to allow easy editing of macros and data without exiting Macanova. By default it uses the DOS program Edit but this can be changed by setting CHARACTER variable EDITOR to the path name of a different editor.

WXBCPP Version (Windows version)

This version is compiled under Borland C++ 5.02 using the WxWin

'dim1', 'dim2', ... If xi is the i-th component of x, the i-th component of dim(x)[j] is dim(xi)[j] (or 0 if ndims(xi) < j).

If x is a macro or built-in function, dim(x) is always 1.

If x is a NULL variable or a structure all of whose components are NULL, dim(x) is NULL.

See also length(), ndims(), structures.

2.76 dmat

Keywords: matrix algebra, variables
Usage: dmat(n,val), n > 0 integer, val a REAL, CHARACTER or LOGICAL scalar dmat(vec), vec a REAL, CHARACTER or LOGICAL vector.

dmat(n,val), where n is a positive integer and val is a scalar (length(val) = 1), produces an n by n diagonal matrix with val down the diagonal.

dmat(a) where a is a vector of length n, a n by 1 matrix, or a 1 by n matrix, produces a n by n diagonal matrix with the elements of a down the diagonal.

Example: dmat(5,1) and dmat(rep(1,5)) both produce an identity matrix.

See also diag().

2.77 dos-windows

Keywords: general
Usage: Type help(dos-windows) for a summary of features specific to DOS/Windows versions of MacAnova.

There are three released versions of MacAnova for IBM compatible computers. Two (BCPP and DJGPP) are designed to run under the MSDOS operating system and, although they can be launched from Windows, do not use Windows features. The third (WXBCPP) is designed to run under Windows and Windows 95 and incorporates many of the familiar multi-window features present in the Macintosh version. All have the full range of MacAnova commands.

Version BCPP (limited memory)

Version BCPP is compiled using Borland C++ 4.5 (BCPP). It runs on almost any PC compatible (80286, 80386, 80486, Pentium) with sufficient memory and a hard disk. It does not use extended memory and the size of individual variables is limited to about 65000 bytes (8125 REAL items).

```
boxcoxvec <- macroread("design.mac","boxcoxvec").

See also macros, macroread(), getmacros, help(), usage(), macrousage().
```

2.73 det

Keywords: matrix algebra
Usage: `det(x)`, where x is a REAL square matrix with no MISSING values

`det(x)` computes the determinant of the square REAL matrix x. MISSING values are not allowed in x.

See also matrices, trace().

2.74 diag

Keywords: matrix algebra, variables
Usage: `diag(A)`, A a matrix.

`diag(a)` creates a vector consisting of the diagonal elements a[1,1], a[2,2], ... of matrix a, which does not need to be square. The length of the result is min(nrows(a),ncols(a)). Matrix a may be REAL (most common), LOGICAL or CHARACTER.

`diag(a)` is defined for an array with more than 2 dimensions, as long as there are only two dimensions with size > 1, interpreted as the number of rows and columns. Thus, after manova(model) command, diag(SS[3,,]) is the diagonal of the sums of squares and cross-products matrix associated with term 3 in the model.

See also dmat(), trace().

2.75 dim

Keywords: variables, null variables
Usage: `dim(x)`

`dim(x)` creates a vector containing the dimensions of x. For example, if x is a vector of length 10, dim(x) has value 10. If x is a 5 by 7 matrix, dim(x) has value equivalent to vector(5,7) and dim(x)[1] and dim(x)[2] have values 5 and 7, respectively.

If x is a structure, dim(x) is a structure, each of whose components is a structure with the same shape as x. If maxdims is the largest number of dimensions of any component of x, dim(x) has maxdims components named

```
  V[g2] = 24*n*(n-1)^2/((n-3)*(n-2)*(n+2)*(n+5))

See also structures, keywords.
```

2.72 design

Keywords: anova, glm, regression
Usage: Type 'help(design)' for information on functions and macros useful in experimental design

```
Functions for sample size and power computations:
  power(), power2(), samplesize(), cumF(), cumstu(), cumchi()

Type, for example, 'usage(power)' or 'help(power)', to get a thumbnail
sketch or a complete description of power().

File design.mac, distributed with MacAnova, contains the following
macros:
                  Macros Useful in Designing Experiments
  confound2     Confounds a 2 series factorial into blocks
  confound3     Confounds a 3 series factorial into blocks
  aliases2      Gets aliases in fractioned 2 series
  aliases3      Gets aliases in fractioned 3 series
  allaliases2   Complete aliases structure in fractioned 2 series factorial
  ffdesign2     Determines factor/level combinations to use for
                fractioned 2 series
  choosegen2    Chooses generators for a 2 series fractional factorial
  ems           Computes the expected mean squares for the terms in the
                ANOVA in a specified mode with specified random
                terms

                     Macros for Statistical Analysis
  randt         Does a permutation t-test
  randsign      Does a permutation paired t-test
  pairedcomp    Does paired comparisons and generates an "underline" diagram
  rscanon       Does canonical analysis of 2nd order response surface design
  boxcoxvec     Gets the error SS for several boxcox transformations
  all3anova     Fits all hierarchical models in a three factor anova and
                sorts them by Cp
  all4anova     Fits all hierarchical models in a four factor anova and
                sorts them by Cp
  quadmax       Finds the location of the maximum of a quadratic
                function, with optional linear equality and inequality
                constraints on the solution
  quadmaxlin    Finds the maximum of x'Ax + b'x subject to Cx = y (used
                by quadmax)
  colproduct    Computes all element-wise products of the columns of two
                matrices (used by ems)
  makemat       Computes various basis matrices (used by ems)

These can be retrieved by, for example, getmacros(boxcoxvec) or
```

not a structure.

If Data is multidimensional (a matrix or array) with dimensions n1, n2, ..., nk, each component of the result (or the result itself if only one statistic is requested) is an array with dimensions n2, n3, ..., nk, that is, it has one fewer dimensions than Data. Each statistic describes all values with the last k-1 subscripts fixed (a column when Data is a matrix).

If Data itself is a structure, each component of the result (or the result itself if only one statistic is requested) is itself a structure with the same shape as Data, whose components contain summary values for the corresponding component of Data.

Examples:

```
  Cmd> meanx <- describe(x)$mean; varx <- describe(x, var:T)
```

compute the mean and variance of x.

```
  Cmd> medians <- describe(split(y,a))$median
```

and

```
  Cmd> medians <- describe(split(y,a), median:T)
```

both compute a structure, each of whose elements is the median of the values of y corresponding to a level of factor a. The first does less computing of results you aren't saving.

```
  Cmd> describe(x, mean:T, var:T)
```

and

```
  Cmd> describe(x)[vector(7,8)]
```

are equivalent, except the latter does much unnecessary computing because it computes and then discards the extremes, the quartiles and the median.

g1 and g2 are based on Fisher's k-statistics:

```
  g1 = k3/k2^1.5 and g2 = k4/s2^2, where
  k2 = var = s2/(n-1), k3 = n*s3/((n-1)*(n-2)), and
  k4 = (n*(n+1)*s4 - 3*(n-1)*s2^2)/((n-1)*(n-2)*(n-3))
```

g1 and g2 are not identical to sqrt(beta1) = m3/m2^1.5 and beta2 = m4/m2^2 - 3 which are also often used to measure skewness and kurtosis.

Expressed in terms of m2, m3 and m4 or sqrt(beta1) and beta2:

```
  g1 = (sqrt(n*(n-1))/(n-2))*m3/m2^1.5
     = (sqrt(n*(n-1))/(n-2))*sqrt(beta1)
  g2 = ((n^2-1)/((n-2)*(n-3)))*(m4/m2^2 - 3 + 6/(n+1))
     = ((n^2-1)/((n-2)*(n-3)))*(beta2 + 6/(n+1))
```

When n <= 2, g1 is computed to be 0. When n <= 3, g2 is computed to be 0.

g1 and g2 are sometimes used to test the null hypothesis that a sample comes from a normal population. If the data are a random sample from a normal distribution, then g1 and g2 have mean 0 and

```
  V[g1] = 6*n*(n-1)/((n-2)*(n+1)*(n+3))
```

2.71 describe

Keywords: descriptive statistics

Usage: describe(data [,all:T,n:T or F,min:T or F,max:T or F,q1:T or F, q2:T or F,median:T or F,mean:T or F,var:T or F, stddev:T or F, m3:T or F, m3:T or F, m4:T or F, g1:T or F, g2:T or F]), where data is REAL or a structure with REAL components; F's should be used only with all:T

describe(Data) computes statistics describing the data in the REAL vector or array Data.

The value of describe(Data) is a structure with following components:

n	sample size, excluding MISSING values
min	minimum
q1	lower quartile
median	median
q3	upper quartile
max	maximum
mean	average
var	variance (with divisor of n-1)

You can specify which of the above statistics to compute using keyword phrases. For example, describe(Data, mean:T), for example, has the same result as describe(x)$mean, except that no unwanted statistics are computed, and describe(Data,mean:T,var:T) returns a structure with components mean and var without computing other statistics.

There are additional statistics that may be computed only by using keyword phrases.

stddev	standard deviation = sqrt(var)
m2	sum((x-xbar)^2)/n = s2/n = (n-1)*var/n
m3	sum((x-xbar)^3)/n = s3/n
m4	sum((x-xbar)^4)/n = s4/n
g1	coefficient of skewness (see below)
g2	coefficient of kurtosis (see below)

For example, describe(x, g1:T, g2:T) returns a structure with components g1 and g2 containing the skewness and kurtosis of x.

describe(Data, all:T) returns a structure with the 8 standard components plus components stdev, m2, m3, m4, g1 and g2. You can suppress any component by, for example, median:F. Thus describe(Data, all:T, q1:F, median:F, q3:F), returns structure containing all statistics except the median and the quartiles.

You can use 'm1' instead of 'mean' and 'q2' instead of 'median' when specifying what to compute; however, when other statistics are also computed, the components still have names 'mean' and 'median'. Thus describe(x,m1:T,q2:T) is equivalent to describe(x,mean:T,median:T).

If only one statistic is requested, the result is a REAL variable and

```
component: b
  component: pi
(1,1)       3.1416
  component: e
(1)       2.7183

Cmd> mystruc_b <- matread("data.txt", "mystruc$b",quiet:T)

Cmd> print(mystruc_b)
mystruc_b:
component: pi
(1,1)       3.1416
component: e
(1)       2.7183
```

quiet:T in these examples suppresses echoing the header line and comments, See matread().

See also matread(), matprint(), matwrite(), files, macro files.

2.70 delete

Keywords: variables, character variables
Usage: delete(var1[,var2, ...] [,all:T, real:T or F,char:T or F, logical:T or F, structure:T or F,macro:T or F, graph:T or F]), F's used only with all:T

delete(var1,var2,...,vark) deletes the variables given as arguments and frees the memory they use for other purposes. The variables can be of any type including MACRO or GRAPH.

You can use keywords 'real', 'char', 'logical', 'structure', 'macro', 'graph' and 'all' with LOGICAL values to delete classes of variables. For example, delete(real:T, logical:T) deletes all REAL and all LOGICAL variables and delete(all:T, macros:F) deletes everything except macros. It is illegal to have both keyword phrases and variables as arguments.

delete(var, return:T) deletes variable var but returns a copy. This is intended to be used as the last command in a macro when the value the macro is supposed to be var, as in the following

```
Cmd> mymacro <- macro("@tmp <- ($1)+($2)
       print(describe(@tmp))
       delete(@tmp,return:T)")
```

mymacro(x,y) prints descriptive statistics for x+y and returns x+y as value.

See also list(), listbrief(), macro(), macros.

```
  without any double
  quotes

  mystruc  2  STRUCTURE
  ) this is a structure with two components, a and b
  ) The blank line before the header of each component is required

  mystruc$a  2 QUOTED COLUMNS
  ) character vector of length 2
  ) Two quoted fields
  "The quick brown fox" "Jumps over the lazy dog"

  mystruc$b  2 STRUCTURE
  ) This component is a structure with two components, pi and e

  mystruc$b$pi  1   1
  ) 1 by 1 matrix
  3.14159265358979

  mystruc$b$e   1
  ) vector of length 1
  2.71828182845905

Examples of reading data sets from this file
  Cmd> sampledata <- matread("data.txt","sampledata", quiet:T)

  Cmd> print(sampledata)
  sampledata:
  (1,1)         34.5          0.17          3.5
  (2,1)         45.2       MISSING          4.7
  (3,1)         23.1         0.883          3.2
  (4,1)         20.1         0.401          5.8

  Cmd> samplechars <- matread("data.txt","samplechars",quiet:T)

  Cmd> print(samplechars)
  samplechars:
  (1,1) "This"
  (1,2) "is"
  (1,3) "by-fields"
  (1,4) "format"
  (2,1) "without"
  (2,2) "any"
  (2,3) "double"
  (2,4) "quotes"

  Cmd> mystruc <- matread("data.txt","mystruc",quiet:T)

  Cmd> print(mystruc)
  mystruc:
  component: a
  (1) "The quick brown fox"
  (2) "Jumps over the lazy dog"
```

Each stretch of "non-white" characters on a line is considered to be an element. This format is signaled by the presence of CHARACTER on the header and the presence of a ")%s..." format among the comment lines. The number of fields on a line is the number of "%s"'s in the format.

By quoted fields
Each element must be enclosed in quotes ("...") and elements in a line are separated by spaces, tabs, and possibly a comma. This format is signaled by the presence of QUOTED on the header. If there is no ")%s..." format among the comment lines, the number of items expected is the size of the last dimension or 1 if the data set is a vector. If COLUMNS or COLS is on the header the default number expected is the size of the first dimension. If there is a ")%s..." format, the number of elements expected per line is no more than the number of "%s"'s in the format.

The name line for a structure data set must be of the form
strname ncomps STRUCTURE
where strName is the name of the structure and ncomps is a positive integer specifying the number of components. There must follow ncomps data sets, each in one of the formats just described, or in the format for a structure. Each component must have a name of the form strName$compName, where compName is the name of the component. If a component is a structure, then the names of its components would thus be strName$compName$compName1.

Each structure component must be preceded by at least one blank line. An individual component can be read by specifying its full name, "mystruc$b", for example.

Example of a data file, data.txt, to be read by matread().

```
info            0
) Sample data file containing REAL data set sampledata,
) CHARACTER data set samplechars, and structure mystruct

sampledata      4      3 COLUMNS
) Small REAL data set with one missing value coded as -99.
) Each line contains data for one column (COLUMNS on header)
) MISSING -99
) '4x' in the following format skips 4 characters (variable label)
)"4x%f %f %f %f"
Temp    34.5    45.2  23.1    20.1
Conc    .170    -99   .883     .401
Secs     3.5    4.7    3.2      5.8

samplechars     2      4 CHARACTER
) 4 by 2 CHARACTER matrix with each row in 2 lines containing
) 3 and 1 unquoted fields
)"%s %s %s"
This is by-fields
format
```

STRUCTURE data sets.

)"%f %f ... %f" specifies a format for each row of a REAL or LOGICAL data set that is analogous to that used by scanf in the C programming language. If the data set is CHARACTER, such a format is an error. Let N1 and Nk be the first and last dimensions. If there are fewer "%f"'s than Nk (or fewer than N1 if COLS or COLUMNS is specified), then this indicates that, for each value of the last index (first index with COLS or COLUMNS), there are several lines in the file containing data. Each such line, except possibly the last, must have the same number of data items as there are %f's. If no explicit format is given, one with Nk or N1 (if COLS or COLUMNS is on line 1 of the header) %f's is assumed. No more than 50 values can be put on a single line.

)"NNx%f %f ... %f" where NN is an integer, causes the first NN characters of each line to be skipped. This allows you to skip case labels or line numbers. Example:)"12x%f %f" skips 12 characters at the start of each line.

)"%s %s ... %s" specifies a format for each row of a CHARACTER data set. If present, the data will be expected to be in "by fields" format or "by quoted fields" (if QUOTED is on header line) format (see below). The number of "%s"'s is the maximum number of elements that will be read per line.

)"NNx%s %s ...%s" where NN is an integer causes the first NN characters of each line read to be skipped before scanning for CHARACTER data in "by fields" or "by quoted fields" format.

) MISSING XX where XX is a number indicates that XX in the data set is to be read as MISSING. The default missing value code is -99999.9999. Because only integers can be guaranteed to be represented exactly in the computer, it is preferable for XX to be an integer, positive or negative. Example:) MISSING -99 specifies MISSING is coded as -99. This is ignored for

There are three possible formats for CHARACTER data, "by lines", "by fields" and "by quoted fields:

By lines
Each element starts on a new line and is not quoted. If an element extends over more than one line, each line except the last must end with '\'. This format is signaled by the presence of CHARACTER on the header and the absence of any ")%s..." format among the comment lines.

By fields

```
) to as 'comment lines' below
) .....
```

Name specifies the name of the data set ('mydata', say) to be matched to setName, the second argument to matread().

Dims is a list of positive integers specifying the dimensions. If Dims is a single number ('mydata 20'), the data set is a vector of length Dims. If it consists of two numbers, nrows and ncols, ('mydata 20 5') the dataset is a nrows by ncols matrix. If Dims consists of p >= 3 numbers, the data set is a p-dimensional array.

It is acceptable for Dims to be '0', in which case no data is expected. When such a data set is read by matread(), the comment lines are printed and NULL is returned. A useful convention is to have the first "data set" on a file be empty, with the comments describing the remainder of the file.

Permissible keywords on the first line of the header are as follows:

COLS or COLUMNS	The data follow in transposed form. For a matrix, this is in column by column order, each column starting on a new line.
ROWS	The data follow a row at a time (constant value for first subscript), each row starting on a new line. This is the default.
FORMAT	Indicates that a Fortran format starting with '(' will follow the last comment line. It is ignored by MacAnova but would be helpful for a program written in Fortran to read the data.
LOGICAL	The data are to be interpreted as logical, with zero and non-zero values translated to False and True, respectively
CHARACTER	The data set is a CHARACTER data set in either "by fields" or "by lines" format (see below)
QUOTED	The data set is a CHARACTER data set in "by quoted fields" format (see below).
NULL	The data set is a NULL variable, containing no data, although there may be comment lines. There can be no other keywords.
STRUCTURE	The data set is a structure. There can be no other keywords

Upper and lower case letters are not distinguished in these keywords.

A vector (single dimension specified) is treated like a matrix with a single column. That is, if COLS or COLUMNS is specified it should all be on one line, and if not, every element must be on a separate line.

Conventions on Comment lines:

) LOGICAL	The data are to be interpreted as being LOGICAL, with zero and non-zero values translated to F and T, respectively. This is retained for backward compatibility and is ignored for CHARACTER, NULL or

although each column can be read as a separate variable by readcols.

A file readable by matread can contain any number of data sets, each with a header line, followed by optional comment lines and macro text lines.

Format of a File to be read by vecread() and readcols

Since macro readcols uses vecread() to read a file, there is no difference between the file formats they can handle. They can contain only REAL or CHARACTER data.

REAL data readable by vecread() and readcols consist of numbers separated by spaces or tabs, possibly on several lines. Missing values are indicated by any of the missing value symbols '?', '.' and '*'. '??', '???', ... are treated a single missing value.

Reading is terminated by running out of data or by finding the "stopping character" '!'.

Anything other than a number, a missing value symbol such as '?', or the stopping character '!' is skipped, and a warning message is printed (once). A stopping character other than '!', say '$', can be used if keyword phrase 'stop:"$"' is an argument to vecread(). Lines starting with a "skipping character" specified by an argument of the form, say, skip:"#" are also skipped. See vecread() for more information.

You can write a file vecdata.txt of REAL data that vecread() can read by

```
Cmd> print(x,new:T,file:"vecdata.txt",header:F,labels:F,missing:"?")
```

where x is a REAL vector or matrix. If x is a matrix, it is written row by row and will be read row by row by vecread(). To be readable by macro readcols, write the transpose of x as in

```
Cmd> print(x',new:T,file:"vecdata.txt",header:F,labels:F,missing:"?")
```

which writes x column by column. You can specify the format or the number of significant digits by keywords 'format' and 'nsig'. See print().

Format of a file to be read by matread()

Files readable by matread() and macroread() must have a special format, the same format in which matprint(), matwrite(), and macrowrite() create files. A single file can contain one or more data sets corresponding to any type of variable except GRAPH. This includes REAL, LOGICAL and CHARACTER data, as well as NULL variables, macros and structures.

Every data set must have a header consisting of an line with a name and other information, followed by 0 or more descriptive comment lines starting with ')', that is, it must start out having the following general form:

```
Name Dims Keywords
) 0 or more descriptive or comment lines starting with ')', referred
```

```
5    ls <- macro("listbrief($0)")
6    if(isdefined(DEGPERRAD)){delete(DEGPERRAD)}
```

If this were your startup file, it would have the following effects:
Line 1:
  Output will be printed with 6 significant digits (option 'nsig')
  Trigonometric functions will assume that angles are measured in cycles with 1 equivalent to 2*pi (option 'angles')
  Output from GLM functions such as anova(), regress(), and glmfit() will include P values, and where appropriate, F-statistics (options 'pvals' and 'fstats')
  Command restore() will not delete existing variables unless they are overwritten or unless keyword phrase 'delete:T' is used on restore() (option 'restoredel')
Line 2:
  Pre-defined CHARACTER variable DATAFILE will be redefined to be "timeser.dat". This will result in pre-defined macro getdata retrieving data from file "timeser.dat".
Line 3:
  Pre-defined CHARACTER vector MACROFILES will start with "mytser.mac", ensuring that pre-defined macro getmacros will search file mytser.mac before the standard macro files.
Line 4:
  Predefined CHARACTER variable DATAPATHS will have folder "MyDisk:Time Series:Data" added to it as an additional place to search for data or macro files to be read. On a DOS/Windows computer the name would be something like "C:/TSeries/Data" and in Unix it would probably be something like "~/TimeSeries/Data".
Line 5:
  Macro ls will be an "alias" for command listbrief()
Line 6:
  Pre-defined REAL constant DEGPERRAD with value 180/pi will be deleted.

2.69 data files

Keywords: variables, files, input, output
Usage: Type help(data files) for information on the file format expected by vecread(), readcols and matread().

This topic describes the format of a data file to be read by vecread() or readcols, and the format of a file to be read by matread(). See topic macro files for information on files that can be read by macroread(), and topic files for more technical information on file names, default directories or folders, and abbreviated file names of the form "~/filename".

Any data file that can be read by MacAnova must be a plain text or ascii file. If you create it in a word processor, be sure to save it as a text or ascii file.

A file readable by vecread() or readcols contains only one set of data,

One purpose of this option is to make it easier to use a Unix binary executable file on a computer configured differently from the one for which it was compiled. By including

    -help helpFile -mpath macroPath -dpath dataPath

in variable MACANOVA, where helpFile includes the complete path name (directory and file name) for the help file, and macroPath and dataPath are the complete path names for directories where macro and data files are kept, all installation dependent information is suppressed. This could be set in an installation's /etc/csh.login and /etc/profile files. It's o.k. for macroPath and dataPath to be the same.

Customizing by Using a Startup File

When MacAnova is launched, it searches for a "startup" file with a special name (see below). If it is found, MacAnova assumes the file contains MacAnova commands and executes it with an implicit batch(startupFile, echo:F) command before the first prompt (see batch(), launching).

Under Unix, the startup file has name ".macanova.ini" (the starting period is important) and should be in the user's home directory.

On a DOS/Windows computer, the startup file is "MACANOVA.INI" which should be in the same directory as MACANOBC.EXE, MACANODJ.EXE and/or MACANOWX.EXE. See topic dos-windows.

On a Macintosh the startup file is "MacAnova.ini" and should be in the same Folder as the MacAnova application.

See topic launching for information on how to specify an alternative startup file using command line flag -f (not on Macintosh).

The use of a startup file is completely optional. If you have one, you can put commands in it to set options such as the default output formatting, file names to replace the default values of variables DATAFILE, MACROFILES, DATAPATHS, and HOME and the units (radians, degrees, or cycles) to be used by trigonometric functions. You can also include commands to create macros that will thus always be available whenever you launch MacAnova.

The version of the startup file distributed with MacAnova does nothing as it stands, since every action in it is in an if(F){...} clause. You can activate actions in the file by editing it to change some or all of the if(F){...} to if(T){...} using any text editor. If you use a word processor, the file must be saved as a text or ASCII file.

Here is a simple example of a possible MacAnova startup file (the line numbers are for easy reference but are not part of the file)

```
Line #
  1    setoptions(nsig:6,angles:"cycles",pvals:T,fstats:T,restoredel:F)
  2    DATAFILE <- "timeser.dat"
  3    addmacrofile("mytser.mac")
  4    adddatapath("MyDisk:Time Series:Data") #Macintosh form
```

When you have K independent normal samples of size n, all with the same variance, you can test the null hypothesis that all means are equal by the studentized range statistic computed as Q <- (max(xbars) - min(xbars))/sqrt(Ssq/n). This is an alternative to the ANOVA F-statistic. You can compute the P value based on Q as 1 - cumstudrng(Q,K,K*(n-1)). Here xbars is a vector containing the K sample means and Ssq is the pooled estimate of variance. See invstudrng() for computing critical values for Q.

cumstudrng(x, K, Df, epsilon:eps), where eps is a small positive scalar, does the same computation with accuracy influenced by eps. The smaller the value of eps, the more accurate the result should be, but the longer it will take to compute it. The default value of eps is 0.0000001.

See also invstudrng(), cumstu().

2.68 customize

Keywords: control, general
Usage: Type help(customize) for information on using environmental variable MACANOVA or preparing a special startup file

There are two ways to customize some aspects of MacAnova -- setting environmental variable MACANOVA (not on a Macintosh) or preparing a special startup file MacAnova.ini (.macanova.ini on Unix).

Environmental Variable MACANOVA

On DOS/Windows and Unix computers, MacAnova recognizes and uses the value of an environmental variable MACANOVA. Its value should be a list of command line options such as '-l 26 -w 75 -q' (see topic launching for details on command line options). These are scanned before the command line options and thus are overridden by options on the command line. You can change the default values of several pre-defined variables and options.

On DOS/Windows, to use this feature you need to put a line like the following in your AUTOEXEC.BAT file.

```
SET MACANOVA=-l 26 -w 75 -mpath c:\macanova\macros
```

Include only options or file or path names whose defaults you want to change.

If your Unix shell is csh or a variant such as tcsh, you should put a line similar to the following in file .cshrc in your home directory:

```
setenv MACANOVA '-l 26 -w 75 -mpath ~/macanova/macros'
```

If your Unix shell is sh or a variant such as ksh, you should put a line similar to the following in file .profile in your home directory:

```
MACANOVA='-l 26 -w 75 -mpath ~/macanova/macros';export MACANOVA
```

2.66 cumstu

Keywords: probabilities
Usage: cumstu(x,df), x and df REAL, elements of df > 0
cumstu(x,df,delta), x, df > 0, delta REAL

cumstu(Val,df) computes P(t <= Val) where t is a Student's t random variable with df degrees of freedom. Val and df can be scalars, vectors, matrices or arrays, but must have the same size and shape if neither is a scalar. If both are scalars, the result is a scalar; if there is a non-scalar argument, the result has the same size and shape as that argument.

The degrees of freedom must be positive, but not necessarily integral. Upper tail areas of t can be computed as 1 - cumstu(Val,df). Two tailed P values for an observed t statistic Val can be computed with the macro twotailt(Val,df) or as 2*(1 - cumstu(abs(Val),df)).

Example:
```
  Cmd> 2*(1 - cumstu(sqrt(n)*(xbar - 10)/stdev)) # test H0: mu = 10
```

cumstu(Val,df,delta) computes P(t <= Val) where t is a non-central Student's t random variable with df degrees of freedom and noncentrality parameter delta. All three arguments can be scalars, vectors, matrices or arrays, but any non-scalar arguments must have the same size and shape which will be the size and shape of the result.

Example:
```
  Cmd> 1 - cumstu(invstu(.95,df),df,sqrt(n)*(mu-10)/sigma) # power
```

See also twotailt and invstu().

2.67 cumstudrng

Keywords: probabilities, comparisons
Usage: cumstudrng(x, ngroup, errorDf [,epsilon:eps]) where x is REAL, elements of ngroup integers >= 2, elements of errorDf >= 1, eps > 0 small

cumstudrng(x, K, Df) computes the probability that Q <= x, where Q is a Studentized range based on K normal variates and an independent estimate of variance with Df degrees of freedom. All three arguments must be REAL. K must consist of integers >= 2, and the elements of Df must be >= 1, not necessarily integers. The value is 0 for any x <= 0. For any element of Df > 1000, the asymptotic value (Df = infinity) is used.

Any of the arguments x, K or Df that are not scalars must be vectors, matrices or arrays all of the same size and shape.

cumstudrng(x,2,Df) should be the same as 2*cumstu(x/sqrt(2),Df) - 1 except for computational error.

or arrays with the same size and shape which will also be the size and shape of the result.

The elements of alpha must be positive. cumchi(x,df) is equivalent to cumgamma(x/2,df/2).

Example:
```
  Cmd> 1 - cumgamma(13.27, 15.5)  # alpha = 15.5
```

See also invgamma(), cumchi(), invchi().

2.64 cumnor

Keywords: probabilities
Usage: cumnor(x), x REAL

cumnor(Val) computes the probabilities that a standard normal (mean 0, variance 1) random variable would be less than the elements of the vector, matrix, or array Val. The size and shape of the result is the same as that of Val.

Upper tail areas of the normal can be computed as 1 - cumnor(Val). Two-tailed P values associated with an observed value of z, may be computed as 2*(1 - cumnor(abs(z))).

Example:
```
  Cmd> 2*(1 - cumnor(1.96)) has value approximately .05.
```

See also invnor().

2.65 cumpoi

Keywords: probabilities
Usage: cumpoi(x,mu), x and mu REAL, elements of mu > 0

cumpoi(Val,lambda) computes the probability that a Poisson random variable with mean lambda would be less than or equal to the elements of the vector, matrix or array Val. If Val and lambda are both not scalars, they must be the same size and shape which will also be the size and shape of the result.

Parameter lambda must be positive.

Example:
```
  Cmd> 1 - cumpoi(0,10) # probability x > 0 when mean of x is 10
```

variable. The file specified by DATAFILE must be in the form readable by matread(). See matread() and topic data files.

y <- getdata(DataName,quiet:T) suppresses the printing of any descriptive comments associated with the data set in the file.

The data set name dataName need not be a legal MacAnova variable name since the name of a data set on a file readable by matread() can include characters such as '.' that are not legal in MacAnova variable names. Thus, 'y <- getdata(jw11.5)' is legal and will attempt to retrieve data set "jw11.5" from DATAFILE.

Variable DATAFILE is normally pre-defined to contain "macanova.dat", the name of a sample file with several data sets. On some systems DATAFILE may be pre-defined to be some standard collection of data. Ordinarily, to make use of getdata you will want to assign to DATAFILE the name of a data file you wish to use by, for example, DATAFILE <- "disease.dat". Then getdata will retrieve data from file "disease.dat".

If the value of DATAFILE is a pure file name ("macanova.dat" but not, say, "data/macanova.dat"), if the file is not in the current default directory or folder it will be sought for in the directories or folders in CHARACTER vector DATAPATHS. See files.

If you regularly use a particular data file, say, "mydata", you might find it convenient to add the line

```
DATAFILE <- "mydata"
```

to your startup file. See customize.

A useful convention is to have the first dataset on the file have name 'info' and 0 lines (first header line 'info 0', with several comment lines starting with ')' listing the datasets available in the file. Then getdata(info) will print this information. See topic data files for an example.

2.102 gethistory

Keywords: general
Usage: gethistory(n) where n > 0 is an integer gethistory()

gethistory(n), where n is a positive integer, returns a CHARACTER vector containing up to n previous commands in the order they were executed.

gethistory(), with no argument, returns a CHARACTER vector containing all available previous commands.

If there are no previous commands available, both gethistory(n) and gethistory() return "".

If nHist is the current value of option 'history', at most nHist-1 commands are returned. See getoptions, setoptions()

See sethistory() for information on how to replace the current internal list of previous commands.

Probably the most important use of gethistory() is to preserve the history of what you have done between sessions. Supppose you execute the following command line:

```
Cmd> HISTORY <- gethistory(); save("chkpoint.sav")
```

Then, on the same or a future MacAnova run,

```
Cmd> restore("chkpoint.save")

Cmd> sethistory(HISTORY)
```

restores the internal list of previous commands to what it was at the time you saved the workspace. See save() and restore().

gethistory() is not implemented in the limited memory DOS version or in any version that does not allow keyboard or menu retrieval of previous commands.

2.103 getlabels

Keywords: general, variables
Usage: getlabels(x [,silent:T]) or getlabels(x, dims [,silent:T]), dims a vector of positive integers

getlabels(x) returns the coordinate labels assocated with variable x.

If x is a structure or ndims(x) = 1, the result is a CHARACTER vector; otherwise it is a structure with ndims(x) components, each of which as a CHARACTER vector of length dim(x)[i].

getlabels(x, dims), where dims is a vector of positive integers, returns the labels associated with coordinates dims[1], dims[2], ... of x. If dims is a scalar, the result is a CHARACTER vector; otherwise it is structure of CHARACTER vectors.

For both usages, if x has no labels, a warning message is printed and NULL is returned.

getlabels(x [, dims], silent:T) suppresses the warning message if there are no labels.

You can determine whether a variable has labels by

```
Cmd> if (!isnull(getlabels(x,silent:T))){...do something...}
```

See also labels, haslabels.

2.104 getmacros

Keywords: macros, files, input
Usage: `getmacros(name1 [,name2 ... ]), name1, name2 ...`
unquoted of macro names on one of files in CHARACTER vector MACROFILES

getmacros(Macro1,Macro2,...) retrieves macros Macro1, Macro2, ... from one of the files whose names are in pre-defined CHARACTER vector MACROFILES. The macro names must not be enclosed in quotes or be CHARACTER variables.

getmacros(Macro1,Macro2,...,quiet:T) retrieves the macros but suppresses printing the descriptive comments associated with them.

If there is more than one copy of any of the named macros, getmacros retrieves the first one found. The files of macros are searched in the order they are in MACROFILES.

MACROFILES has predefined value vector("macanova.mac", "tser.mac", "design.mac"). Each name in MACROFILES may also include a "path" with directory or folder information. You can easily add additional files to this list using pre-defined macro addmacrofile (see topic addmacrofile) or replace it entirely by, say, MACROFILES <- vector("mymacrofile1", "mymacrofile2"). If you often use a particular macro file or files you might find it convenient to have MACROFILES modified in your startup file. See topic customize.

Example: If MACROFILES has its default value
```
  Cmd> getmacros(covar, spectrum, confound3)
```
retrieves macros "covar" from file macanova.mac, "spectrum" from tser.mac and "confound3" from design.mac.

Note: Prior to 4/28/96, getmacros searched only the file specified in CHARACTER scalar MACROFILE. For backward compatibility, if vector MACROFILES does not exist, getmacros uses MACROFILE.

2.105 getoptions

Keywords: control

Usage: getoptions(option1:T [,option2:T ...]), option1, option2, ... option names. Legal option names are 'angles', 'batchecho', 'dumbplot', 'errors', 'format', 'fstats', 'height', 'inline', 'labelabove', 'labelstyle', 'maxwhile', 'missing', 'nsig', 'prompt', 'pvals', 'restoredel', 'seeds', 'update', 'warnings', 'wformat', and 'width' plus 'font' and 'fontsize' on Macintosh, 'scrollback' on Macintosh, Windows and Motif, and 'history' on Macintosh, Windows, Motif and some Unix and DOS/Windows versions. getoptions() or getoptions(all:T) gets all option values as a structure

getoptions(option1:T, option2:T, ...), where option1, option2, ... are option names, returns the values of the specified options. If more than one option is specified, the result is a structure with appropriately named components. Thus getoptions(format:T) returns the default format used in printing, getoptions(seeds:T) is equivalent to getseeds(quiet:T), and getoptions(height:T,width:T) returns a structure with components 'height' and 'width'.

getoptions() or getoptions(all:T) returns the values for all options.

getoptions(all:T, option1:F, option2:F,...) returns values for all options except those specified.

Legal option names are 'angles', 'batchecho', 'dumbplot', 'errors', 'format', 'fstats', 'height', 'inline', 'labelabove', 'labelstyle', 'maxwhile', 'missing', 'nsig', 'pvals', 'prompt', 'restoredel', 'seeds', 'update', 'warnings', 'wformat', and 'width'.

Option name 'lines' is also recognized as a synonym for 'height' for compatibility with earlier versions.

On versions (Macintosh, Windows, Motif, extended memory DOS and Unix) that allow you to recall previous commands, option 'history' is also legal.

On windowed versions (Macintosh, Windows, Motif), option 'scrollback' is also legal.

On computers such as a Macintosh, with changeable fonts, options 'font' and 'fontsize' are also legal.

See setoptions() for details on these options..

The value returned for 'format' or 'wformat' always has the type specifier ('f' or 'g') at the end ("12.5g"), even if it was set with a string starting with 'f' or 'g' ("g12.5").

See also restore(), save().

2.106 getseeds

Keywords: random numbers
Usage: getseeds([quiet:T])

getseeds() prints the current seeds and returns the current seeds in the random number generator used by runi(), rnorm(), rbin() and rpoi() as an "invisible" REAL vector of length 2.

getseeds(quiet:T) returns the seeds (as a regular REAL vector of length 2) but does not print them.

You can use getseeds() together with setseeds() to restart the random number generators from the same point more than once. If you retrieve the seeds by 'seeds <- getseeds(quiet:T)', you can later reset the random number generators to the same place by 'setseeds(seeds)'.

The seeds are saved by save() or asciisave() (unless you specify 'options:F') and then restored by restore().

See also setseeds(), runi(), rnorm(), rbin(), rpoi().

2.107 gettime

Keywords: general
Usage: gettime(), gettime(quiet:T), or gettime(keep:T [,quiet:F]) gettime(interval:T), gettime(interval:T,quiet:T), or gettime(interval:T,keep:T [,quiet:F])

gettime() prints the time in seconds since the start of the run.

gettime(interval:T) prints the time in seconds since the last time gettime() was used (since the start if this is the first usage).

gettime(quiet:T) and gettime(interval:T, quiet:T) do nothing but save the current time for the next time gettime(interval:T) is used.

gettime(keep:T [, quiet:F]) returns the time since start as a REAL scalar. It prints nothing unless quiet:F is an argument.

gettime(interval:T, keep:T [, quiet:F]) returns the time since last usage as a REAL scalar. It prints nothing unless quiet:F is an argument.

```
You can create a macro that will print the elapsed time of an action by
  Cmd> timeit <- macro("gettime(quiet:T);$0;gettime(interval:T)")
Then, for example,
  Cmd> timeit(x <- rnorm(1000);stuff <- describe(x))
will print time spent generating x and computing descriptive statistics.
See also macro(), macros.

On most computers, gettime() returns the actual time elapsed as might be
measured with a stop watch.  On a few computers, the time is the amount
of central processor time used.  This will generally be less, often much
less than the actual elapsed time.

Examples:
  Cmd> gettime()
  Time since start is 377.65 seconds.

  Cmd> gettime(quiet:T);mymacro(x,y);gettime(interval:T)
  Elapsed time is 3.6718 seconds

  Cmd> gettime(quiet:T);mymacro(x,y);gettime(interval:T,keep:T)
  (1)        3.699

  Cmd> gettime(quiet:T);mymacro(x,y);gettime(interval:T,keep:T,quiet:F)
  Elapsed time is 3.6523 seconds
  (1)       3.6523
```

2.108 glm

Keywords: glm, anova, categorical data, multivariate analysis, regression

Usage: `Type usage(command), where command is anova, fastanova, glmfit, ipf, logistic, manova, poisson, probit, regress, screen, robust`

```
The commands for analyzing linear and generalized linear models are as
follows:
   anova(), fastanova()              Analysis of Variance
   glmfit()                          Generalized linear model analysis
   ipf(), logistic()                 Logistic Regression
   manova(),                         Multivariate Analysis of Variance
   poisson()                         Log linear models
   probit()                          Probit analysis
   regress()                         Linear Regression
   robust()                          Robust Regression
   screen()                          Best subset linear regression

These are generally referred to as GLM commands in help topics.  See
their individual help entries for details.  Type help(key:"glm") for a
list of help entries related to analyzing linear and generalized linear
models.
```

In addition, wtanova(), wtmanova() and wtregress() do weighted ANOVA, MANOVA and regression. Since the same computations are done when weights are specified using keyword 'weights' or 'wts' (see below), these are not further mentioned here.

Function glmfit() is a general function that can, with appropriate keyword arguments, be used instead of anova(), logistic(), poisson(), and probit(). In the future, additional options will allow analyses not possible at present.

All GLM commands have certain elements in common:

Their first argument specifies a model as a quoted string or CHARACTER variable. Examples are regress("y=x1+x2+x3") and anova("x=a + a.b"). If the model is absent (for example, anova() or logistic(,n)) the most recent GLM model is assumed or the model in CHARACTER variable STRMODEL is used. Type help(models) for information on how to specify a model.

When there are MISSING values in any of the variables in a GLM model, any case with any MISSING values is omitted entirely. The maximum level of any factor is taken to be the maximum level on any of the complete data cases.

All but screen() compute certain side-effect variables. The most important are the following (not all may be produced by every command).

STRMODEL, a CHARACTER scalar containing the model used.

TERMNAMES, a CHARACTER vector containing the names of the terms in the model.

DEPVNAME, a CHARACTER scalar containing the name of the response variable in the model.

SS, a REAL vector of sums of squares or deviances, one for each term in the model. For manova() this is an array of SSCP matrices, with the first subscript indexing the term. Except when marginal:T is an argument to anova(), manova() or robust(), these are computed sequentially and measure the importance of a term after fitting previous terms, and ignoring later terms.

DF, a REAL vector containing the degrees of freedom associated with each term in the model.

RESIDUALS, a REAL vector or matrix of residuals from the fitted model. For any case with MISSING values in the data, RESIDUALS is MISSING.

WTDRESIDUALS, a REAL vector or matrix of weighted residuals from the fitted model. For analyses using iteratively re-weighted least

squares sucu as logistic(), probit(), or poisson(), the weights are those used on the last iteration. For any case with MISSING values in the data, WTDRESIDUALS is MISSING. WTDRESIDUALS is not created by anova(), regress() or manova() unless weights are provided.

XTXINV (regress()), the inverse or generalized inverse of X'X or X'WX, where X is the n by k matrix of predictors, including the constant vector if it is in the model, and W is the diagonal matrix of weights, if any.

HII, the REAL vector of leverages, the diagonal elements of X(XTXINV)X' or W X(XTXINV)X', where W is the diagonal matrix of weights, if any.

COEF (regress() only), the model coefficients.

Besides creating side effect variables, most GLM commands save "private" information about the analysis. This is used by commands such as regpred(), contrast(), coefs() and secoefs(). It can be retrieved by command modelinfo(). This information is not preserved by save() and asciisave() unless keyword phrase 'all:T' is used. It is discarded when you assign a value to STRMODEL.

Here is a list of keyword phrases recognized by more than one GLM command:

Keyword phrases Commands recognizing

print:F All GLM commands
Directs that most of the output to the screen is suppressed, although side effect variables are created.

silent:T All GLM commands but screen()
Directs that all output is suppressed; only side effect variables are computed.

coefs:F All GLM commands but screen(), regress(), fastanova(), ipf(), robust();
Directs that no computation of coefficients or a generalized inverse to X'X (X'WX when there are weights) is done. Except in the case of balanced ANOVA, coefs() and secoefs() cannot be used to retrieve coefficients later. coefs:F is not legal with marginal:T.

fstats:T regress(), anova(), manova(), robust()
Directs that F-statistics and P values are computed and printed. The denominator is the mean square for the next following term whose name is of the form "ERROR1", "ERROR2", ... For manova(), statistics are given separately for each variable and printing of the SS/SP matrices is suppressed.

pvals:T All GLM commands except screen()
Directs that F or Chi-Squared P values are computed and printed for F-statistics, t-statistics, and deviances. pvals:F suppresses P values when they might otherwise be printed.

```
inc:T          poisson(), ipf(), logistic(), glmfit()
 Specifies that an incremental analysis of deviance table is to be
 computed and printed.  No longer legal on robust().

marginal:T     anova(), manova(), robust()
 Specifies that SS (or SS/SP matrices) are computed marginally.  When
 there are no empty cells, and sometimes when there are, the computed
 SS or SS/SP are equivalent to SAS Type III quantities.  marginal:T
 is not legal with coefs:F.

maxit:n        fastanova(), poisson(), ipf(), logistic(), robust(),
               glmfit()
 Specifies the maximum number of iterations allowed in fitting

eps:smallVal   fastanova(), poisson(), ipf(), logistic(), robust(),
               glmfit()
 Specifies the a threshhold in relative change of objective function
 for determining when convergence has been reached

wts:vec        anova(), manova(), regress()
weights:vec
 Specifies a REAL vector to be used as weights.  'wts' and 'weights'
 are equivalent.

offsets:vec    poisson(), ipf(), logistic(), probit(), glmfit(),
               robust()
 Specifies a REAL vector to be used as offset vector in the linear
 scale.  Thus for poisson() and ipf(), vec should be in log units;
 for logistic(), vec should be in units of log(p/(1-p)) and for
 probit() vec should be in units of invnor(p).
```

If neither fstats:T nor fstats:F is an argument, for anova() and fastanova() (but not manova()), the printing of F-statistics is controlled by option 'fstats'. See setoptions().

If neither pvals:T nor pvals:F is an argument, for all GLM commands except manova(), robust() and screen(), the printing of P values is controlled by option 'pvals', except that if options 'pvals' has value False, P values will be printed if F-statistics are.

Sums of squares or deviances are normally computed sequentially. For anova(), manova(), and robust() these are SAS Type I quantities. Thus in the unbalanced case, several analyses may be necessary to compute all the sums of squares or deviances needed.

Keyword phrase, 'marginal:T', when it can be used, causes SS or SS/SP to be computed differently. When there are no empty cells in the design and no aliased variates, and sometimes when there are, the SS or SS/SP computed are SAS Type III quantities. In every case, they are numerator SS or SS/SP for a test that all the coeffients of non-aliased X-variables in a term are 0, where aliasing is determined by the original order of the sequential fit. If there is aliasing, the

quantities computed may depend on the order in which the terms are fit.

2.109 glmfit

Keywords: glm, anova, regression, categorical data

Usage: glmfit([Model] [,dist:distName,link:linkName, n:denom, incr:T, print:F or silent:T, maxiter:m, epsilon:eps, coefs:F, offsets:OffVec, scale:sigma]), distName and linkName CHARACTER scalars, denom > 0 REAL scalar or vector, integer m > 0, REAL eps > 0, REAL vector OffVec

glmfit(Model,dist:DistName ,link:LinkName,...) does a generalized linear model analysis with assumed response distribution DistName and link function LinkName, somewhat in the manner of program GLIM. The response variable y must be a vector (isvector(y) is True).

See topic 'models' for information on and examples of quoted string or CHARACTER scalar Model.

Current legal values for DistName are "binomial", "poisson", and "normal" (or "gaussian"). If DistName is "binomial" or "poisson", you must have y[i] >= 0.

Current legal values for LinkName are "logit", "probit", "log", and "identity".

If dist:DistName is omitted, the default DistName is "normal".

If link:LinkName is omitted the default LinkName depends on DistName -- "logit" for "binomial", "log" for "poisson", and "identity" for "normal".

Because of these defaults, glmfit(Model), with no distribution or link specified, is equivalent to anova(Model, unbalanced:T).

If DistName is "binomial" you must specify the number of trials using keyword 'n' as for logistic() or probit(). The value Denom for 'n' must either be a REAL scalar >= max(y) or a REAL vector of the same length as y with Denom[i] >= y[i].

glmfit() sets the side effect variables RESIDUALS, WTDRESIDUALS, SS, DF, HII, DEPVNAME, TERMNAMES, and STRMODEL.

An iterative algorithm is used to model link(E[y]) or link(E[y/Denom]) as a linear function of X-variables associated with the right hand side of Model. Normally a two line Analysis of Deviance table is printed. Line 1 is the difference 2*L(1) - 2*L(0), where L(0) is the log likelihood for a model with all coefficients 0 and L(1) is the maximized log likelihood for the model fit. Line 2 is 2*L(2) - 2*L(1) where L(2) is the maximized log likelihood under a model fitting one parameter for

every y[i]. Under certain conditions, the latter can be used to test the goodness of fit of the model using a chi-squared test.

glmfit(Model,dist:DistName,link:LinkName,inc:T,...) computes the full fitted model and all partial models -- only a constant term, the constant and the first term, and so on. It prints an Analysis of Deviance table, with one line for each term, representing a difference 2*L(i) - 2*L(i-1) where L(i) is the maximumized log likely for a model including terms 1 through i, plus the deviance of the complete model labeled as "ERROR1". Each line except the last can be used in a chi-squared test to test the significance of the term on the assumption that the true model includes no later terms.

The use of glmfit() provides an alternative method to specify a logistic or probit analysis of binomial responses, or a log linear analysis of Poisson responses.

Function	dist	link
logistic()	"binom"	"logit"
probit()	"binom"	"probit"
poisson()	"poisson"	"log"
anova()	"normal"	"identity"

In the future additional distributions such as "gamma" will be implemented, as well as additional links such as "sqrt", "recip", or "power". If you specify an unimplemented combination of LinkName and DistName, an informative error message is printed.

Other Keyword Phrases

Keyword phrase	Default	Meaning
maxiter:m	50	Positive integer m is the maximum number of iterations that will be allowed in fitting
epsilon:eps	1e-6	Small positive REAL specifying relative error in objective function (2*log likelihood) required to end iteration
print:F	T	Suppress all output except warning and error messages. Side effect variables are set.
silent:T	F	Suppress all output except error messages. Side effect variables are set.
coefs:F	T	Suppresses computation of quantities necessary for coefs() and secoefs() to compute estimated coefficients and their standard errors. Thus it effectively disables coefs() and secoefs() as well as some of the options available in modelinfo().
offsets:OffVec	none	Causes model to be fit to link to be 1*Offvec + Model, where OffVec is a REAL vector the same length as response y. OffVec must be in the

same units as the link function, say, logits, logs, or probits.

scale:sigma	1	sigma must be a positive REAL scalar or ? (MISSING). Its value will replace a default multiplier used by secoefs() and contrast() to compute standard errors. If the value is MISSING, sigma will be computed as sqrt(SS[m]/DF[m]), where m = length(SS). The default is 1 unless dist is "normal" when it is sqrt(SS[m]/DF[m]). In secoefs(), scale multiplies the square roots of the diagonal values of the inverse of X'WX, where X is the matrix of X-variables, and W is a diagonal matrix of weights computed using the converged fit.

See also logistic(), poisson(), probit(), glm.

2.110 glmpred

Keywords: glm, regression, anova, categorical data
Usage: glmpred(variates,factors [, estimate:F, seest:F, sepred:T, n:N]), variates and factors REAL vectors or matrices or NULL, N a positive scalar or REAL vector with positive elements.

glmpred(Variates, Factors) computes estimates of the expected value of the response variable y for specified values of any variates and levels of any factors in the latest GLM model. It returns a structure with REAL components (vectors, except after manova()) "estimate" and "SEest".

If there are no variates in the model, Variates should be NULL. Otherwise, Variates should be REAL. If there are Nvar variates in the model, Variates should either be a vector of length Nvar containing values for each of the variates, or a matrix with Nvar columns, with each row containing values for each variate.

If there are no factors in the model, Factors should be omitted or explicitly NULL. Otherwise, Factors should be REAL. If there are Nfac variates in the model, Factors should either be a vector of length Nfac containing levels for each of the factors, or a matrix with Nfac columns, with each row containing levels for each factor.

If either Variates or Factors contains data for only one case, it is used for all cases. Otherwise, you must have nrows(Variates) = nrows(Factors).

glmpred(Variates, Factors, sepred:T) adds component SEpred to the output structure containing a vector or matrix of prediction standard errors. This is only permissible after regress(), anova() or manova() and their

weighted alternatives.

glmpred(Variates, Factors, seest:F) suppresses the computation of standard errors.

glmpred(Variates, Factors, estimate:F) suppresses the computation of expected values. This option is legal only after anova(), manova(), regress() and their weighted alternatives.

You cannot use glmpred() after fastanova() or ipf() or when coefs:F was used on the preceding GLM command.

After GLM functions involving a Binomial response variable (logistic(), probit(), glmfit(...,dist:"binomial")), the values computed are the estimated probabilities p of "success" associated with each case (set of values). In this case, you can also use keyword phrase n:N, where N is a REAL variable, to specify the number of trials for each case. N can be a scalar or a vector whose length matches the number of cases. The resulting estimated values are N*phat, where phat are the estimated probabilities.

After GLM functions such as poisson(), logistic(), or probit(), where the expectation of the response is a non-linear function of a linear combination of the predictors, the standard error is computed from the expectation and standard error in the linear scale using the delta-method. When the response is binomial and you also use n:N, the standard errors are those of N*p.

Comment: Standard errors are computed on the assumption that all effects are fixed and not random. When this is not appropriate, the standard errors will usually indicate more precision than is warrented.

Examples:
  After regress(), glmpred(x,sepred:T) is equivalent to regpred().
  After anova("y=x+a+b"), x a variate, a and b factors,
     glmpred(x, hconcat(a,b)) computes fitted values and their standard
     errors for each case.
  glmpred(modelvars(variates:T), modelvars(factors:T)) computes fitted
  values and their standard errors for each case, regardless of the
     model.

2.111 glmtable

Keywords: glm, anova

Usage: glmtable([wtdmeans:T or x:vals, estimate:F, seest:F, sepred:T, n:N]) or glmtable(Term,[wtdmeans:T or x:vals, estimate:F, seest:F, sepred:T, n:N]) where vals is REAL vector and TERM is CHARACTER scalar of form "A.B. ...", where A, B are factors in current GLM model, N is a positive REAL scalar or vector or array of positive numbers.

glmtable() computes tables of fitted values (estimated cell expected values) and their standard errors based on the computations of the most recent GLM (generalized linear or linear model) command such as anova() or poisson(). It returns a structure with components "estimate" and "SEest" containing the tables, each of which has a dimension for each factor in the model, in the order the variables appear in the model. If there are variates in the model, the fitted values are computed with each variate set to its unweighted mean value and thus are what are sometimes called the covariate adjusted cell means.

glmtable(sepred:T) adds component SEpred to the output structure containing a table of prediction standard errors. This is only permissible after regress(), anova() or manova() and their weighted alternatives.

glmtable(seest:F) suppresses the computation of standard errors.

glmtable(estimate:F) suppresses the computation of expected values. This option is legal only after anova(), manova(), regress() and their weighted alternatives.

If only one array of values is computed, glmtable() returns that array, not a structure.

glmtable(wtdmeans:T [,...]) does the same except it adjusts cell fitted values to the weighted means of the variates. Use of wtdmeans:T when there are no variates or when the previous GLM command used unweighted OLS (anova() or manova() with no weights supplied) is an error. This option would be probably appropriate when the weights were proportional to sample sizes.

glmtable(x:Vals [,...]), where Vals is a REAL vector with as many elements as there are variates (non-factors) in the model, does the same computation, except it uses the elements of Vals instead of unweighted or weighted variate means. This option allows you to estimate cell means that are adjusted to any level of the covariates. Use of x:Vals is an error if there are no variates in the current GLM model.

glmtable(Term [,...]) returns an estimated marginal table for the factors specified by Term. Term is a quoted string or CHARACTER scalar of the form "Name1.Name2.Name3....", where Name1, Name2, ... are names of factors in the current GLM model. If there are k factor names in

Term, the value will be an array with k dimensions (vector if k = 1, matrix if k = 2), with the dimensions ordered in the same order as in the model, not the order in Term if that is different. You cannot use glmtable(term [,...]) after anova() with a balanced design unless Term includes all the factors in the model.

Example:

```
Cmd> glmtable("a.b", x:17) # same as glmtable("b.a",x:17)
```

glmtable(term:k [,...]) is equivalent to glmtable(TERMNAMES[k] [,...]), computing the marginal table matching term k in the model.

Example:

```
Cmd> glmtable(term:3, x:17).
```

You can use sepred:T when estimating a marginal table.

For GLM functions involving a binomial response variable (logistic(), probit(), glmfit() with dist:"binomial"), the values computed are the estimated probabilities p of "success" associated with each cell. In this case, you can also use keyword phrase n:N, where N is a REAL variable, to specify the number of trials for each cell. N can be a scalar, a vector whose length matches the size of the table, or a matrix or array whose dimensions match those of the table. The resulting table is a table of N*p.

Comment: When the marginal table for any term in the model contains empty cells, especially when a factor is nested in another with different numbers of levels, the estimated means may not be what you want.

After GLM functions such as poisson(), logistic(), or probit(), where the expectation of the response is a non-linear function of a linear combination of the predictors, the standard error is computed from the expectation and standard error in the linear scale using the delta-method. When the response is binomial and you also use n:N, the standard errors are those of N*p. You cannot use seest:T or sepred:T after fastanova() or ipf().

Comment: Standard errors are computed on the assumption that all effects are fixed and not random. When this is not appropriate, the standard errors will usually indicate more precision than is warrented. In particular, this is would be the case when one factor indexes replicates in a randomized block design and you use glmtable(Term,seest:T) to estimate treatment means, where Term contains all the factors except blocks.

After fitting a non-linear model by logistic(), probit(), poisson(), or glmfit(), when Term doesn't contain all the factors in the model, glmtable(Term) first computes the estimated marginal table in the linear scale (logit, probit, or log) and then transforms it back into the scale of the response. This means that the computed marginal table is not the marginal means of the fitted table. For example, if b is a factor with 3 levels, after logistic("y=a*b", n:40), sum(glmtable("a.b"))/3 is not the same as glmtable("b").

When keyword phrase coefs:F was an argument on the most recent GLM command, glmtable() is not available.

See also anova(), anovapred, glm.

2.112 grade

Keywords: ordering
Usage: grade(x [,down:T]), x REAL or CHARACTER or a structure with all REAL or all CHARACTER components.

grade(a) is similar to rank(a), producing a vector, matrix, or array of the same shape as a, but with the indices of the minimum, second smallest, ..., maximum values in each column in place of the ranks. Thus, if x is vector(3.2,1.4,5.6,2.1), grade(x) computes the vector(2,4,1,3) since x[2], x[4], x[1], x[3] are the values of x in increasing order. The basic property is that, if x is a vector, x[grade(x)] is the same as sort(x).

Argument a can be either REAL or CHARACTER. When a is CHARACTER, ordering is based on the ASCII collating sequence. See sort() for the complete ordering of characters.

grade(a,down:T) or simply grade(a,T) does the same except the underlying sort is in decreasing order so that grade(a,down:T)[1,] computes the case (row) numbers of the maximum of each column.

If there are k MISSING values of a column, the last k elements of the result are the indices of the MISSING values. For example, grade(vector(3,1,?,0,?)) computes vector(4,2,1,3,5). Consequently, if x is a vector, x[grade(x)] and x[grade(x,down:T)] compute the same vector as sort(x) and sort(x,down:T), even when there are MISSING values.

When there are ties, the values computed for the tied elements are unpredictable but still satisfy that x[grade(x),] is the same as sort(x).

Two uses for grade() are x[grade(x[,i]),] which reorders the rows of x so that column i is ordered, and grade(x)[1,] or grade(x,T)[1,] which compute the indices of the minimum and maximum of each column of x.

It is also acceptable for x to be a structure, whose non-structure components are all REAL or all CHARACTER. In that case, grade() returns a structure of the same form, each of whose non-structure components is the result of applying grade() to the corresponding component of x.

Examples:

```
  Cmd> grade(vector(27,22,25,26,22,21,?,24))
```

yields vector(6,2,5,8,3, 4,1,7), since the minimum (21) is in position 6, the 2nd and 3rd smallest (22) are in positions 2 and 5, . . ., the

```
largest (27) is in position 1, and the only MISSING value is in position
7.

  Cmd> grade(vector(27,22,25,26,22,21,?,24),down:T)
yields vector(1,4,3,8,2,5,6,7).

See also sort(), rank().
```

2.113 graphs

Keywords: plotting

Usage: Type usage(cmd) or help(cmd) where cmd is plot, chplot, lineplot, boxplot, addpoints, addlines, addchars, addstrings or showplot Type help(graph keys) for information on graphic keywords title, xlab, ylab, xmin, xmax, ymin, ymax, xaxis, yaxis, keep, show, dumb, height, width, file, new, landscape, ps, epsf, window, pause, lines, linetype, thickness, xticks, yticks, xticklen, yticklen, screendump Type help(graph files) for information on how to save a plot in a file using keywords 'file', 'new, 'ps', 'screendump', and 'epsf'. Type help(graph ticks) for information on modifying default tick mark placement and labeling using keywords 'xticks', 'yticks', 'xticklen' and 'yticklen'.

```
The basic plotting commands are as follows:
  plot(x,y)              Plot of columns of y against x
  chplot(x,y,ch)         Plot of columns of y against x using symbols ch
  lineplot(x,y)          Connected line plot of columns of y against x.
  boxplot(x)             Box plots of columns or components of matrix or
                         structure x.
  addpoints(x,y)         Add data to an existing graph
  addlines(x,y)          Add line connected data to an existing graph
  addchars(x,y,ch)       Add character labeled data to an existing graph.
  addstrings(x,y,chVec) Add labeling information in chVec at positions
                         specified by x and y.
  showplot()             Redisplay previously displayed graph

Arguments x and y can be replaced by a structure with at least two
components which are interpreted as x and y.  Any additional components
are ignored.

See topic graph keys for information on optional keyword phrases that
can be used to specify axis labels and a title or plotting limits.
Examples are xmin:0, xmax:10,ymin:-1, ymax:1,xlab:"X axis label",
ylab:"Y axis label", and title:"Title above graph".

See topic graph files for information on how to save a plot in a file
using keywords 'file', 'new, 'ps', 'screendump', and 'epsf'.
```

See topic graph ticks for information on modifying default tick mark placement and labeling using keywords 'xticks', 'yticks', 'xticklen' and 'yticklen'.

Low resolution ("dumb") plots

By default, all plotting commands produce high resolution graphs. Keyword phrase 'dumb:T' on any plotting command directs that the graph should be "dumb", that is a low resolution plot using characters that can be printed on any printer. If you prefer to have dumb plots as the default, type setoptions(dumbplot:T); 'dumb:F' will then be necessary to get high resolution plots. The default size of a dumb plot, including labels, is M lines by N - 1 character positions, where M and N are the values of options 'height' and 'width'. See setoptions(). You can override these defaults by graphics keywords 'height' and 'width' whose values define M and/or N. See topic graph keys.

Specification of data to be plotted

Commands plot(), chplot(), lineplot(), addpoints(), addlines(), addchars() and addstrings() all require arguments x and y which specify plotting positions. x is a REAL vector and y is a REAL vector or matrix. If y has more than 1 column, each column is plotted against x. For addstrings(), y must be a vector of the same length as x.

Alternatively, arguments x and y can be replaced by a structure with at least two REAL components. Thus, for example, plot(structure(x,y)) is equivalent to plot(x,y). If there are more than two components, the additional ones are ignored. Thus plot(structure(x,y, info:"Test Data")) is also equivalent to plot(x,y).

Except for addstrings(), x can be a scalar or a vector of length 2 which implicitly specifies ny equally spaced values where ny = nrows(y). When x = x0 is a scalar x0, the implied vector is vector(x0,x0+1,x0+2,...). If x is vector(x0,dx), the implied vector is vector(x0,x0+dx,x0+2*dx, ...). Otherwise, x and y must have the same number of rows.

For plot(), chplot(), and lineplot(), if x or y are specified as keyword phrases, as in plot(Time:tm,Level:y), the keywords are used as axis labels; however, keywords xlab or ylab (see below) will override the name:x or name:y forms. See topic graph keys.

Examples:

```
plot(1,y) is short for plot(run(nrows(y)),y)
plot(vector(1979,1/12),y) is short for
          plot(run(0,nrows(y)-1)/12+1979,y)
```

The second example might be used to plot monthly data starting January 1979 against time.

GRAPH variable LASTPLOT

As a "side effect", all plotting commands create a GRAPH variable with name LASTPLOT which encapulates all the information used to create the plot. The information is saved in a resolution independent form. You can assign LASTPLOT to another variable (for example, plot1 <- LASTPLOT)

or redisplay it, possibly with changed limits or labeling information, using showplot(). You can print it (as a "dumb" plot) by print(LASTPLOT) or write(LASTPLOT) or simply by typing LASTPLOT.

You can suppress the creation of LASTPLOT by keyword phrase 'keep:F'. This might be useful if you were running out of memory.

Adding information to a plot

Commands addpoints(), addchars(), addlines(), and addstrings() allow you to display GRAPH variables with added information. Alternatively and equivalently, you can use the keyword phrase 'add:T' as an argument to plot(), chplot(), or lineplot(). If the first argument is a GRAPH variable (for example, addlines(graph,x,y)), the plot combines the information in the GRAPH variable with the new information provided. Otherwise, the information in LASTPLOT is used. In no case is any GRAPH variable other than LASTPLOT changed.

If graph is a GRAPH variable, plot(graph,x,y), chplot(graph,x,y,c) and lineplot(graph,x,y) are equivalent to addpoints(graph,x,y), addchars(graph,x,y,c) and addlines(graph,x,y), respectively.

To force recomputation of xmin and xmax and/or ymin and ymax to include all data use keyword phrases xmin:0, xmax:0 and/or ymin:0, ymax:0.

To suppress immediate display of the graph, as when you are building a complex graph in stages, use 'show:F' as an argument to each plotting command. When you are done, simply type showplot(). It is an error to use both 'show:F' and 'keep:F'.

Examples:

```
  Cmd> plot(x,y,show:F); graphVar <- LASTPLOT
  Cmd> addpoints(3,4)  # or plot(3,4,add:T) or plot(LASTPLOT,3,4)
  Cmd> addpoints(graphVar,10,20,keep:F)#or plot(graphVar,10,20,keep:F)
  Cmd> showplot(xmin:0,xmax:0,ymin:0,ymax:0)
```

produces three plots. The first and third are plots of y vs x with the addition of a single point at x=3 and y=4, and the second is a plot of y vs x with the addition of a point at x = 10 and y = 20. MacAnova also recomputes the extremes displayed for the third plot. Because of 'keep:F' LASTPLOT is not updated after the second addpoints() command.

Graph Windows

On versions with windows (Macintosh, Windows and Motif), up to eight windows are available for use by plotting commands. In addition, on a Macintosh, "Panel of Graphs" windows, containing miniature replicas of up to four plots in their four corners are also created.

On any plotting command you can specify which window to draw in by keyword phrase 'window:n', where 1 <= n <= 8. Keyword phrase 'window:0' means you want to reuse the most recently drawn window. This is the default on any plotting command adding information to a previous plot. This is useful for displaying a sequence of related plots that differ in the value of a parameter. If 'window:n' is not used and a plot is not being added to, the first unused window is selected, or if all windows

are in use, a message is printed.

The graph in the currently displayed window, including the panel windows, may be saved to the Clipboard (not in Motif version) by selecting Copy from the Edit menu, saved to a file by selecting Save Graph As... on the File menu (Macintosh only) or printed by selecting Print... on the File menu.

Pausing between plots

Keyword phrase 'pause:T' on any plotting command results in MacAnova pausing after the plot is drawn. This is particularly useful when drawing repeated graphs in a 'for' or 'while' loop. Keyword phrase 'pause:F' suppresses any such pause. The default on windowing versions is pause:F while under DOS or Unix it's pause:T. On all machines, the pause can be terminated by hitting RETURN. On the Macintosh, any non-command key terminates the pause, and you can use the File and Edit menus to print the graph or to Copy the graph to the Clipboard. Together with the window keyword, this permits you to display an unlimited number of graphs successively in the same window, pausing after each one to examine it and possibly print it or copy it.

Plotting on Unix

Except for the Motif version, on UNIX machines, plotting commands produce Tektronix compatible plotting sequences. If running in a xterm pseudo VT100 window on a workstation, a pseudo Tektronix 4014 graphics window is opened and drawn to, switching back to the VT100 window when done. If running on Unix using a terminal emulator such as DOS Kermit or Macintosh NCSA Telnet 2.6 that supports Tektronix 4014 plotting, you may have to use putascii() before and/or after a plotting command to switch into and/or out of Tektronix mode. See vt, tek.

2.114 graph files

Keywords: plotting, files, output
Usage: Type help(graph files) for information on saving plots in files.

This topic summarizes those plotting options allowing you to save a plot in a file. See also topic graphs, graph keys, files.

When keyword phrase file:FileName is an argument to a plotting command, no plot is displayed. Instead, PostScript commands for the new plot are written to file FileName, which must be a CHARACTER variable or string. If keyword phrase landscape:T is also present, the plot will be rotated to fill an 8.5" by 11" page. If option 'dumbplot' has been set to True (see setoptions()), you will need to put dumb:F to get a PostScript file.

With file:fileName, if keyword phrase dumb:T or ps:F is also an argument, a low resolution "dumb" plot is written rather than PostScript. This consists only of characters that can be printed on any

printer. On some systems, if ps:F appears and dumb:T does not, the plot is written in a binary form appropriate to the platform (Tektronix commands on Unix; PICT format on a Macintosh).

If the keyword phrase 'new:T' is an argument, the information in the file is destroyed before the plotting commands are written. Otherwise, information is added at the end of the file. It formerly was the case tha a PostScript "prolog" was written only with new:T but now a prolog is always written before PostScript commands.

On a Macintosh you can write the plot as an encapsulated PostScript file by using 'epsf:T, file:fileName'. The file can be imported into some word-processors and graphics editing programs. epsf:T is illegal with new:F or ps:F.

On a Macintosh and in the DOS extended memory version you can write a binary version of the plot to a file using keyword phrase screendump:fileName. The format used is one appropriate to the computer. The Macintosh version writes a so called PICT file and the extended memory DOS version writes a PCX file. 'screendump:fileName' is not legal together with dumb:T or file:FileName.

On a Macintosh, item Save Graph As on the File menu writes the graph window currently being displayed as a PICT file.

See topics data files and macro files for information on files containing data sets and macros.

2.115 graph keys

Keywords: plotting

Usage: Type help(graph keys) for a list of optional keyword phrase arguments to plotting commands. Keywords described are dumb, epsf, file, height, keep, landscape, lines, linetype, new, pause, ps, screendump, show, thickness, title, width, window, xaxis, xlab, xmax, xmin, xticklen, xticks, yaxis, ylab, ymax, ymin, yticklen, yticks Type help(graphs) for general information on makeing graphs.

Here is a summary of the optional keyword phrase arguments recognized by most plotting commands. See also topic graphs, graph files, keywords.

Keyword Phrase	Description
title:"Plot title of your choice"	(up to 75 characters)
xlab:"X-axis label"	(up to 50 characters)
ylab:"Y-axis label"	(up to 20 characters)
xmin:xMinVal	Minimum value for x-axis.
xmax:xMaxVal	Maximum value for x-axis.
ymin:yMinVal	Minimum value for y-axis.

```
ymax:yMaxVal                    Maximum value for y-axis.
xaxis:F                         Do not draw x axis (line y = 0).
yaxis:F                         Do not draw y axis (line x = 0).
xticks:RealVec                  Locations for x- or y-axis tick
xticks:RealVec                  marks and labels. xticks:? and
                                yticks:? mean compute from data.
                                xticks:NULL and yticks:NULL mean
                                no tick marks or labels.
xticklen:length                 Length of x- or y-axis ticks, where
yticklen:length                 length >= -1.  length < 0 means
                                outside frame; length > 2 means
                                full gridline.
lines:T                         On plot() and chplot(), connect
                                points with lines.
linetype:n                      On lineplot() and on plot() and
                                chplot() with lines:T sets
                                the line type to n, default is 1.
                                n must be integer 1 <= n < 100.
thickness:W                     On lineplot() and on plot() and
                                chplot() with lines:T sets
                                the line thickness to W times
                                normal thickness, default is
                                1. W must be between .1 and 10.
                                Has no effect when not feasible
                                as with dumb:T.
show:F                          Do not display plot, only save
                                as LASTPLOT (see below)
keep:F                          Do not save plot as LASTPLOT;
                                only display.
dumb:T                          Make low resolution plot using
                                printable characters only,
                                suitable for printing on a line
                                printer.  Default is taken from
                                option 'dumbplot'; F means high
                                resolution.
height:n                        Number of lines to be used in
                                "dumb" plot; n >= 12 is
                                required; ignored without
                                dumb:T.
width:n                         Width in character positions to
                                be used in "dumb" plot; n >= 30
                                is required; ignored without
                                dumb:T.
window:n                        Draw plot in window n (1 <= n <=
                                8).  (versions with windows
                                only) If n is 0, use window most
                                recently used.
pause:T (versions with windows) Forces (T) or suppresses (F) a
                                pause
pause:F (other versions)        after the graph is drawn.
file:FileName                   Write PostScript to file
                                FileName.
new:T                           Clear file FileName before
```

	writing
landscape:T	PostScript plot will be rotated so as to fill 8.5" by 11" page.
ps:F	Suppresses PostScript when writing a plot to a file. On Unix Tektronix plotting commands are written. On a Macintosh, a PICT file is written. On other systems, a 'dumb' plot is written.
epsf:T (Macintosh only)	Produces an encapsulated PostScript file. Legal only with file:fileName.
screendump:FileName (Macintosh and DOS extended memory versions only)	Save a copy of screen in file FileName in a form other applications may be able to import.

For xmin, xmax, ymin and ymax, a value of MISSING (for example, xmax:?) results in computation of the value using all data being plotted, including data previously encapsulated in a GRAPH variable.

2.116 graph ticks

Keywords: plotting
Usage: Type help(graph ticks) for information on how to modify default tick marks in graphs. Keywords are xticks, yticks, xticklen, yticklen.

By default, plotting commands place tick marks whose length is about the width of a character at "neat" positions. All tick marks are labeled. See topic graphs for general information on plotting commands, topic graph keys for information about keyword phrases.

You can control the tick mark positions by keywords xticks and yticks whose values must be REAL vectors or NULL. xticks:NULL and yticks:NULL suppress ticks and their labels entirely. xticks:? or yticks:? cause the default tick mark positions to be used.

You can control the lengths of the tick marks by keywords xticklen and yticklen whose values should be REAL scalars >= -1. Values < 0 give ticks outside the frame and 0 values suppress drawing ticks but not their labels. Values > 2 cause full gridlines from one side of the plot to the other to be drawn.

Examples:

```
  Cmd> plot(x,y,xticks:vector(1,2,4),yticks:NULL,xticklen:1.5)
```

gives x-axis ticks 1.5 times normal at x = 1, 2 and 4 and suppresses all y-axis ticks and their labels.

```
  Cmd> plot(x,y,xticklen:3,yticklen:-.5)
```

draws full gridlines perpendicular to the x-axis and half length ticks along the outside of left edge of the frame.

```
  Cmd> showplot(xticks:?,yticks:?)
```

sets the default tick positions, without altering their length.

Currently, save() and asciisave() do not save tick information with the result that a restored GRAPH variable may not be displayed exactly the same as the one that was saved. This is a known bug that will be fixed in a future release.

2.117 halfnorm

Keywords: transformations, descriptive statistics, ordering

Usage: halfnorm(x [,ties:"ignore" or "average" or "minimum"]), x REAL or a structure with REAL components.

halfnorm(x) computes the vector of approximate half normal scores for the data in the REAL vector x. This probably makes sense only when the elements of x are all non-negative, although that is not required.

The most important use of halfnorm() is probably plot(halfnorm(ss), sqrt(ss)), where ss is a vector of 1 degree of freedom sums of squares. This produces a half normal plot of sqrt(ss).

What is computed is equivalent to

```
   invnor(.5 + .5*(rank(abs(x),ties:"ignore") - .375)/(n + .25))
```

where n is the number of non-MISSING values. The value corresponding to a missing value is MISSING.

halfnorm(x,ties:Method), where Method is "ignore", "average", or "minimum" (or "i", "a", "m") computes invnor(.5 + .5*(rank(abs(x),ties:Method) - .375)/(n + .25)) See rank() for a detailed discussion of the three methods. It is hard to think of a situation when you would want to use "minimum" with halfnorm().

If x is a matrix, the result is a matrix each of whose columns contains the half normal scores for the corresponding column of x.

If x is an array, halfnorm(x) is an array of the same size and shape with all the elements with fixed values of subscripts 2, 3, ... defining a "column" whose half normal scores are computed. An array with dimension > 2 is always treated as an array and not as a matrix, even if there are at most two dimensions greater than 1.

It is also acceptable for x to be a structure, whose non-structure components are all REAL. In that case, halfnorm(x) returns a structure of the same form, each of whose non-structure components is the result of applying halfnorm() to the corresponding component of x.

See also rankits().

2.118 haslabels

Keywords: general, variables
Usage: haslabels(x), x a variable that is not NULL.

haslabels(x) is True if and only if variable x has coordinate labels.

Example;

```
Cmd> x <- yourMacro(y)
Cmd> if (haslabels(y)){
       x <- matrix(x,labels:structure("",getlabels(y,2)))}
```

See topics labels, getlabels().

2.119 hconcat

Keywords: combining variables, variables, null variables
Usage: hconcat(a,b,c,...) where a, b, c, ... matrices with same number of rows.

hconcat(a,b,c,...) combines matrices a, b, c ... side to side by concatenating their rows.

All arguments must be of the same type, REAL, LOGICAL, or CHARACTER, and have the same number of rows m. The result is a matrix of that type with m rows and na+nb+nc+... columns, where na, nb, nc, ... are the number of columns of a, b, c, ...

An argument that is a vector of length m is considered to be a m by 1 matrix. In particular, if a is a vector of length m, hconcat(a) is a m by 1 matrix.

An argument that is an array with only two dimensions not equal to 1 is considered to be a matrix (see matrices). Thus

```
Cmd> hconcat(array(run(6),1,2,3),array(2*run(8),2,1,4))
```

is equivalent to

```
Cmd> hconcat(matrix(run(6),2),matrix(2*run(8),2))
```

If a is a vector of length n, hconcat(a) is a matrix with n rows and 1 column.

Any argument of type NULL is ignored. If all arguments are NULL, so is the result.

See also vconcat(), vector(), matrices, vectors.

2.120 hconj

Keywords: time series, complex arithmetic
Usage: hconj(hx), hx a REAL matrix representing complex data with Hermitian symmetry

hconj(hx) returns the complex conjugate (in packed Hermitian form) of the columns of the vector or matrix hx, considered as complex series with Hermitian symmetry in packed Hermitian form.

See also cconj(), hreal(), himag(), creal(), cimag().

See topic 'complex' for discussion of complex matrices in MacAnova.

2.121 help

Keywords: general
Usage: help([Topic1,Topic2,...] [,file:FileName or orig:T or alt:T] [,scrollback:T])
help(Pattern) where Pattern has form "part*", "*part", or "*part*".
help(key:KeyNames), where KeyNames is a CHARACTER vector or "?"
help(news), help(news:yymmdd1) or help(news:vector(yymmdd1,yymmdd2)), where yymmdd1 and yymmdd2 are integers like 970103 or 19970103.

help() with no argument will print a short message giving some of the help() options.

help("*") lists all help topics.

help(Topic) prints information about the named topic. If the topic name is longer than 12 characters or is a control word ('if', 'while', 'for', 'else', 'elseif', 'break', 'breakall'), it must be quoted.

Examples:

```
Cmd> help(anova); help(macros); help("break");help("transformations")
Cmd> # help(break); help(transformations) do not work (no quotes)
```

See usage() for how to get just usage information, not full help which can be quite lengthy

See macrousage() for a way to get usage information on currently defined macros.

On a windowed version (Macintosh, Windows or Motif), after help(Topic, scrollback:T), the output from help() is automatically scrolled back to its start. You can make this the default behavior by

```
setoptions(scrollback:T). See setoptions().

On the Macintosh, Help on the Apple menu (Command+H) is equivalent to
'help()'. If you select a command name or other topic name in the
command window with the mouse, menu item Help gives you help on that
topic. In the Windows and Motif versions, Help on the Help menu is like
typing 'help()'.

help(key:"foo") lists all topic names associated with key "foo".
Examples of keys are "Residuals", "Missing Values", "ANOVA", and
"Variables". In matching keys, case (upper or lower) is ignored.
Moreover, you need type only enough letters to identify a key uniquely.
Thus help(key:"resid") gives the same output as help(key:"Residuals").

help(key:"?") lists all recognized keys.

You can also use one of keywords 'file', 'orig', or 'alt' (see below)
with 'key'.

help("part*"), help("*part"), and help("*part*") prints a list of
available help topics that start with "part", end with "part", or
contain "part", respectively. If only one name matches, full help will
be given on that topic.

Examples:
  Cmd> help("*anova") #lists anova,fastanova, manova, wtanova, wtmanova
  Cmd> help("*plot") #lists names ending in "plot" like chplot, boxplot
  Cmd> help("*tat*") #lists cellstats, rotate, rotation

help(Topic1,Topic2,... [,scrollback:T]) prints information about each of
the topics specified by the arguments. If an argument is a quoted
string starting and/or ending with "*", it is used as a pattern as
above, but if there is more than one topic, only the first topic in the
help file whose name matches is printed. The same rules concerning
quotes apply as when there is only one topic.

For example, help(help,"break") will produce this text and information
about syntax element 'break'.

help(news [,scrollback:T]) lists in reverse chronological order news
items about MacAnova starting with the most recent entry back for three
months.

help(news:vector(Date1,Date2) [,scrollback:T]), where Date1 and Date2
are numbers of the form yymmdd or yyyymmdd, lists in reverse
chronological order news items about MacAnova development dated between
Date1 and Date2. For example, help(news:vector(971201,971230)) and
help(news:vector(19971201,19971230)) list all news items dated in
December, 1995.

help(news:Date) lists all news items on or after Date. For example,
help(news:970901) and help(news:19970901) list all news items on or
after September 1, 1997.
```

help(news:0) lists all available news items. This will produce a thousands of lines of output and is not recommended.

Older news items
The help information is kept in a file in a particular format. The default file name is macanova.hlp. A description of the format is near the start of the file.

help(file:FileName) where FileName is a quoted string or CHARACTER variable specifying the name of a file, will make that file an alternate help file and switch over to using it. In versions with windows (Macintosh, Windows, Motif), if FileName is "", a dialog box appears to select the file. One or more topic names can follow file:FileName or file:FileName can be the last argument. The alternate help file remains active until another is specified, or keyword phrase orig:T appears in a help() command. If the file is not in the correct format for a help file, the results are unpredictable but not pleasant.

help(orig:T) restores the help file to what it was at startup. This will either be the standard help file or the one specified (under Unix or DOS) by the -h option on the command line. See launching. One or more topic names can follow orig:T, or orig:T can be the last argument.

help(alt:T) restores the help file to the one most recently specified by file:fileName. It is an error if no alternative file was previously set. This allows you easily to use two help files, the default one and one alternate. You switch back and forth between them by help(orig:T) and help(alt:T). One or more topic names can follow alt:T, or alt:T can be the last argument.

'Help' is a synonym for 'help' and is used identically.

2.122 Help

Keywords:
Usage: Type usage(help)

Help() is used identically with help(). Type help(help).

2.123 hft

Keywords: time series, complex arithmetic
Usage: hft(hx [,divbyT:T]), hx a REAL matrix considered as complex in Hermitian form

hft(hx) where hx is a REAL vector or matrix, computes the real discrete Fourier transform of each column of hx, considered as a complex series

with Hermitian symmetry in packed Hermitian form. Any MISSING values in hx are treated as if they were 0.

hft(hx,divbyt:T) does the same, except the result is divided by the number of rows of hx.

hconj(rft(rx,divbyt:T)) is the inverse of rft() in the sense that hx and hconj(rft(hft(hx),divbyt:T)) are equal except for rounding error.

The largest prime factor of nrows(hx) must not exceed 29.

See topic 'complex' for discussion of complex matrices in MacAnova.

See also cft(), rft(), hconj().

2.124 himag

Keywords: time series, complex arithmetic
Usage: himag(hx), hx a REAL matrix representing complex data with Hermitian symmetry

himag(hx) computes the imaginary part of the packed Hermitian matrix hx. Thus himag(vector(1,2,3,4,5)) is vector(0,5,4,-4,-5) and himag(vector(1,2,3,4,5,6)) is vector(0,6,5,0,-5,-6).

See topic 'complex' for discussion of complex matrices in MacAnova.

See also hconj(), cconj(), hreal(), creal(), cimag().

2.125 hist

Keywords: plotting, descriptive statistics
Usage: hist(x [, nbars] [, graphics keyword phrases]), x a REAL vector, nbars >= 2 an integer hist(x, boundaries [, graphics keyword phrases]), boundaries a REAL vector with increasing elements

hist(x) draws a histogram of the data in REAL vector x using approximately log2(n) bars, where n is the length of x. The bar boundaries are equally spaced, but are not "neat" (1,1.5,2,2.5,... would be "neat", 2.71,3.82, 4.93, 6.04 are not "neat").

hist(x, nbars) draws a histogram with n equal width bars. The boundaries are not "neat".

hist(x, Boundaries), where Boundaries is a REAL vector, draws a histogram with the boundaries of the bars given by the elements of Boundaries. The number of bars is nbars = length(Boundaries) - 1.

Boundaries must consist of at least two elements, in increasing order (that is, Boundaries[i] < Boundaries[i+1]), and such that Boundaries[1] < min(x) and Boundaries[nbars+1] >= max(x). A value x is included in the i-th bar if Boundaries[i] < x[i] <= Boundaries[i+1].

In every case, the bar heights are in the so called "density scale" so that the total area of all the bars is 1.

If option 'dumbplot' has been set False (see setoptions()), the plot will be a low resolution plot unless 'dumb:F' is an argument.

All of the usual plotting related keywords, including dumb, xlab, ylab, and title, may be used with hist. See also graphs.

2.126 hpolar

Keywords: time series, complex arithmetic
Usage: hpolar(hx [,unwind:F or crit:val]), hx a REAL matrix representing complex data with Hermitian symmetry, val a REAL scalar, 0.5 < val <= 1

hpolar(hx) computes the polar form of the packed Hermitian matrix hx, storing it in pseudo packed Hermitian form, with the modulus as the real parts and the phase (argument) as imaginary part. Thus hreal(hpolar(hx)) returns a REAL matrix whose columns are the moduli (absolute values) of the complex series represented by the columns of hx and himag(hpolar(hx)) returns the phases. By default the latter are "unwound" so as to minimize discontinuities arising from wrap-around.

hpolar(hx,crit:val) changes the criterion controlling "unwinding". See unwind() for details.

hpolar(hx,unwind:F) suppresses the unwinding.

See topic 'complex' for discussion of complex matrices in MacAnova.

See also cpolar(), crect(), hrect().

2.127 hprdh

Keywords: time series, complex arithmetic
Usage: hprdh(hx1 [, hx2]), hx1 and hx2 REAL matrices representing complex data with Hermitian symmetry

hprdh(hx1, hx2) computes the element-wise complex product of elements in the columns of REAL matrices or vectors hx1 and hx2, considered as complex matrices in packed Hermitian form. The result is also a complex matrix in packed Hermitian form.

If hx1 or hx2 represents a single complex series (has 1 column), that series is multiplied by all the series in the other arguments. Thus for example, if hx1 is m by 1 and hx2 is m by 3, hprdh(hx1,hx2) is equivalent to hprdh(hconcat(hx1,hx1,hx1),hx2).

hprdh(hx) is equivalent to hprdh(hx,hx).

See topic 'complex' for discussion of complex matrices in MacAnova.

See also hprdhj(), cprdc(), cprdcj().

2.128 hprdhj

Keywords: time series, complex arithmetic
Usage: hprdhj(hx1 [, hx2]), hx1 and hx2 REAL matrices representing complex data with Hermitian symmetry

hprdhj(hx1, hx2) computes the element-wise complex product of elements in the columns of hx1 and hconj(hx2), considered as complex matrices in packed Hermitian form. The result is also a complex matrix in packed Hermitian form.

If hx1 or hx2 represents a single complex series (has 1 column), that series is multiplied by all the series in the other arguments. Thus for example, if hx1 is m by 1 and hx2 is m by 3, hprdhj(hx1,hx2) is equivalent to hprdhj(hconcat(hx1,hx1,hx1),hx2).

hprdhj(hx) is equivalent to hprdhj(hx,hx) and produces an packed Hermitian output matrix with the squared moduli of the elements of hx, considered as complex numbers, in the real part of the result, with zeros in the imaginary part.

See topic 'complex' for discussion of complex matrices in MacAnova.

See also hprdh(), hprdhj(), hconj().

2.129 hreal

Keywords: time series, complex arithmetic
Usage: hreal(hx), hx a REAL matrix representing complex data with Hermitian symmetry

hreal(hx) computes the real part of the packed Hermitian matrix hx. Thus hreal(vector(1,2,3,4,5)) returns vector(1,2,3,3,2) and hreal(vector(1,2,3,4,5,6)) returns vector(1,2,3 4,3,2).

See topic 'complex' for discussion of complex matrices in MacAnova.

See also hconj(), cconj(), himag(), creal(), creal().

2.130 hrect

Keywords: time series, complex arithmetic
Usage: hrect(hx), hx a REAL matrix representing complex data with Hermitian symmetry

hrect(hx) is the inverse operation to hpolar(). Thus hx is assumed to represent the polar form of a complex matrix in packed Hermitian form and the result contains that series in packed Hermitian form.

See topic 'complex' for discussion of complex matrices in MacAnova.

See also cpolar(), crect(), hpolar().

2.131 htoc

Keywords: time series, complex arithmetic
Usage: htoc(hx), hx a REAL matrix representing complex data with Hermitian symmetry

htoc(hx) returns the fully complex equivalent of the matrix hx, considering its columns as complex series with Hermitian symmetry. If hx is m by n, htoc(hx) is m by 2*n, the real and imaginary parts of column i of hx in columns 2*i-1 and 2*i of the result.

See topic 'complex' for discussion of complex matrices in MacAnova.

See also ctoh(), cconj(), hconj(), hreal(), himag(), creal(), cimag().

2.132 hypot

Keywords: transformations
Usage: hypot(x,y), x and y REAL of the same size and shape, or structures with matching REAL components

hypot(x,y) is mathematically equivalent to sqrt(x^2 + y^2) but is computationally more accurate. x and y must be REAL vectors, matrices, or arrays with the same dimensions.

hypot(x,y) is also defined when x and y are structures of the same shape. The result is a structure whose i-th component is hypot(xi,yi), where xi and yi are the i-th components of x and y.

hypot(x,y) can be used when both x and y are CHARACTER variables with matching dimensions. In that case the result is a CHARACTER variable describing the transformation of the arguments. For example, hypot(vector("X1","X2"),vector("Y1","Y2")) returns vector("hypot(X1,Y1)", "hypot(X2,Y2)"). This feature may be useful in creating new labels for a transformed variable.

See also atan(), transformations, structures, labels.

2.133 if

Keywords: syntax, control
Usage: `if (Logical1)command1;command2; ...[elseif (Logical2)...[else...]]`

if(Logical){Statement} allows conditional execution of Statement, an arbitrarily complex statement or compound statement. Statement will be executed if and only if Logical has the value True. Logical should be a LOGICAL variable or expression. A simple example would be

```
Cmd> if(min(x) > 0){logx <- log(x);;}
```

if(Logical){Statement1} else {Statement2} results in Statement1 being executed if Logical is True, and Statement2 being executed if Logical is False. An example would be

```
Cmd> if(min(x) > 0){logx <- log(x);;}else{
       error("Illegal non-positive values")}
```

if(Logical1){Statement1} elseif(Logical2){Statement2} else {Statement3} executes Statement1 if Logical1 is True, executes Statement2 if Logical1 is False and Logical2 is True, and executes Statement3 if both Logical1 and Logical2 are False. An example would be

```
Cmd> if(min(x) > 0){logx < -log(x);;}elseif(max(x) < 0){
          logx < -log(-x);;}else{
          error("Not all positive and not all negative")}
```

There can be additional elseif(Logical){Statement} constructs before the concluding 'else{...}', and the concluding 'else{...}' can be omitted. The first Statement for which the corresponding Logical is True is executed.

The value of any of these constructs starting with 'if' is the value of whichever '{Statement}' is actually executed. If no value is wanted, it is good practice to terminate each Statement with ';;' so the value will be null, as in the examples above. If all the Logicals are False and there is no concluding 'else{...}', there is no value. For example, if(2 > 3){ ...}elseif(5 < 4){...} has no value. See below for examples of how you can the fact that a conditional statement has a value.

Once a Logical has been found to be True, any subsequent Logicals are not evaluated and are not even checked for syntactical correctness.

```
Examples:
  Cmd> if(ismatrix(x)){xinv <- solve(x);;}else {error("not matrix");;}
  Cmd> a <- if(x < 3){1} else {2}# a <- 1 if x < 3 and a <- 2 otherwise
  Cmd> b <- if(x > 0){1} elseif(x < 0){-1} else{0} # b is 1, -1, or 0

Note: Each '{' following 'if(...)', 'elseif(...)' or 'else' must be on
the same line as the preceding 'if', 'elseif' or 'else'; 'elseif' and
'else' must be on the same line as the preceding '}'. As usual,
however, you can terminate the line with '\', and continue on the next
line as in the following:
  Cmd> if(x<0)\
         {y <- 1;;}\
       else\
         {y <- 2;;}
```

2.134 inforead

Keywords: input, files, character variables

Usage: `inforead(fileName,Name [,quiet:T, echo:T or F, silent:T, notfoundok:T]), fileName and Name CHARACTER scalars`
`inforead(string:charVal [,quiet:T, echo:T or F, silent:T, notfoundok:T]), charVal CHARACTER scalar or vector`

inforead(FileName,Name) searches file FileName for a macro or data set with name Name. If found, it reads the comments (lines starting with ')') following the header line and returns a CHARACTER variable containing these lines, with the leading ')' stripped off. FileName and Name must be quoted strings or CHARACTER variables. The actual contents of the data set or macro are ignored and there is no checking as to whether the header line is in correct format.

inforead(FileName) does the same for the first data set or macro on the file, assuming that line 1 is the header line.

In versions with windows (Macintosh, Windows, Motif), if FileName is the null string "", you will be able to select the file using a dialog box.

There are 3 keywords, 'quiet', 'silent' and 'notfoundok' which control what will be printed by inforead(). 'echo' is recognized but ignored.

Keyword phrase	Meaning
quiet:T	Header and descriptive comments will not be printed
quiet:F	All header and descriptive comments will be printed
silent:T	Only error messages will be printed; incompatible with quiet:F or echo:T
notfoundok:T	Failure to find the macro or data set is not considered an error so no error message is printed.

Without quiet:T or quiet:F, the header and comment lines not starting

with '))' preceding the macro or data set will be echoed to output.

When notfoundok:T is an argument and the data set or macro is not found, inforead() returns NULL as value. When used in a macro, this feature allows special action if data set or macro Name is not found.

Example:
  Cmd> epidemicinfo <- inforead("tseries.dat","epidemic",quiet:T)
creates a CHARACTER variable containing the information describing data set "epidemic" on file "tseries.dat". Without quiet:T, this information would also be echoed to output.

Commands save() and asciisave() would save epidemicinfo in a file along with the rest of your workspace. When you later use restore() to recover the workspace, print(epidemicinfo) would display without having to refer to the original data file.

inforead(file:FileName,...) is equivalent to inforead(FileName,...).

inforead(string:CharVar) where CharVec is a CHARACTER scalar or vector, does not read from a file. Instead, it "reads" CharVar as if each element were a line (or several lines if there are embedded end-of-line characters) read from a file. The first element or line of CharVar must be a header line for a data set or a macro. In particular, info <- inforead(string:CLIPBOARD) would read the header information of the first variable on a replica of a data file in the special variable CLIPBOARD. In the Macintosh, Windows and Motif versions this would be taken from the Clipboard. See topic clipboard.

If either keyword 'file' or 'string' is used, they can appear in any position in the argument list, as can setName which must be the only non-keyword argument. For example, inforead(quiet:T,"mymacro", file:"myfile.dat") is equivalent to inforead("myfile.dat","mymacro", quiet:T).

See topics data files and macro files for information on the format of files readable by inforead().

See also matread(), macroread(), save(), asciisave().

2.135 invbeta

Keywords: probabilities, random numbers
Usage: invbeta(P, alpha, beta), P, alpha and beta REAL, elements of P between 0 and 1, those of alpha and beta > 0

invbeta(p,a,b) computes the pth quantile (100*p percent point) of the beta distribution with parameters a and b.

The elements of p must be between 0 and 1 and the elements of a and b

must be positive REAL numbers.

If p, a, and b are not all scalars (single numbers), all non-scalar arguments must have must have the same size and shape and any scalar arguments are used to compute all the elements of the result.

invbeta(runi(n),a,b) will generate a random sample of size n from a beta distribution .

invbeta() is the inverse of cumbeta().

See also cumbeta(), runi().

2.136 invchi

Keywords: probabilities, random numbers, confidence intervals
Usage: invchi(P, df [,noncen, epsilon:eps]), P, df and noncen REAL, elements of P between 0 and 1, elements df > 0, elements of noncen >= 0, eps > 0 small.

invchi(p,df) computes the pth quantile (100*p percent point, critical value) of the chi squared distribution with df degrees of freedom.

invchi(p,df,Noncen [, epsilon:eps]) computes the pth quantile of non-central chi-squared with non-centrality parameter Noncen. The accuracy of the inverse is controled by eps which has default value 1e-10.

The elements of p must be between 0 and 1 and the elements of df must be positive but need not be integers. If present, the elements of Noncen must be non-negative.

Any of p, df or Noncen that are not scalars (single numbers) must be the same size and shape. Any argument that is a scalar is used to compute all elements of the result.

If alpha is small, invchi(1-alpha,df) is the critical value for a chi-squared test of significance level alpha. If Ssq is the sample variance from a normally distributed random sample of size n, then (n-1)*Ssq/invchi(vector(1-alpha/2, alpha/2),n-1) is a 1-alpha confidence interval for the population variance.

invchi() is the inverse of cumchi().

invchi(runi(n),df [,Noncen]) will generate a random sample of size n from a possibly non-central chi-squared distribution .

See also cumchi(), runi().

2.137 invdunnett

Keywords: probabilities, comparisons

Usage: invdunnett(P, ngroup, errorDf [,groupSizes][,onesided:T][,epsilon:ep
P REAL with elements between 0 and 1, elements of
ngroup integers >= 2, elements of errorDf >= 1,
elements of groupsizes >= 0, eps > 0, default = .00001.

invdunnett(P, K, Df) computes Pth quantile (probability point, critical value) of Tmax, where Tmax is the maximum of abs(t21), abs(t31), ... abs(tK1), where t21, t31, ..., tK1 are K-1 t-statistics of the form tI1 = (xbarI-xbar1)/ stderr(xbarI-xbar1), I = 2,...,K. xbar1, xbar2, ..., xbarK are the means of independent normal random samples of the same size with identical population means and variances, and the standard errors are computed using an independent estimate of error variance with Df degrees of freedom. When K = 2 the value is the same as invstu((1+P)/2, Df). See invstu().

invdunnett(P, K, Df, onesided:T) computes the quantiles for Tmax, where Tmax is now the maximum of t21, t31, ..., tK1, not of their absolute values. When K = 2 the value is the sames as invstu(P,Df).

See below for computing quantiles when the sample sizes differ.

P, K and Df must be REAL. The elements of P must be between 0 and 1. The elements of K must be integers >= 2, and the elements of Df must be >= 1, not necessarily integers.

Any of the arguments P, K or Df that are not scalars must all be vectors, matrices or arrays of the same size and shape; the value has the same size and shape.

invdunnett(P, K, Df [, onesided:T], epsilon:eps), where eps is a small positive number (default .00001) which controls the accuracy to which the quantile is computed. Specifically the logit of the probability corresponding to the computed quantile should be no farther than eps from the true logit of P (logit(P) = log(P) - log(1-P). Since invdunnett() uses the algorithm underlying cumdunnett() configured so as to compute probabilities to within .00001, eps should not be smaller than the default.

invdunnett() is primarily used to compute critical values for a multiple comparisons procedure due to C. W. Dunnett wherein a control group (group 1) is compared to K-1 other treatment groups using K-1 t-tests. For a completely randomized design with k treatment groups of size n, the P value is computed as 1-cumdunnett(maxt, k, k*n - k [,onesided:T]). See cumdunnett() for computing P values for the Dunnett test.

Caution: invdunnett() is very computation intensive. If you do not have a fairly fast computer, it may be unacceptably slow. On one Macintosh 68000 computer with no math coprocessor, a single value took over 2300 seconds to compute.

```
Example:
  Cmd> invdunnett(1 - .05, 5, 5*8 - 5)
```
computes the two-sided critical value for the Dunnett test with significance level alpha = 0.05 for a completely randomized design with 5 groups, all with sample size 8.

invdunnett(x, K, Df, groupSizes [,onsided:T]) computes quantiles for Tmax, with REAL argument groupSizes specifying the sample sizes. In the simplest usage, groupSizes is a vector (ndims(groupSizes) = 1), with elements >= 0. If groupSizes is a matrix or array (ndims(groupSizes) > 1), it is treated as if it were a vector, matrix or array, with one less dimension, each of whose elements is a vector with length = last dimension of groupSizes. The first ndims(groupSizes) - 1 dimensions of groupSizes must match the dimensions of any of x, K, or DF which is not a scalar. In particular, a m by 1 matrix, which is treated as a vector of length m by most MacAnova functions, is interpreted by invdunnett() as a set of m vectors of length 1.

In computing an element of the result based on a vector of group sizes (either all of groupSizes when it is a vector, or a row or "slice" of groupSizes when ndims(groupSizes) > 1), invdunnett() uses up to k of the non-zero leading values in the vector, where k is the corresponding element of K. If there are fewer than k non-zero values, the last one is replicated as many times as needed. It is an error to have a zero value followed by a nonzero value or to have all values zero.

If there is only 1 non-zero value in a row or "slice" of groupSize, the replication of this element means the group sizes are assumed to be equal. In particular, this is the interpretation when groupSizes is a scalar or a m by 1 matrix.

Examples:
```
  Cmd> invdunnett(1 - .05, 4, 12 - 4, vector(6,2,2,2))
```
computes the 5% critical value for a completely randomized design with 4 groups and sample sizes 6, 2, 2 and 2.

```
  Cmd> invdunnett(1 - .05, vector(3,4), vector(12 - 3, 12 - 4),\
        matrix(vector(6,3,3,0, 6,2,2,2),4)')
```
computes 5% critical values for two completely randomized designs, one with 3 groups and sample sizes 6, 3, and 3, the other with 4 groups with sample sizes 6, 2, 2, and 2. Because trailing values in the rows of groupSizes are replicated, matrix(vector(6,3, 6,2),2)' would be an equivalent way to specify the group sizes.

```
  Cmd> invdunnett(1 - .01, 4, 12 - 4, 3)
```
computes the same result as invdunnett(1 - .01, 4, 12 - 4), because groupSizes is a scalar.

Only the ratios of non-zero elements of groupSizes are relevant. Thus for 5 groups (K=5), the following groupSizes are equivalent: vector(5,4,3,3,3), vector(5,4,3), vector(5,4,3,0,0), vector(10,8,6). vector(5,4,3,0,3) and vector(5,4,3,0,0,1) would be errors because a non-zero value follows a zero.

Caution: invdunnett() is quite computation intensive. On a slow computer you may have to wait several minutes for a result. Until you know how long it will take on your computer, don't compute more than one value at a time. Using a somewhat larger value for epsilon, for example, epsilon:.0001 or epsilon:.0005, may speed up the calculation at the cost of some loss of accuracy.

See also cumdunnett(), invstudrng().

2.138 invF

Keywords: probabilities, random numbers, confidence intervals
Usage: invF(P, df1, df2), P, df1 and df2 REAL, elements of P between 0 and 1, those of df1 and df2 > 0

invF(p,df1,df2) computes the pth quantile (probability point, critical value) of the F distribution with df1 and df2 degrees of freedom.

The elements of p must be between 0 and 1 and the elements of df1 and df2 must be positive REAL numbers (not necessarily integers).

If p, df1, and df2 are not all scalars (single numbers), all non-scalar arguments must have the same size and shape. Any scalar arguments are used to compute all elements of the result.

If S1sq and S2sq are sample variances from independent normal random samples of sizes n1 and n2,, you can compute a 1 - alpha confidence interval for the variance Var1/Var2 as
(S1sq/S2sq)/invF(vector(1-alpha/2,alpha/2,n1-1,n2-1)).

invF() is the inverse of cumF().

invF(runi(n),df1,df2) will generate a random sample of size n from a F distribution .

See also cumF(), runi().

2.139 invgamma

Keywords: probabilities, random numbers
Usage: invgamma(P, alpha), P and alpha REAL, elements of P between 0 and 1, those of alpha > 0

invgamma(p,alpha) computes the pth quantile (100*p percent point) of the gamma distribution with shape parameter alpha. Its principal use is to compute critical values for test statistics with a gamma distribution.

The elements of p must be between 0 and 1 and the elements of alpha must be positive but need not be integers.

If neither p nor alpha is a scalar (single number), they must be the same size and shape. If just one argument is a scalar, it is used to compute all the elements of the result.

invgamma() is the inverse of cumgamma().

2*invgamma(p,df/2) is equivalent to invchi(p,df).

mu*invgamma(runi(n),alpha)/alpha will generate a random sample of size n from a gamma distribution with mean mu and shape parameter alpha.

See also cumgamma(), cumchi(), invchi(), runi().

2.140 invnor

Keywords: probabilities, confidence intervals
Usage: invnor(P), P REAL with elements of P between 0 and 1, df > 0

invnor(P) computes the quantiles (probability points, critical values) of the normal distribution corresponding to each element of P. P must be a REAL vector or array with elements between 0 and 1. The result has the same size and shape as P.

A critical value for a two-tail Z-test with significance level alpha or for a 1-alpha confidence interval may be computed as invnor(1-alpha/2).

Critical values for a one-tail Z-test with significance level alpha are computed as invnor(alpha) (lower tail test) and invnor(1-alpha) (upper tail test).

invnor() is the inverse of cumnor() in the sense that invnor(cumnor(z)) should be the same as z within rounding error and cumnor(invnor(P)) should be the same as P within rounding error.

See also cumnor(), rnorm()

2.141 invstu

Keywords: probabilities, random numbers, comparisons, confidence intervals
Usage: invstu(P, df), P and df REAL, elements of P between 0 and 1, those of df > 0

invstu(P,df) computes the quantiles (probability points, critical values) of Student's t- distribution with df degrees of freedom

corresponding to each element of P. P must be a REAL vector or array with elements between 0 and 1 and df must be a REAL vector or array with positive but not necessarily integral elements.

If df is a scalar the result has the same size and shape as P and df is used to compute all the values.

If P is a scalar, the result has the same size and shape as df and consists of P-th probability points for the different values of df.

If neither P nor df are scalars, they must be the same size and shape and corresponding elements of P and df are used to compute elements of the result.

A critical value for a two-tail t-test on df degrees of freedom with significance level alpha or for a 1-alpha confidence interval may be computed as invstu(1-alpha/2,df).

Critical values for a one-tail t-test on df degrees of freedom with significance level alpha are computed as invstu(alpha,df) (lower tail test) and invstu(1-alpha,df) (upper tail test).

Bonferronized critical values for K simultaneous two-tail t-tests with significance level alpha or K simultaneous 1 - alpha confidence intervals are computed as invstu(1-.5*alpha/K,df).

invstu() is the inverse of cumstu() in the sense that, within rounding error, invstu(cumstu(x,df),df) should be the same as x and cumstu(invstu(P,df),df) should be the same as P.

invstu(runi(n),df) will generate a random sample of size n from a Student's t-distribution .

See also cumstu(), runi().

2.142 invstudrng

Keywords: probabilities, comparisons, confidence intervals
Usage: invstudrng(P, ngroup, errorDf [,epsilon:eps]), elements of P between 0 and 1, elements of ngroup integers >= 2, elements of errorDf >= 1, eps > 0 small

invstudrng(P, K, Df) computes the quantiles (probability points, critical values) of the Studentized range based on K normal variates and an independent estimate of variance with Df degrees of freedom. All three arguments must be REAL. The elements of P must be between 0 and 1. K must consist of integers >= 2, and the elements of Df must be >= 1, not necessarily integers.

Any of the arguments P, K or Df that are not scalars must be vectors,

matrices or arrays all of the same size and shape.

invstudrng(P,2,Df) should be the same as sqrt(2)*invstu((1+P)/2,Df) except for computational error.

Many so-called multiple comparison methods are based on these quantiles, among them the Tukey HSD (Honestly Significant Difference) and the SNK (Student-Newman-Keuls) methods. For example, if you have K independent normal samples of size n, all with the same variance, and Ssq is the pooled estimate of the variance, you can compute the 5% HSD as q05 <- invstudrng(1-.05,K,K*(n-1));hsd <- q05*sqrt(Ssq/n)

In the same situation, you can test the null hypothesis that all means are equal by the studentized range statistic computed as Q <- (max(xbars) - min(xbars))/sqrt(Ssq/n). This is an alternative to the ANOVA F-statistic. You can compute the alpha-level critical value for Q as invstudrng(1-alpha, K,K*(n-1)). Here xbars is a vector containing the K sample means and Ssq is the pooled estimate of variance. See cumstudrng() for computing P values for Q.

invstudrng(P, K, Df, epsilon:eps), where eps is a small positive scalar, does the same computation with accuracy influenced by eps. The smaller the value of eps, the more accurate the result should be, but the longer it will take to compute it. The default value of eps is 0.00001.

See also cumstudrng(), invstu().

2.143 ipf

Keywords: glm, categorical data
Usage: ipf([Model] [, print:F or silent:T, incr:T, pvals:T, maxit:m, epsilon:eps]), vec a REAL vector, m an integer > 0, eps REAL > 0

ipf(Model) uses iterative proportional fitting to compute a Poisson regression (log linear) fit of the model specified in the CHARACTER variable Model. The default output is the deviance from the full model.

See topic 'models' for information on specifying Model.

ipf(Model,inc:T) fits the same model except a sequential analysis of deviance is computed. The sequential analysis of deviance has a line for each term in the model giving the term name, degrees of freedom, and the change of deviance obtained by including the term in the given order. Because each of the submodels must be fit iteratively, with a complicated models or a large data set ipf(Model,inc:T) can take many times longer to execute than ipf(Model).

ipf() or ipf(,inc) fits the last model used by any of the GLM commands such as regress() or poisson(). See topic glm.

ipf(Model [,...], pvals:T) prints chi-squared P values with each deviance.

If option 'pvals' has value True, P values will be printed unless pvals:F is an argument.

If there are any non-factors in the model ipf() defaults to poisson().

ipf() also defaults to poisson(), if it does not identify the model as balanced. The only forms of balance it recognizes are complete balance (equal number of cases in every cell) and balanced main effect models (no interactions and all two-way marginals have equal cell sizes, for example a Latin square design)

ipf() sets the side effect variables RESIDUALS, WTDRESIDUALS, SS, DF, HII, DEPVNAME, TERMNAMES, and STRMODEL. All except HII should be the same as computed by poisson(Model,inc) used. Since HII cannot be computed easily, it is set to a constant vector with values m/n where m = (Model degrees of freedom) and n is the number of values in the dependent variable vector. Thus sum(HII) = m as it should.

ipf(Model,maxiter:m,epsilon:eps), where m is a positive integer and eps is positive, is the same as ipf(Model) except up to m iterations may take place (the default is 25) and eps is the convergence criterion (default 1e-6). You need not specify either or both.

ipf(Model [,...], print:F) is the same as ipf(Model [,...]) except that most printing is suppressed and the only result is to set the side effect variables.

ipf(Model [,...], silent:T) does computations, creating side effect variables, but prints nothing except actual error messages.

Keyword phrase 'coefs:F' cannot be used with ipf().

Coefficients may be retrieved by coefs(); standard errors are not available. You must use poisson() if you require standard errors.

2.144 isarray

Keywords: macros, general, variables
Usage: isarray(arg1 [,arg2, ...] [,real:T, logic:T, char:T])

isarray(arg) returns True if arg is an array of any type, REAL, LOGICAL, CHARACTER or LONG, and False otherwise. If arg is undefined, isarray() returns False.

isarray(arg,real:T) returns True if and only if arg is a REAL array Similarly isarray(arg,char:T) and isarray(arg,logic:T) return True only if arg is an array of the specified type. You can specify more than one

```
acceptable type; for example, isarray(arg,real:T,logic:T) returns True
only if arg is a REAL or LOGICAL array.

isarray(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of
which is True or False depending on whether or not the corresponding
argument is a array.  You can use keyword phrases 'real:T', 'logic:T',
and 'char:T' in this case, too.

The principal usage of isarray() is in checking the arguments of a macro
for appropriateness.

Examples:
  Cmd> isarray(vector(x), matrix(x,4), array(x,2,2,2), structure(x))
      has value vector(T,T,T,F) when x has 8 elements.

   if (!isarray($1,real:T,logic:T)){
     error("ERROR: $1 is not a REAL or LOGICAL")}

in a macro would check argument 1 is REAL or LOGICAL.

See also array(), error(), ischar(), isdefined(), isfactor(), isgraph(),
islogic(), ismacro(), ismatrix(), isnull(), isreal(), isscalar(),
isstruc(), isvector().
```

2.145 ischar

Keywords: macros, general, variables, character variables
Usage: `ischar(arg1 [, arg2, ...])`

```
ischar(arg) returns True if arg is a CHARACTER variable or quoted string
and False otherwise.  If arg is undefined, ischar() returns False.

ischar(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of
which is True or False depending on whether or not the corresponding
argument is of type CHARACTER.

The principal usage of isfactor() is in checking the arguments of a
macro for appropriateness.

Example:
  Cmd> ischar("hello",3,T) # returns vector(T,F,F).

See also macros, isarray(), isdefined(), isfactor(), isgraph(),
islogic(), ismacro(), ismatrix(), isnull(), isreal(), isscalar(),
isstruc(), isvector().
```

2.146 isdefined

Keywords: macros, general, variables
Usage: `isdefined(arg1 [, arg2, ...])`

isdefined(arg) returns True if arg exists in the MacAnova workspace and False otherwise. If arg is a built-in function, isdefined() returns True.

isdefined(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of which is True or False depending on whether or not the corresponding argument actually exists.

In a macro, isdefined() is useful for checking errors in a macro. For instance, a macro to compute mean square errors might have the line

```
 if(!isdefined(SS) || !isdefined(DF)){
    error("ERROR: SS or DF not defined")}
```

before attempting to use SS or DF.

See macros, isarray(), ischar(), isfactor(), isgraph(), islogic(), ismacro(), ismatrix(), isnull(), isreal(), isscalar(), isstruc(), isvector()..

2.147 isfactor

Keywords: macros, general, glm, variables
Usage: `isfactor(arg1 [, arg2, ...])`

isfactor(arg) returns True if arg is a factor created by function factor() and false otherwise. If arg is underfined, isfactor() returns False.

isfactor(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of which is True or False depending on whether or not the corresponding argument is a factor.

The principal usage of isfactor() is in checking the arguments of a macro for appropriateness.

See also factor(), models, glm, anova(), isarray(), ischar(), isdefined(), isgraph(), islogic(), ismacro(), ismatrix(), isnull(), isreal(), isscalar(), isstruc(), isvector()..

2.148 isgraph

Keywords: macros, general, variables
Usage: `isgraph(arg1 [, arg2, ...])`

isgraph(arg) returns True if arg is a GRAPH variable and False otherwise. If arg is undefined, isgraph() returns False.

isgraph(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of which is True or False depending on whether or not the corresponding argument is a GRAPH variable.

The principal usage of isgraph() is in checking the arguments of a macro for appropriateness.

See also graphs, macros, isarray(), ischar(), isdefined(), isfactor(), islogic(), ismacro(), ismatrix(), isnull(), isreal(), isscalar(), isstruc(), isvector().

2.149 islogic

Keywords: macros, general, variables, logical variables
Usage: islogic(arg1 [, arg2, ...])

islogic(arg) returns True if arg is a LOGICAL variable and False otherwise. If arg is undefined, islogic() returns False.

islogic(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of which is True or False depending on whether or not the corresponding argument is of type LOGICAL.

The principal usage of islogic() is in checking the arguments of a macro for appropriateness.

Example:

```
  Cmd> islogic("hello",3,T) # returns vector(F,F,T).
```

See also logic, macros, isarray(), ischar(), isdefined(), isfactor(), isgraph(), ismacro(), ismatrix(), isnull(), isreal(), isscalar(), isstruc(), isvector().

2.150 ismacro

Keywords: macros, general, variables
Usage: ismacro(arg1 [, arg2, ...])

ismacro(arg) returns True if arg is a macro and False otherwise. If arg is undefined, ismacro() returns False.

ismacro(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of which is True or False depending on whether or not the corresponding argument is a macro.

The principal usage of ismacro() is in checking the arguments of a macro for appropriateness.

See also macros, isarray(), ischar(), isdefined(), isfactor(), isgraph(), islogic(), ismatrix(), isnull(), isreal(), isscalar(), isstruc(), isvector().

2.151 ismatrix

Keywords: macros, general, variables
Usage: ismatrix(arg1 [,arg2, ...] [,real:T, logic:T, char:T])

ismatrix(arg) returns True if arg is a matrix of any type, REAL, LOGICAL, or CHARACTER, and False otherwise. For arg to be considered a matrix it is not necessary that ndims(args) be 2, just that no more than two dimensions have length greater than 1. In particular, a scalar or a vector is considered to be a matrix by ismatrix(). If arg is undefined, ismatrix() returns False.

ismatrix(arg,real:T) returns True if and only if arg is a REAL matrix. Similarly ismatrix(arg,char:T) and ismatrix(arg,logic:T) return True only if arg is a matrix of the specified type. You can specify more than one acceptable type; for example, ismatrix(arg,real:T,logic:T) returns True only if arg is a REAL or LOGICAL matrix.

ismatrix(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of which is True or False depending on whether or not the corresponding argument is a matrix. You can use keyword phrases 'real:T', 'logic:T', and 'char:T' in this case, too.

The principal usage of ismatrix() is in checking the arguments of a macro for appropriateness.

Examples:

```
Cmd> ismatrix(vector(x), matrix(x,4), array(x,4,1,2), array(x,2,2,2))
    has value vector(T,T,T,F) when x has 8 elements.

 if (!ismatrix($1,real:T)){error("ERROR: $1 is not a REAL matrix")}
   in a macro would check argument 1 is a REAL matrix.
```

See also matrices, error(), isarray(), ischar(), isdefined(), isfactor(), isgraph(), islogic(), ismacro(), isnull(), isreal(), isscalar(), isstruc(), isvector().

2.152 ismissing

Keywords: macros, general, missing values, variables, null variables
Usage: `ismissing(x)` where x is REAL, LOGICAL or CHARACTER or a structure all of whose components are REAL, LOGICAL, or CHARACTER

ismissing(x) returns a LOGICAL variable (with the same shape as x) which is True where x is MISSING and False where x is not MISSING. x must be a vector, matrix, or array. If x is a CHARACTER variable, an empty string ("") is considered to be a missing value.

If x is a NULL variable, ismissing(x) is NULL.

It is also acceptable for x to be a structure, whose non-structure components are vectors, matrices or arrays. In that case, ismissing() returns a structure of the same form, each of whose non-structure components is the result of applying ismissing() to the corresponding component of x.

Examples:
- ismissing(vector(1, 3, ?, 7)) and ismissing(vector("A","B","","Z")) both return vector(F, F, T, F)
- x[vector(ismissing(x))] <- -1 replaces all MISSING values in x by -1.
- sum(vector(ismissing(x))) returns the number of MISSING values in x, whether x is a vector, matrix, array, or structure.

See also anymissing(), sum().

2.153 isnull

Keywords: macros, general, variables, null variables
Usage: `isnull(arg1 [, arg2, ...])`

isnull(arg) returns T if arg has type NULL and false otherwise. You can create a NULL variable by 'var <- NULL'. In addition, the values of functions that are not normally thought of as having a value, for example print() and regress(), are NULL. Thus, 'var <- anova("y=a+b")' creates a NULL variable var.

isnull(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of which is True or False depending on whether or not the corresponding argument is NULL.

The principal usage of isnull() is in checking the arguments of a macro for appropriateness.

Example:
```
Cmd> isnull(NULL, sqrt(2)) # returns vector(T, F)
```

See also macros, isarray(), ischar(), isdefined(), isfactor(), isgraph(), islogic(), ismacro(), ismatrix(), isreal(), isscalar(), isstruc(), isvector().

2.154 isreal

Keywords: macros, general, variables
Usage: isreal(arg1 [, arg2, ...])

isreal(arg) returns True if arg is a REAL variable and False otherwise. If arg is undefined, isreal() returns False.

isreal(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of which is True or False depending on whether or not the corresponding argument is REAL.

The principal usage of isreal() is in checking the arguments of a macro for appropriateness.

Example:
```
  Cmd> isreal("hello",3,T)  # returns vector(F,T,F).
```

See also macros, isarray(), ischar(), isdefined(), isfactor(), isgraph(), islogic(), ismacro(), ismatrix(), isnull(), isscalar(), isstruc(), isvector().

2.155 isscalar

Keywords: macros, general, variables
Usage: isscalar(arg1 [,arg2, ...] [,real:T, logic:T, char:T])

isscalar(arg) returns True or False, depending on whether arg is a scalar, that is a REAL, LOGICAL, or CHARACTER variable all of whose dimensions are 1. If arg is undefined, isscalar() returns False.

isscalar(arg,real:T) returns True if and only if arg is a REAL scalar. Similarly isscalar(arg,char:T) and isscalar(arg,logic:T) return True only if arg is a scalar of the specified type. You can specify more than one acceptable type; for example, isscalar(arg,real:T,logic:T) returns True only if arg is a REAL or LOGICAL scalar.

isscalar(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of which is True or False depending on whether or not the corresponding argument is a scalar. You can use keyword phrases 'real:T', 'logic:T', and 'char:T' in this case, too.

The principal usage of isscalar() is in checking the arguments of a macro for appropriateness.

```
Examples:
  Cmd> isscalar(1,matrix(PI,1), run(5),"hello",F)
has value vector(T,T,F,T,T)

In a macro
  if (!isscalar($1,logic:T)){error("ERROR: $1 not T or F")}
would check that argument 1 is a LOGICAL scalar.

See macros, isarray(), ischar(), isdefined(), isfactor(), isgraph(),
islogic(), ismacro(), ismatrix(), isnull(), isreal(), isstruc(),
isvector().
```

2.156 isstruc

Keywords: macros, general, structures
Usage: `isstruc(arg1 [, arg2, ... ])`

```
isstruc(arg) returns True or False, depending on whether arg is a
structure.  If arg is undefined, isstruc() returns False.

isstruc(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of
which is True or False depending on whether or not the corresponding
argument is a structure.

The principal usage of isstruc() is in checking the arguments of a macro
for appropriateness.

See also structures, macros, isarray(), ischar(), isdefined(),
isfactor(), isgraph(), islogic(), ismacro(), ismatrix(), isnull(),
isreal(), isscalar(), isvector().
```

2.157 isvector

Keywords: macros, general, variables
Usage: `isvector(arg1 [,arg2, ...]  [,real:T, logic:T, char:T])`

```
isvector(arg) returns True or False, depending on whether arg is a
vector of any type, REAL, LOGICAL or CHARACTER.  A matrix or array is
considered to be a vector by isvector() if all dimensions except the
first have length 1.  In particular, if arg is a scalar, isvector(arg)
returns True, while if arg is a row vector (dimensions 1,m with m > 1),
isvector(arg) returns False.  If arg is undefined, isvector(arg) returns
False.

isvector(arg,real:T) returns True if and only if arg is a REAL vector.
Similarly isvector(arg,char:T) and isvector(arg,logic:T) return True
only if arg is a vector of the specified type.  You can specify more
```

than one acceptable type; for example, isvector(arg,real:T,logic:T) returns True only if arg is a REAL or LOGICAL vector.

isvector(arg1, arg2, ..., argk) returns a LOGICAL vector, each element of which is True or False depending on whether or not the corresponding argument is a vector. You can use keyword phrases 'real:T', 'logic:T', and 'char:T' in this case, too.

The principal usage of isvector() is in checking the arguments of a macro for appropriateness.

Examples:

```
  Cmd> isvector(7, vector(x), matrix(x,5), array(x,5,1,1), matrix(x,1))
```

has value vector(T,T,T,T,F) if x has 5 elements.

In a macro

```
  if(!isvector($1,char:T)){
        error("ERROR: $1 is not a CHARACTER vector")}
```

would check that argument 1 is a CHARACTER vector.

See also vectors, macros, isarray(), ischar(), isdefined(), isfactor(), isgraph(), islogic(), ismacro(), ismatrix(), isnull(), isreal(), isscalar(), isstruc().

2.158 keyvalue

Keywords: syntax, macros

Usage: keyvalue(keyname1:value1, [keyname2:value2, ...] targetname, type), targetname, type CHARACTER scalars
keyvalue(str, targetname, type), str a structure
keyvalue(, targetname, type)

keyvalue(keyname1:value1, keyname2:value2, ... , TargetName, Type) attempts to match keyname1, keyname2, ... with TargetName. If no match is found, keyvalue() returns NULL. If a match is found, the corresponding keyword value is returned provided its type matches the type specified by Type.

TargetName and Type must be quoted strings or CHARACTER scalars. The permissible values for Type are restricted to "real", "logic", "character", "structure", "macro" or "graph". It is an error if the type of the value of a matched keyword does not match Type.

keyvalue(structure(keyname1:value1, keyname2:value2, ...), TargetName, Type) is equivalent to keyvalue(keyname1:value1, keyname2:value2, ... , TargetName, Type).

keyvalue(, TargetName, Type) is legal and returns NULL. Arguments TargetName and Type are checked for appropriateness.

The principal use for keyvalue() is in writing macros. Here is a fragment of a macro that recognizes an optional macro argument of the form 'weights:w' where w must be REAL.

```
@weights <- keyvalue($K, "weights", "real")
@weights <- if(isnull(@weights)){rep(1,@n)}else{@weights}
```

If 'weights:w' is an argument to the macro, @weights is set to w; otherwise @weights is set to the default rep(1,@n). Presumably @n is set previously.

keyvalue(structure($K),...) is almost equivalent to keyvalue($K,...), except that a warning message is printed when there are no keyword arguments to the macro ($K expands to nothing).

See also keywords, macros.

2.159 keywords

Keywords: syntax
Usage: print(nsig:5,x), plot(ObsNo:x,Response:y), regress(Model, pvals:T), setoptions(format:"12.5g") are examples of keyword usage

Some functions accept 'keyword phrases' as optional arguments. A keyword phrase has the form Name:argValue (Example: 'nsig:7'), where Name is a name of no more than 10 characters and argValue is a variable or constant. 'T' and 'F' can be used as keyword names. Name cannot start with '_'.

Keywords often provide optional information, or specify variants of the usual computation. Thus print(nsig:7,x,y,nsig:3,z) prints x and y with 7 significant digits and z with 3, and screen(Model,mbest:6) specifies that screen() should find the 6 best subsets of independent variables.

Keyword phrases may also be used to name components when using structure() and strconcat(), to label output on most output commands (print, write, etc.), and to provide labeling information on plotting commands such as plot() and chplot(). The only limitation is that any keyword that is specially recognized by the command such as 'labels' or 'format' cannot be used as such a label.

See also syntax, structure(), strconcat(), keyvalue()

2.160 kmeans

Keywords: multivariate analysis

Usage: kmeans(y [,means or classes] [,kmax:k1, kmin:k2, start:method, standard:F, weights:wts, quiet:T]), y a REAL matrix, means a REAL matrix with ncols(y) columns, classes a REAL vector with nrows(y) rows, k1 and k2 positive integers, k1 >= k2, method one of "random", "optimal", "means", or "classes", wts a REAL vector with nrows(wts) = nrows(y)

kmeans(y, kmax:k1 [, kmin:k2]) performs k-means clusterings of the rows of REAL matrix y, starting with k1 clusters, and successively merging clusters until there are k2 clusters. By default the data are standardized and the initial clusters are selected randomly. At each stage, cases are reallocated among clusters in an attempt to minimize the sum of the within-cluster sums of squares. If kmin:k2 is omitted, k2 is taken to be k1.

kmeans() returns a structure with components 'classes' and 'criterion'. Component classes is a nrows(y) by k2-k1+1 matrix (vector if k2 = k1) containing the cluster membership at each stage. Component criterion is a k2-k1+1 REAL vector containing the minimized criterion at each stage.

By default, a brief history of the merging process is printed, including the values of the criterion being minimized.

kmeans(y, kmax:k1 [, kmin:k2], start:"random") is identical to kmeans(y, kmax:k1 [, kmin:k2]).

kmeans(y, kmax:k1 [, kmin:k2], start:"optimal") attempts to select the initial clusters so as to minimize the within-cluster sums of squares for column 1 of y.

kmeans(y, Means [, kmin:k2], start:"means"), where Means is a k1 by ncols(y) matrix, selects as initial cluster j those rows of y that are closer to row j of Means than to any other row of Means using (Euclidean distance). If kmax:k1 is an argument with k1 != nrows(Means), a warning message is given and nrows(Means) is used. If there are duplicates among the rows of Means, a warning message is printed.

kmeans(y, Classes [, kmin:k2], start:"classes"), where Classes is a vector of nrows(y) positive integers <= 255, uses Classes to specify initial clusters. If kmax:k1 is an argument with k1 != max(Classes), a warning message is given and max(Classes) is used. If there are empty classes (not all integers between 1 and max(Classes) are present), the empty classes are "squeezed out", and max(Classes) reduced accordingly.

It is an error if k2 > k1.

Additional keywords

standard:F	Do not standardize before clustering
weights:wts	Use weighted means and sums of squares

```
                              with wts a REAL vector of length
                              nrows(y) with w[i] > 0.
      quiet:T                 Suppress printing of clustering
                              history.

See also cluster().
```

2.161 labels

Keywords: general, variables, output

Usage:

```
y <- vector(x,labels:labs [, silent:T])
y <- matrix(x,labels:structure(rowLabs,colLabs) [, silent:T])
y <- array(x,labels:structure(lab1,lab2,...) [, silent:T])
y <- array(x,labels:NULL) #remove labels
y <- matread(fileName,labels:structure(rowLabs,colLabs))
labs <- getlabels(x [,silent:T])
labs <- getlabels(x,vector(1,3,5) [,silent:T])
if (haslabels(x)){..do something with labels...}
```

MacAnova vectors, matrices and arrays may have labels for each dimension. When a variable x has any labels it has them for all dimensions. The label for dimension k of x is a CHARACTER vector with length dim(x)[k].

A structure str may have a single vector of labels of length ncomps(str) to label the components.

The primary function of the labels for a variable is to provide informative identification of coordinates when the variable is printed.

In many, but not all situationse, when x has labels, they are propagated to new variables computed from x. See below for details.

getlabels(x) retrieves all the labels, if any, associated with variable x. getlabels(x,vector(1,3)), for example, retrieves the labels associated with dimensions 1 and 3 of x as a structure with two components. See getlabels().

haslabels(x) is True if and only if variable x has labels associated with it.

Removing Labels

x <- array(x, labels:NULL), removes the labels from a scalar, vector, matrix or array x.

x <- strconcat(x, labels:NULL) removes labels from a structure, but not from its components.

Adding Labels

You can create a variable with labels or add labels to a variable using

vector(), matrix(), array(), structure(), and strconcat(). The general usage is to include keyword phrase 'labels:labs' as an extra argument, where labs is a CHARACTER vector or scalar, or a structure of CHARACTER vector or scalar components, one for each dimension of the variable to be labeled.

You can also use keyword phrase 'labels:labs' on matread() to add labels to a vector, matrix or array being read from a file.

It is not an error if the number of labels supplied does not match the number of dimensions. Extra labels are ignored and missing ones are assumed to be "@" which will be printed as numbers in parentheses (see below). In both cases a warning message is normally printed.

If the label supplied for a dimension is a vector of the wrong length, a warning message is printed, and the operation is carried out ignoring all labels.

On any command with this use of keyword 'labels', you can suppress any warning messages by keyword phrase silent:T.

Expanding Scalar Labels

Whenever labels are provided using keywork 'labels', scalar labels (quoted strings or CHARACTER vectors of length 1) are treated specially.

If a scalar label, say "root", for a coordinate with length > 1 doesn't start with '@' and is not one of "", "#", "(", "[", "{", "<", "/", or "\\", it is expanded to vector("root1","root2",...). Thus labels:structure("A","B ") generates labels vector("A1","A2",...) and vector("B 1","B 2",...) .

Scalar label "" is expanded to rep("", length) resulting in its dimension having no labels.

Scalar label "#" is expanded to vector("1","2",...).

Scalar label "(" is expanded to vector("(1)","(2)", ...) and similarly for the other special characters, with the first element of the label being "[1]", "{1}", "<1>", "/1/", or "\\1\\".

A scalar label starting with '@', say "@anything" is expanded to rep("@anything", n) and is further expanded when it is printed.

Expansion of labels when they are printed

A label of the form rep("@anything", n), as occurs when scalar "@anything" is part of the value of keyword 'labels', is interpreted specially when it is printed. It is further expanded similarly to the way scalar labels that do not start with '@' are expanded when they are created.

rep("@#", n) prints as '1', '2', ...

rep("@[", n) prints as '[1]', '[2]', ..., and similarly with "@(", "@{",

"@<", "@/",or "@\\".

rep("@", n) prints "bracketed" labels using the default labeling style, usually using '(' and ')'. This style can be changed by option 'labelstyle'. Thus, for example, after setoptions(labelstyle:"["), "@" has the same effect as "@[". See setoptions().

rep("@anythingelse", n) prints as 'anythingelse1', 'anythingelse2', ...

Note: The use of '@' delays the substitution of numerical indices until they are actually used in printing, so that the same value may have different printed labels at different times. See the paragraph on propagation of labels below.

If successive coordinates have the same type of "bracket" label starting with '@' created by, say, labels:structure("@[","[@","["), the printed labels are combined to, say, '[1,2]'.

If all the labels are specified by scalars "@", the effect is the same as when an unlabeled variable is printed after setoptions(labelabove:T). See setoptions().

Examples

```
  Cmd> x <- matrix(x, labels:structure("#","X"))
```

x has labels vector("1","2",...) and vector("X1","X2",...)

```
  Cmd> y <- vector(vecread(fileName),labels:structure("Case ","Y"),\
       silent:T)
```

y has labels vector("Case 1", "Case 2", ...) and vector("Y1", "Y2", ...)

```
  Cmd> z <- matread("MacAnova.dat","irisdata",\
       labels:structure("",vector("Variety","Y1","Y2","Y3","Y3")))
```

The first label of z is rep("",nrows(z)) which does not print.

```
  Cmd> logy <- matrix(log(y),labels:structure("@",log(getlabels(y,2))))
```

The labels of logy y will be rep("@",nrows(y)) and vector("log(Y1)", "log(Y2)", ...). When printed the row labels of logy or any subset of rows of logy will be '(1)', '(2)', ...

Propagation of Labels

A variable created by extracting part of a labeled variable using subscripts is labeled with the appropriate subsets of the labels. Thus for example,

```
   matrix(run(9),3,labels:structure("[","A "))[-1,-1]
```

has labels vector("[2]","[3]") and vector("A 2", "A 3"). However,

```
   matrix(run(9),3,labels:structure("@[","@A "))[-1,-1]
```

has labels vector("@[","@[") and vector("@A ","@A ") which will print as '[1]', '[2]' and 'A 1', 'A 2', even though these are rows 2 and 3 and columns 2 and 3 of matrix(run(9),3). That is, the special expansion of labels starting with '@' occurs when they are printed, not when they are created.

cos(x), sqrt(x), and other transformation of x have the same labels as x.

x' has the same label vectors as x but in reverse order

sum(x), min(x), and other transformation that operate along the first dimension of x have labels for the last ndims(x) - 1 dimensions matching those of x.

+x, -x and !x all have the same labels as x.

If OP is a binary operator such as '+', '-', '*', '==', ..., but not a matrix operator such as %*%. %c%, and %C%, then x OP y often has the labels of x. When x does not have labels, x OP y may have the labels of y.

If matrices x and y both have labels then x %*% y, x %c% y, and x %C% y have labels taken from the row and or column labels of x and y in the obvious way.

If x is a matrix with labels, eigen(x)$vectors and releigen(x,y)$vectors have the same row labels as x and null column labels.

If x is a matrix with labels, cor(x) has row and column labels matching the column labels of x. If cor() has more than 1 argument, its result has no labels.

If x is a matrix with labels, rft(x) and hft(x) have the same column labels as x with null row labels. The same is true for cft(x) when ncols(x) is even.

If x is a response variable in a GLM command, its labels are propagated to side effect variables RESIDUALS, WTDRESIDUALS, and HII.

After regress(), COEF and XTXINV are labeled with the names of the variables (including "CONSTANT" when appropriate).

After manova() with a multivariate response with labels, SS is labeled with TERMNAMES and two copies of the column labels of the response. Also, the column labels of the response are attached to the last dimension of each vector, matrix, or array returned by coefs() and secoefs().

If any term names are longer than 12 characters (the maximum size for a structure component name), the output of coefs() and secoefs() is labeled with the full term names.

2.162 launching

Keywords: general, files

Usage: Unix and DOS: macanova [-q] [File Options] [Screen Options]
Macintosh: double click on MacAnova icon or MacAnova file icon.
Windows: double click on MacAnova icon on Program Manager desktop

For non-windowed versions (Unix or DOS), type 'macanova' at the Unix or DOS prompt. If the MacAnova directory is not in the search path, you will need to specify the complete path. See below for command line options.

Launching Windows/Windows 95 Version

Under Windows, if a MacAnova icon for any version has been defined in the Program Manager, you can start up MacAnova by double clicking on the icon. Otherwise, use Run on the Program Manager File menu specifying the path name for MacAnova.

Under Windows 95, if a desktop icon (shortcut) has been defined for any version of MacAnova, you can double click on it; or you can use the Navigator to find MacAnova and double click on it.

The Windows version of MacAnova will start up with an Untitled command/output window. The DOS versions will start up in a DOS window. You cannot start the Windows version from the DOS prompt under Windows 3.1 but you can under Windows 95.

See also dos-windows, wx.

Launching Motif Version of MacAnova

Assuming the Motif version has been named macanovawx and is in a directory in your search path, type

macanovawx [-q] [File Options] [Path Options] [Screen Options] &

at the Unix prompt, where items in [...] are options. See below for details on command line options. This will open an Untitled MacAnova command/output window. The trailing '&' is not required but allows you to type more commands in the window from which you launched MacAnova without quitting MacAnova.

See also unix, wx.

Launching MacAnova on a Macintosh

MacAnova may be started up in several ways.

1. Double click the MacAnova icon.

2. Double click on the icon associated with a file previously created by save() or asciisave() or either the Save Workspace or Save Workspace As... items on the File menu. This restores the workspace previously saved.

3. Double click on a the icon associated with a file previously created by the Save Window or Save Window As... items on the File menu. This initializes the command window with commands and output from previous run.

4. Select both the icon associated with a file previously created by save() or asciisave() or Save Workspace ... and the icon associated with a file previously created by Save Window or Save Window As and then double click. This restores both the previous workspace and previous output. This means you can interrupt a session and restore things exactly to how they were at the point of interruption.

If you hold down the Option or Q keys (or both), no startup message is displayed or printed.

Macintosh Non-Interactive ("batch") Mode

If you hold down either the Command or B keys (or both) during lauching, MacAnova will be run in non-interactive mode without a command/output window, and with commands and their output written ("spooled") directly to a file. A scrolling dialog box allows you to select this file. If you click on Cancel, then standard interactive mode will be used. Next, another scrolling dialog box allows you allows you to select a file which contains MacAnova commands to be run.

If you launched MacAnova by double clicking a file created by Save Window or Save Window As and hold down Command or B, the second dialog box will be skipped and the clicked-on file will be used as the file of commands. You can make such a file from within MacAnova by selecting the New Window item on the Windows menu to open a new command/output window. Then type in the commands *before* the prompt 'Cmd>'. When you are satisfied, delete 'Cmd>' and use Save As on the File menu to save the commands you have entered in a file.

See also macintosh.

Launching non-Windowed MacAnova on Unix or DOS

On Unix or DOS you start MacAnova by typing

```
MacAnovaName [-q] [File Options] [Path Options] [Screen Options]
```

at the Unix or DOS prompt. In Unix, MacAnovaName is usually macanova; in DOS MacAnovaName is macanodj (extended memory version) or macanobc (limited memory version).

If file cmdFile contains MacAnova commands to do an entire analysis,

```
MacAnovaName [options] < cmdFile > outputFile
```

will save all the results in file outputFile

Items in [] are optional. You can use most of the command line options on the Motif version or on the Windows version when it is started by selecting Run on the Program Manager File menu.

If -q is present, the startup message will not be printed.

File Options for Unix and DOS versions

```
-f initFile
File initFile is executed silently as a batch file at startup instead of
file .macanova.ini in Unix or MACANOVA.INI in DOS (see customize).

-restore saveFile  or  -r saveFile
  The equivalent of 'restore("saveFile")' is executed at startup and
  .macanova.ini (Unix) or MACANOVA.INI (DOS) is not read.  See
  customize, restore().

-batch batchFile  or  -b batchFile
  The equivalent of 'batch("batchFile")' is executed after
  initialization.

-bprompt Prompt  or  -bp Prompt
  Sets a prompt to be used with echoed commands in batchFile; this is
  meaningful only with -batch batchFile.  Usually Prompt should end with
  a space, for example, -bprompt "HW 1> "..

-prompt Prompt  or  -p Prompt
  Sets the non-batch command line prompt.  Usually Prompt should end with
  a space, for example, -prompt "Next? ".  This becomes the default
  prompt the will be set by setoptions(default:T).  See setoptions().

-help helpFile  or  -h helpFile
  Help information will be taken from file helpFile rather than the
  default help file.

-data dataFile  or  -d dataFile
  Pre-defined CHARACTER variable DATAFILE will have "dataFile" as value
  instead of a default value.  DATAFILE is used by pre-defined macro
  getdata to make it easy to read data from a standard file.  See
  getdata.

-macro macroFile  or  -m macroFile   (only -macro in Motif version)
  "macroFile" will be added to the beginning of Pre-defined CHARACTER
  variable MACROFILES.  This will mean that pre-defined macro getmacros
  will search the file before the standard macro files.  You can
  accomplish the same thing after starting MacAnova by
  addmacrofile("macroFile").  See getmacros and addmacrofile.

                  Path Options for Unix and DOS versions
-home homePath
  Predefined CHARACTER variables HOME will have "homePath" as value
  instead of a default value.  HOME is used to expand file names of the
  form "~/filename".  For instance, when HOME is "dataDir", "~/filename"
  is expanded to "dataDir/fileName".  See files.

-dpath dataPath  or  -dp dataPath
  Predefined CHARACTER variables DATAPATHS and DATAPATH will be set to
  "dataPath".  When you attempt to read a file, say using vecread(),
  matread() or macroread(), if it cannot be found in the default
  directory or folder (see files), MacAnova will search in the
  directories or folders in DATAPATHS.  If this option is not used, the
```

```
  default is the value of HOME (see -home above) except in Motif or Unix
  versions where it is the name of an installation-dependent standard
  data directory such as "usr/local/data".

-mpath macroPath  or  -mp macroPath
  "macroPath" will be added to predefined CHARACTER vector DATAPATHS.
  See discussion of -dpath above.

If any of the file names or path names is not a legal file or path name,
MacAnova immediately terminates.

Even if -data, -macro, -home, -dpath or -mpath are used, the default
values of DATAFILE, MACROFILES and DATAPATHS can be changed in your
startup file (see topic customize).  The help file can be changed by
help(file:FileName) which can also be in the startup file.  See help().

            Screen Options for Unix and DOS/Windows versions
-l Nlines
  This pre-defines setoptions() variable 'height' to be Nlines, where
  Nlines is either 0 or an integer at least 5.  See setoptions().

-w Ncols
  This pre-defines setoptions() variable 'width' to be Ncols, where
  Ncols is an integer at least 20.  See setoptions().

On the Windows and Motif versions and any non-windowed version for which
line editing is available (Unix and the DOS extended memory version,
DJGPP) the following additional option is available.

-history Nhist or -hist Nhist
  This pre-defines setoptions() variable 'history' to be Nhist, a non-
  negative integer.  This limits the number of previous command lines
  that can be saved and recalled to Nhist.  The default value is 100.
  See dos-windows, wx, unix.

See also quitting, customize.
```

2.163 length

Keywords: variables, null variables
Usage: `length(x)`, x a vector, matrix, or array

`length(x)` computes the total number of elements in the vector, matrix, or array x. `length(x)` is equivalent to `prod(dim(x))`.

If x is a NULL variable, `length(x)` is 0.

If x is a structure, `length(x)` is a structure. If xi is the i-th component of x, the i-th component of `length(x)` is `length(xi)`. If xi is a null component as might be produced by `split()`, the i-th component of `length(x)` is 0.

See also dim(), ndims(), ncomps().

2.164 lineplot

Keywords: plotting
Usage: lineplot(x,y [,linetype:m, impulse:T] [,other graphics keyword phrases]), where x is a REAL vector or scalar, y is a REAL vector or matrix, and m >= 0 is an integer

lineplot(x,y) makes a scatterplot of the data in vector x and vector or matrix y, drawing lines between the successive points. If y has more than one column, each column is graphed separately with different line types, solid, dashed, etc. You can use keywords 'linetype' and 'thickness' to control the type of lines used. The effects of 'linetype' and 'thickness' depend on the computer system on which MacAnova is running. See graphs.

lineplot(Struc), where Struc is a structure with at least two REAL components, is equivalent to lineplot(Struc[1], Struc[2]). Thus lineplot(x,y) and lineplot(structure(x,y)) are equivalent. Any components beyond the first two are ignored.

lineplot(graph,x,y) or lineplot(graph,Struc), where graph is a GRAPH variable, draws the plot encapsulated in graph, adding to it the new information. See topic graphs for details on adding information to a plot.

If y has more columns than there are different line types, line types will repeat cyclically.

Alternatively, arguments x and y can be replaced by a structure with at least two REAL components which are used instead of x and y. Thus lineplot(x,y [,...]) and lineplot(structure(x,y)[,...]) are equivalent. Any components beyond the first two are ignored.

If option 'dumbplot' has been set False (see setoptions()), the plot will be a low resolution plot unless 'dumb:F' is an argument.

Generally, lineplot() should be used only when the values in x are in increasing or decreasing order.

lineplot(x,y,add:T) and lineplot(structure(x,y),add:T) are equivalent to addlines(x,y) and addlines(structure(x,y)).

lineplot(x,y,impulses:T) draws vertical lines to the points from the x = 0 line, in addition to drawing connecting lines

See topic graphs for the use of a scalar or length 2 vector for x, and for the use of keywords title, xlab, ylab, xmin, xmax, ymin, ymax,

xaxis, yaxis, dumb, add, file, linetype and thickness.

Use addchars(), addlines(), and addpoints() to add information to a plot.

See topic graphs for information on how to save and print plots on a Macintosh, and for information on writing graphic information to a file.

See also chplot(), plot(), showplot(), addchars(), addlines(), addpoints(), tek, tekx, vt, vtx.

2.165 list

Keywords: general

Usage: list([invis:T]) or list(var1 [, var2, ...])
list([all:T, real:T or F, char:T or F, logic:T or F, macro:T or F, struct:T or F, null:T or F, keep:T, nrows:n1, ncols:n2, ndims:n3]), use F's only with all:T, n1, n2, n3 > 0 integers

list() lists the name, type, and dimensions of all currently active variables, including structures and macros.

list(invis:T) does the same, but also includes temporary variables and variables whose names start with '_'.

list(var1, var2, ..., vark) gives the same information only for the specified variables.

For a macro, list() also prints 'out-of-line' or 'in-line' depending on whether or not it has been marked to be expanded out-of-line. See macros, macro().

list(size:T [,invis:T]) or list(var1,var2,...,vark,size:T) also lists the total size of each variable in bytes. This total includes a fixed amount (164 in one Unix implementation) of overhead for each symbol.

list(varType:T [,invis:T]) where varType is one of 'real', 'factor', 'logic', 'char', 'macro', 'struc', or 'null' specifies that all variables of the specified types are listed. More than one keyword phrase can appear but no variable names. For example, list(real:T, logic:T) will list all variables of type REAL or LOGICAL and list(factor:T) will list all variables that are factors.

list(all:T,varType:F [,varType:F...] [,invis:T]) lists all types except those specified. For example, list(all:T,macros:F) lists all objects except macros.

list(nrows:r [,...]) limits variables listed to REAL, CHARACTER or LOGICAL with first dimension r.

list(ncols:c [,...]) limits variables listed to REAL, CHARACTER or LOGICAL with second dimension c. If c = 1, vectors are also listed. nrows and ncols can be used together.

list(ndims:d [,...]) limits variables listed to REAL, CHARACTER or LOGICAL with exactly d dimensions.

Keywords nrows, ncols and ndims can be used together and with char:T, real:T, or logic:T

list(pat:Pattern,... [,invis:T]) where Pattern is a quoted string or CHARACTER variable starting and/or ending with '*' lists only objects whose names match Pattern. If Pattern is of the form "*Part", where Part is a sequence of characters legal for names, only variables whose names end with Part will be listed. If Pattern is of the form "Part*", only variables with names starting with Part will be listed. If Pattern is of the form "*Part*", only variables with names containing Part will be listed.

If Pattern is a quoted string, the keyword 'pat' may be omitted. Thus, list("a*") is equivalent to list(pat:"a*"); even if var is a CHARACTER variable with value "a*", list(var) will give information only about var itself.

If a variable is "special" (currently the only special variables are CLIPBOARD and, in Motif version, SELECTION) the type is preceded by "*". See clipboard.

Examples:

```
  Cmd> list("*plot") # lists colplot but not plot1 or myplots,
  Cmd> list("plot*") # lists plot1 but not colplot or myplots, and
  Cmd> list("*plot*") # lists all three.
```

list(...,keep:T [,invis:T]) suppresses the listing, but returns a CHARACTER vector containing the names of the variables that would otherwise have been listed; no information on type or dimensions is returned.

Example:

```
  Cmd> list("a*", real:T) # or list(pat:"a*", real:T)
```

will list all REAL variables whose names start with "a".

See also delete(), listbrief(), dim().

2.166 listbrief

Keywords: general
Usage: listbrief([invis:T]) or listbrief(var1 [, var2, ...])
listbrief([all:T, real:T or F, char:T or F, logic:T or F, macro:T or F, struct:T or F, null:T or F, keep:T, nrows:n1, ncols:n2, ndims:n3]), use F's only with all:T, n1, n2, n3 > 0 integers

listbrief() lists the names of currently active variables, including structures and macros. No information on type or dimension is given.

listbrief(invis:T) does the same, but also includes temporary variables and variables whose names start with '_'.

listbrief(var1, var2, ..., vark) does the same for specified variables, except that undefined variables in the list are identified.

listbrief(pat:Pattern [,invis:T]) lists variables whose names match the value of quoted string or CHARACTER variable Pattern. If Pattern is a quoted string, 'pat:' may be ommitted. See listbrief() for details.

You can also use keywords 'real', 'factor', 'logic', 'char', 'macro', 'struc', 'null', 'all', 'keep', 'nrows' , 'ncols', and 'ndims' as in list().

See also delete().

2.167 loadUser

Keywords: general, control, files
Usage: loadUser(fileName [,reload:T or clear:T]), CHARACTER scalar fileName.

loadUser(FileName) loads a user function (separately compiled routine) into MacAnova. FileName should be a quoted string or CHARACTER scalar giving the name of the file containing the user function to be loaded. Once loaded, a user function can be executed by function User(). As usual, in windowed versions (Macintosh, Windows, Motif), FileName can be "". If the file has been previously loaded, it is not reloaded, but it may be put at the start of the entry search list for the next use of User().

loadUser(FileName, reload:T) does the same, except that the file will be reloaded, even if it has been previously loaded into MacAnova.

loadUser(FileName, clear:T) does the same, except all previously loaded files will be forgotten.

On some systems, the user function can be written in Fortran, although some features such as call back functions and argument checking may not

```
be available.

Functions loadUser() and User() are inherently specific to a particular
computer system although it is possible to write user functions that can
be compiled on multiple systems without change.

Unix:
  FileName must be the name of a shared library.
Windows:
  FileName must be the name of a DLL.
Extended memory DOS (DJGPP):
  FileName must be the name of a .dxe file linked by program dxegen.
Macintosh:
  FileName must be the name of a file containing one or more code
  resources.  The PPC version of MacAnova can call both 68K and PPC code
  resources, but a 68K version of MacAnova can call only 68K code
  resources.  Resource types must be one of 'MVPP' (PPC), 'MV6n' (68K
  without math coprocessor) or 'MV6c' (68K with math coprocessor).

See also User() and, in file Userfun.hlp, user fun (type
help(file:"Userfun.hlp", user fun)).
```

2.168 logic

Keywords: variables, syntax, logical variables, missing values, operations

Usage: `a && b, a || b, !a,` where a and b are LOGICAL or structures with LOGICAL components

```
Elements of a LOGICAL variable have only three possible values -- True,
False, and MISSING.  A LOGICAL variable may be a vector, matrix, or
array.

Logical values are printed as 'T' (True) and 'F' (False), the same
symbols as you use to enter them.  Thus 'a <- vector(T,F,F,T)' creates a
LOGICAL vector of length 4,

When used as the value of a keyword phrase, as in 'quiet:T', T and F can
usually be interpreted as 'yes' and 'no', respectively.

You can also create LOGICAL data as the result of comparing REAL,
LOGICAL or CHARACTER variables using the following comparison operators:
Comparison Operator  Precedence  Meaning
    a == b               8       Equal or same
    a != b               8       Not equal or different
    a < b                8       Less than
    a <= b               8       Less than or equal
    a > b                8       Greater than
    a >= b               8       Greater than or equal

As you would expect, precedence is lower than all arithmetic operations
```

(see arithmetic) so that, for example, 3*4 == 14-2 is interpreted as (3*4) == (14-2) and is True.

CHARACTER variables are compared using the ASCII collating sequence. Most punctuation and all numerals are "less than" upper case letters which in turn are "less than" lower case letters. A space is "less than" all printable characters. Here is the explicit ordering starting with space:

```
    !"#$%&'()*+,-./0123456789:;<=>?
   @ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
   `abcdefghijklmnopqrstuvwxyz{|}~
```

On computers with extended character sets, the ordering is dependent on the internal representation of the characters.

Behavior when either or both operands are MISSING:
Operators '<', '<=', '>', or '>=': The result is MISSING.
Operator '==' : Result is True only when if both operands are MISSING
Operator '!=' : Result is True only when only one operand is MISSING.

Comparison operators are most useful with REAL and CHARACTER data. When they are used with LOGICAL data, True and False are interpreted as 1 and 0, respectively, in the same way as with arithmetic operators +, -, *, and ^ or ** (see arithmetic). In particular F == F and T == T are True and F == T and T < F are False.

There are three purely logical operators.

Logical Operator	Precedence	Meaning
a || b	5	Logical Or (T||T,T||F,F||T are True, F||F False)
a && b	6	Logical And (T&&T is True, T&&F,F&&T,F&&F False)
!a	7	Logical Not (!T is False, !F is True)

If any operand of '||', '&&' or '!' is MISSING, so is the result . The precedence is lower than that of arithmetic expressions (see arithmetic) and that of comparison operations. Thus 'x > 1 && x < 4' is interpreted as '(x > 1) && (x < 4)', and is True if and only if the value of x is greater than 1 and less than 4. Because the precedence of '!' is less than the precedence of the comparison operators, expressions like '! a < b' are evaluated as '!(a < b)'.

Comparison operators do not "associate". For example, an expression like '3 < x <= 5' is meaningless and is an error. Instead, you can use '3 < x && x <= 5'.

Examples:
2 == 1, 3 != 3, 3 < 1, T == F, and T > 1 all have the value False
2 == 2, 3 != 2, 3 > -1, F == F, and 0 == F all have value True
"A" > "a", "A" != "A", and "MacAnova 2" == "MacAnova 3" are all False
F && T, F || F, and !T all have the value False
F || T and !F have the value True

In contrast with the C programming language, both expressions that are combined with '&&' and '||' are always evaluated regardless of the value of the first expression. Thus in

```
   (sqrt(2) < sqrt(3)) || (log(5) < log(6))
```
both log(5) and log(6) are evaluated although the final value of the expression (True) could have been determined once it was found that sqrt(2) < sqrt(3) was True. Pre-defined macros alltrue and anytrue provide the C behavior. For example
```
  Cmd> anytrue(sqrt(2) < sqrt(3), log(5) < log(6))
```
evaluates only its first argument.

MacAnova allows comparison of or logical combination of arrays of different sizes entirely analogously to the way they can be combined arithmetically by '+', '-', '*', '/', '^', and '%%'. For instance, '2 < run(3)' is vector(F,F,T). See arithmetic for details.

When one of the operands is a structure, each of its components is combined with the other argument, producing a structure with the same shape as the structure argument. If both arguments are structures, they must have the same shape and the corresponding components are combined. In either case, all the components of a structure must have the same type and all components must be compatible. See structures.

On the Macintosh, you can use 'Option+<', 'Option+>' and 'Option+=' in place of '<=', '>=', and '!=', respectively.

Caution: Because '<-' is the assignment operator, 'a <- 5' will always be interpreted as "assign 5 to a" even if what is wanted is a comparison of a with -5. To obtain the latter a space must follow '<' as in 'a < -5'.

2.169 logistic

Keywords: glm, regression, categorical data
Usage: logistic([Model],n:Denom [, incr:T, offsets:vec, print:F or silent:T, pvals:T, maxit:m, epsilon:eps, coefs:F]), Denom REAL scalar or vector > 0, vec a REAL vector, m an integer > 0, eps REAL > 0

logistic(Model,n:Denom) computes a logistic regression fit of the model specified in the CHARACTER variable Model. If y is the response variable in the model it must consist of integers y[i] >= 0. Denom must either be an integer scalar >= max(y) or a REAL vector of the same length as y with Denom[i] >= y[i]. Estimation is by maximum likelihood on the assumption that y[i] is binomial with Denom[i] trials (Denom trials for scalar DENOM).

If either Denom or y contains non-integer values a warning message is printed.

See topic 'models' for information on specifying Model.

logistic() sets the side effect variables RESIDUALS, WTDRESIDUALS, SS,

DF, HII, DEPVNAME, TERMNAMES, and STRMODEL.

If, say, Model is "y=x1+x2", an iterative algorithm is used to predict logit(E[y/Denom]) as a linear function of x1 and x2, where logit(p) = log(p/(1-p)). A two line Analysis of Deviance table is printed.

Line 1 is the difference 2*L(1) - 2*L(0), where L(0) is the log likelihood for a model with all coefficients 0 and L(1) is the maximized log likelihood for the model fit.

Line 2 is 2*L(2) - 2*L(1) where L(2) is the maximized log likelihood under a model fitting one parameter for every y[i]. Under certain conditions, the latter can be used to test the goodness of fit of the model using a chi-squared test.

logistic(Model,n:Denom,inc:T) computes the full logistic model and all partial models -- only a constant term, the constant and the first term, and so on. It prints an Analysis of Deviance table, with one line for each term, representing a difference 2*L(i) - 2*L(i-1) where L(i) is the maximumized log likely for a model including terms 1 through i, plus the deviance of the complete model labeled as "ERROR1". Each line except the last can be used in a chi-squared test to test the significance of the term on the assumption that the true model includes no later terms.

If you omit Model (logistic(,n:Denom ...)), the model from the most recent GLM command such as poisson() or anova(), or the model in CHARACTER variable STRMODEL is used.

Computations are carried out using iteratively reweighted least squares.

Other keyword phrases

Keyword phrase	Default	Meaning
maxiter:m	50	Positive integer m is the maximum number of iterations that will be allowed in fitting
epsilon:eps	1e-6	Small positive REAL specifying relative error in objective function (2*log likelihood) required to end iteration
print:F	T	Suppress all output except warning and error messages. Side effect variables are set.
silent:T	F	Suppress all output except error messages. Side effect variables are set.
coefs:F	T	Suppresses computation of quantities necessary for coefs() and secoefs() to compute estimated coefficients and their standard errors. Thus it effectively disables coefs() and secoefs() as well as some of the options available in modelinfo().
offsets:OffVec	none	Causes model to be fit to logit(p) to be

1*Offvec+Model, where OffVec is a REAL vector the same length as response y. Note OffVec is in logit units.

The default value for pvals can be changed by setoptions(pvals:T). See setoptions().

When OffVec a linear combination of X-variables in the model with specified coefficients, the overall model deviance from logistic(Model,n:Denom, offsets:OffVec) tests the hypothesis that the true coefficients are the same as the specified ones. Thus, if the elements of y are independent binomial random variables with n = 10, the error deviance from

```
logistic("y=1",10,offsets:rep(log(p0/(1-p0)),length(y)))
```

is a test statistic for H0: elements of y/10 have identical expectation p0.

logistic(Model,n:Denom,...) is equivalent to glmfit(Model,n:Denom, dist:"binomial", link:"logit",...).

2.170 macintosh

Keywords: general

Usage: Type help(macintosh) for a summary of features special to the Macintosh version of MacAnova.

Summary of features special to the Macintosh version.

Windows

There can be up to nine command/output windows. Everything you type and all character output goes in the frontmost one. You can create a new window using New Window on the Windows menu or Open on the File menu. The latter loads the window with the contents of the file selected. You can switch between windows, close them or hide them using the Windows menu. You can print all of a window or a selection from a window by selecting Print Window or Print Selection on the File menu. You can save their contents as files on disk by selecting Save Window or Save Window As on the File menu.

There are eight high resolution graphics windows plus two panel windows, Panel 1-4 and Panel 5-8, which display the other graph windows in miniature. You can switch between graphics windows, close them or hide them using the Windows menu. Any graphics window including the panel windows can be printed using Print Graph... on the File menu or copied to the clipboard for pasting into the Scrapbook or other application using Copy on the Edit menu. You can direct a plot to graph window i by plotting keyword phrase 'window:i', where 1 <= i <= 8. 'window:0' puts the graph in the window most recently written in.

You can display any graph window by pressing Command+1, Command+2, ..., Command+8 or Command+F1, ..., Command+F8, and display panel windows by

Command+G. Command+G also switches to the other panel window when a panel window is displayed.

All windows have a close box in the upper left corner, a resizing box in the lower right corner, and a zoom box in the upper right corner. Output windows also have a scroll bar for moving backward and forward through output.

Executing a Command

MacAnova recognizes a command as being ready to execute only if the cursor is at the end of the line when you hit Return, or if you hit Enter (see next paragraph). Of course, as in all versions, the line will not be accepted as complete if there is any '{' unmatched by a corresponding '}' or if there is an unfinished quoted string started by '"'.

The Enter key is not equivalent to Return but might be considered an "execute" key. It behaves differently depending on whether you have selected text before the prompt in the output window. Command+Return or Shift+Return are both equivalent to Enter.

(a) If nothing is selected with the mouse in the output window, Enter is equivalent to moving the cursor to the end of the command line and pressing Return. This causes the command on that line to be executed unless there is an incomplete quoted string or unbalanced curly brackets ({...}). This is particularly useful when editing a command about to be executed. No matter where in the line the cursor is, Enter causes the edited command to be executed, while Return does not initiate execution unless the cursor is at the end of the line

(b) If you have used the mouse to select text before the current prompt, Enter causes it to be copied to the end of the current command line, followed by Return. Except when there are unbalanced curly brackets or an incomplete quoted string, the current command line is then executed. This makes for very easy re-execution of commands, possibly after editing them in place.

Item Copy and Execute (or Execute, when nothing is selected in the output window) on the Edit menu does the same thing as Enter, as does pressing Command+\ or key F6.

History of Previous Commands

Typed commands are automatically saved in an internal "history" list. You can move through this list, inserting previous commands after the prompt, using items Up History and Down History on the Edit menu. The number of commands saved is determined by option 'history' whose default value is 50. See setoptions().

Pressing F7 or the key combination Option+up-arrow is equivalent to selecting menu item Up History. Pressing F8 or Option+down-arrow is equivalent to selecting Down History.

Command+Option+up-arrow retrieves the oldest saved command.

Command+Option+down-arrow reinserts whatever you had originally typed, if anything, before recalling earlier commands.

By default, MacAnova saves the most recent 50 lines. To change this, say, to 100, type 'setoptions(history:100)'.

Moving Around the Window

Arrow keys move the cursor as you would expect. Command+B (Back) and Command+F (Forward) are equivalent to the left- and right-arrow keys. There is no equivalent to the up- and down-arrow keys.

Pressing the Option key while pressing a left- or right-arrow key moves backward or forward one "name" (stretch of characters that are legal in names). Pressing the Option key while pressing an up- or down-arrow key inserts previous commands at the prompt similar to items Up History and Down History on the Edit menu.

Pressing the Command key while pressing an arrow key moves the cursor to the start or end of the line (left- or right-arrow) or the top or bottom of the window (up- or down-arrow). Repeated use of Command+up or Command+down-arrow scrolls through the window, moving the cursor as it goes.

Pressing the Shift key while using an arrow key selects whatever is between where you start and where you end.

Command+A (Go To Prompt on the Windows menu) moves the cursor to the start of the current command line, just after the current prompt.

Command+E (Go To End on the Windows menu) or End on suitable keyboards moves it to the end of the current command line at the very end of window.

Command+T (Scroll To Top on the Windows menu) or Home scrolls to the Top of the current command/output window. Command+U (or Page Up) scroools back a screenful. Command+D (or Page Down) scrolls forward a screenful. These do not move the cursor.

Specifying File Names

When you use "" as the file name in any command requiring one (for example vecread("")), the usual Macintosh scrolling dialog box lets you select the file. You can also use an explicit file or "path" name. In the latter case, if the name does not contain ':' (which would identify it as a "path" name), MacAnova will look for it first in the default Folder (see topic Files) and then in the Folders specified in pre-defined CHARACTER vector DATAPATHS. See adddatapath, customize.

Help

Selecting item Help on the Apple menu is equivalent to typing 'help()'. If a word, say 'matrix', is selected in the window, item Help is equivalent to 'help("matrix")'. Command+H or the Help key does the same.

Interrupting

Command+. (period) or Command+I may be used to interrupt an operation or output. Depending on the operation it may not be recognized immediately.

File Menu

Open (Command+O) creates a new output window and reads a file into it. It might, for example contain output from a previous MacAnova session.

Save (Command+S) and Save As write the current command/output window to a file.

Page Setup and Print/Print Selection/Print Graph (Command+P) do what you would think they should do.

Interrupt (Command+I) is equivalent to pressing Command+. (period) (see above).

Go On resumes computing after a graphing command with keyword phrase 'pause:T'. See topic graph keys.

Save Workspace (Command+K) and Save Workspace As invoke save() (asciisave() if asciisave() was used previously). See save() and asciisave().

Open Batch File (Command+Option+O) is equivalent to 'batch("")'. See batch().

Spool Output to File (Command+Option+S) is equivalent to 'spool("")'. If a spool file has previously been specified this menu item will be Stop Spooling or Resume Spooling and is equivalent to 'spool()'.

Edit Menu

In the output/command window you can use Undo/Redo (Command+Z) , Cut (Command+X), Copy (Command+C), and Paste (Command+V) in the usual way. For a graph window only Copy is active.

Copy to End (Command+/) copies the current selection to the end of the command line without putting it on the Clipboard.

Copy and Execute/Execute (Command+\) is equivalent to pressing Enter (see above) in the output command window.

Up History inserts the previously typed command after the prompt. Repeated selection of Up History successively inserts older and older commands. It is equivalent to pressing the Option and up-arrow keys.

Down History moves forward through previously saved commands, inserting them after the prompt. It is equivalent to pressing the Option and down-arrow keys.

Function keys F1, F2, F3, F4, F5, F6, F7 and F8 are equivalent to Undo/Redo, Cut, Copy, Paste, Copy To End, Execute (Copy and Execute), Up History and Down History, respectively. Enter, Shift+Return and Command+

Return are additional keyboard short cuts for Execute.

Windows Menu

This menu allows you to close or hide a window (Close and Hide), create a new output/command window (New Window or Command+N), select a graph window (Graph 1, Graph 2, ... Graph 8, Panel 1-4 and Panel 5-8), select an output command window by name, and move around the output/command window (Scroll To Top, Go To End, Go To Prompt, Page Up, Page Down).

Command Menu

The last 8 items are pre-defioned commands that are inserted in the output/command window. You can also select them by pressing Command+ +Option+1, ..., Command+Option+8.

Edit Commands... allows you to edit or replace any or all of the 8 commands. Since each command can be a macro, this allows a great deal of flexibility.

Options Menu

Significant Digits, Output Formats, Random # Seeds, Angle Units, GLM Options, Batch Options and Other Options allow you to set many of the items that can be changed by setoptions().

Font Menu

Size allows you to change the font size of the text in the current output/command window.

The remaining items are the names of fonts you can select for the text in the current output/command window. You should probably restrict your choices to non-proportional (equal character width) fonts such as Monaco or Courier. The default font is McAOVMonaco 9, a modified form of Monaco 9. Courier 18 may be preferable for use with an overhead projector. You can also change fonts using setoptions() using keywords 'font' and 'fontsize'.

Other Information

MacAnova can "background", that is it will continue running when you switch to another application under System 7 or when you are using Multifinder under earlier systems.

Option+'<', Option+'>' and Option+'=' are recognized as equivalent to '<=', '>=', and '!=', respectively. Option+'\' and Option+'|' (Option+Shift-'\') are recognized as equivalent to '<<' and '>>', respectively.

The shell() command and special treatment of lines starting with '!' is not available on the Macintosh. Use Multifinder, System 7, or a Desk Accessory instead.

Files produced by save() in all Macintosh versions 3.xx and 4.xx, but not 2.xx, can be restored. However, files produced by asciisave() should restore correctly.

Color is not supported under MacAnova except in so far as the system provides it automatically for all applications.

2.171 macro

Keywords: macros, control, syntax
Usage: macro(text [, dollars:T]), text a CHARACTER scalar

macro(Text) creates a macro from the commands contained in the CHARACTER variable or string Text. Text can also be an exisiting macro.

macro(Text,dollars:T) creates a macro from Text, adding "$$" to every temporary name (name starting with '@') that does not already end in "$$". When the macro is executed, "$$" is replaced by a 2 digit number unique to the particular macro invocation so that the actual name is unique to the macro. The length of any temporary name you use must be no more than 10 characters, counting '@'. You can ensure that "$$" is appended to temporary variables in an existing macro myMacro by

```
Cmd> myMacro <- macro(myMacro, dollars:T)
```

macro(Text,inline:F [,dollars:T]) marks the macro being created as one to be expanded out-of-line instead of having the expansion inserted into the input line. This primarily means that the macro will be freshly expanded every time it is encountered, even on multiple trips through a loop. An instance of an in-line macro is expanded just once. See macros. inline:T marks the macro as to be expanded using the default expansion mode specified by the value of option 'inline'. See setoptions(). You can mark an existing macro myMacro to be expanded out-of-line by

```
Cmd> myMacro <- macro(myMacro, inline:F)
```

When you are using macro() to define a macro, you should type any quotes and backshlashes ('"' and '\') in the macro as '\"' and '\\'. Thus, in particular, quoted quotes in a macro must be typed as '\\\"'.

Within quotes ("..."), any characters other than "\n" and "\t" whose ASCII codes are either 127 or less than 32, are replaced by their escaped octal representation of form \xxx. Non-standard characters with ASCII codes >= 128 are not escaped. Thus

```
Cmd> myplot <- macro("chplot($1,$2,\"\001\",$K)")
```

and

```
Cmd> myplot <- macro("chplot($1,$2,\"\\001\",$K)")
```

are completely equivalent. If 001 were replaced 301 (ASCII 129), the first would produce a macro containing the character with this code (which has a computer specific interpretation) and the second would produce a macro not containing this character.

See macros for details on writing macros, including the use of special symbols '$0, '$1', '$2', ..., '$N', '$V', '$v', '$K', '$k', '$S' and '$$'.

```
Examples:
  Cmd> median <- macro("describe($1,median:T)")
  Cmd> myread <- macro("matrix(vecread(\"$1\"),$2)'") #note transpose
  Cmd> greetings <- macro("print(\"\\\"Hello\\\"\")")
  Cmd> xlogx1 <- macro("@x <- 1;@x*log(@x)")
  Cmd> xlogx2 <- macro("@x$$ <- 1;@x$$*log(@x$$)")
  Cmd> xlogx3 <- macro("@x <- $1;@x*log(@x)", dollars:T)

median(x) would compute the medians of the columns of x.  See
describe().

y <- myread(family.dat,23) would create a matrix with 23 columns from
data from file family.dat, assumed to have a total of n*23 data items
arranged in n rows.

greetings() prints "Hello", complete with the quotation marks.

xlogx1(x), xlogx2(x) and xlogx3 all compute x*log(x).  However, use of
xlogx1 might lead to a problem if some other macro also used @x as a
temporary variable.  xlogx2 avoids this problem because the temporary
variable name @x$$ will be expanded to @x01, say, a name that should not
conflict with any other temporary name.  xlogx3 is identical to xlogx2
because each instance of @x is converted to @x$$.

See also macros, macroread(), macrowrite().
```

2.172 macro files

Keywords: macros, files, input, output
Usage: Type help(macro files) for information on the file format expected by macroread().

```
This topic describes the format of a file to be read by macroread().
See topic data files for information on files to be read by vecread(),
readcols and matread(), and topic files for more technical information
on file names, default directories or folders, and abbreviated file
names of the form "~/filename".

A file that can be read by macroread() must be a plain text or ascii
file.  If you create it in a word processor, be sure to save it as a
text or ascii file.  On the file can be any number of macros, each with
a header line, followed by optional comment lines and macro text lines.

There are two possible forms for macros that can be read by macroread()

Form A, requiring a count of the lines
   Name Nlines MACRO
   ) 0 or more descriptive comment lines starting with ')'
   ) .......
   Line 1 of macro
   . . . . .
```

```
   Line Nlines of macro

and form B, terminated by a special line
   Name MACRO
   ) 0 or more descriptive comment lines starting with ')'
   ) .......
   Line 1 of macro
   . . . . .
   Line Nlines of macro
   %Name%
```

Name is the macro name to be searched for. In form A, Nlines is an integer specifying the number of lines in the macro, not counting the descriptive comment lines, if any.

If the name line is of the form

```
   Name MACRO Nlines OUTOFLINE
```

or

```
   Name MACRO OUTOFLINE
```

the macro will be marked so that it will always be expanded out-of-line. If INLINE appears instead of OUTOFLINE, it will not be so marked. Simply OUT or IN can be used instead of OUTOFLINE or INLINE.

The text of the macro immediately follows the header and comment lines. In form A, there must be exactly Nlines. The count includes lines starting with "#".

In form B, the line immediatly after the last line of macro text must consist of the macro name preceded and followed by "%".

It is good practice to include both header comment lines starting with ")" and macro comment lines starting with "#" describing the usage of the macro. See also macrousage().

It is permissible for there to be no lines of macro text (Nlines = 0 for Form A). When macroread() reads such a "macro", only the descriptive comments are printed (if 'quiet:T' is not present). It is a useful convention to make the first macro on a file have 0 length with the header describing the contents of the file. Then 'macroread(FileName)' will print the header for the empty macro.

Example of macro file containing a 0 length information only macro and 3 genuine macros:

```
   info      0 MACRO
   ) This file contains macros median, rm, and means

   median    2  MACRO
   ) median(x) computes medians of columns of x
   # usage: median(x) [it's helpful to include usage in body of macro]
   describe($1,median:T)

   rm        2 MACRO
```

```
) rm(a,b,...) alias for delete for Unix lovers
# usage: rm(a,b,...) equivalent to delete(a,b,...)
delete($0)

means      MACRO
) means(x1,x2,...) computes means of vector arguments
# usage: means(x1,x2,...) computes means of vector arguments
@args$$ <- structure($0) # make structure of all arguments
for(@i$$,1,$N){
    if(!isvector(@args$$[@i$$])){
        error("ERROR: Arguments to macro $S must be vectors")
    }
}
@result$$ <- rep(0,$N) #create vector of right length
for(@i$$,1,$N){
    @result$$[@i$$] <- sum(@args$$[@i$$])/nrows(@args$$[@i$$])
}
@result$$
%means%
```

Note: Form B is new with Version 4.04 of MacAnova.

2.173 macro syntax

Keywords: macros, syntax, control
Usage: Type 'help(macro syntax)' for information on how to write macros.

This topic presumes familiarity with topic 'macros'. It describes the use of comments in macros, how values are returned, the special symbols that may be used in macros and gives some tips on writing robust and efficient macros.

Macro Self-Documentation

Often the first few lines of a macro are comments ("#" at the start of the line) which describe how the macro is used. You can use command macrousage() to print such comment lines. When you write a macro, it is good practice to include such lines.

The Value of a Macro

The last expression appearing in a macro is returned as its value. If this expression is the name of an "invisible" variable (name starting with '_' or '@_') the macro value can be assigned or used in an expression but will not be printed. If no value is to be returned, the macro should end with ';;'.

Referencing macro arguments

In the text of a macro the place holders '$1', '$2', ... are used to refer to argument 1, argument 2, ... When the macro is executed, MacAnova examines its text and substitutes the appropriate argument, almost exactly as typed, for any such place holder. It then executes

the commands as if they had been typed at the keyboard. Thus, if the first argument to a macro is '3+4', '@a <- $1' gets expanded to '@a <- 3+4'.

If a macro references a missing argument (for example, $3 when there are only two arguments) it usually terminates immediately with an error message. If, however, a missing argument appears in a quoted string ('print("$3")' when there are only two arguments) it is "expanded" to nothing ('print("")'). If you use place holders of the form '$01', '$02', ..., with a leading 0 for the number, then a missing argument is expanded to NULL outside of a quoted string. This may or may not result in an error message if the argument is used, but you can test the value using one or more of the isxxxx() functions (see below).

If an argument which is itself a quoted string, possibly containing backslashes ('\'), is referenced in the macro as part of a quoted string, then all instances of '"' and '\' in the argument are changed to '\"' and '\\'. Thus, for example, in a macro, 'print("$1")' expands to 'print("3+4")' if the first argument is '3+4' but expands to 'print("\"foo\"")' if the first argument is '"foo"'.

Use of Temporary Variables

It is good practice to use temporary variables (names starting with '@') in macros so they will be deleted automatically just before the next prompt. You can also append '$$' to names to create a name that should be unique to the macro (see below). If you use 'dollars:T' as an additional argument when using macro() to create a macro, '$$' will be automatically appended to all temporary names not already ending in '$$'.

Because temporary names are not deleted until the next prompt, it is often a good idea to delete them before exiting a macro, except for a result being returned, of course. To delete a variable, say '@result', whose value is to be returned, the last command in the macro should be delete(@result,return:T). @result will be deleted but its value will still be returned. See delete().

Occasionally you may want to create a non-temporary "invisible" variable, that is, one that is not normally listed by list() or listbrief(). You do this by assigning a name starting with '_' such as '_x'. See also list() and listbrief(). A temporary variable is "invisible" if its name starts with '@_', say '@_x'.

Expansion of other special symbols containing '$'

In a macro, certain symbols starting with '$' have a special meaning. Although they would be errors outside a macro, in a macro these symbols are "expanded", that is, something is substituted for them in the text of the macro. In brief, these symbols are as follows:

Symbol	Expanded value
$0	The complete comma-separated list of macro arguments.
$N	The number of macro arguments = 1 + (number of separating commas).

```
  $V        A comma separated list of all macro arguments that are not
            keyword phrases.
  $v        The number of macro arguments that are not keyword phrases.
  $K        A comma separated list of all macro arguments that are
            keyword phrases.
  $k        The number of macro arguments that are keyword phrases.
  $A        A CHARACTER vector whose elements are the character strings
            specifying the arguments, that is, vector("$1","$2",...).
  $S        The name of the macro.
  $$        Expands to a unique 2 digit number, 00, 01, ..., 49 in out-
            of-line macros and 50, 51, ..., 99 in in line macros.

$N, $V, $v, $K, $k, $S and $A are not treated specially if they are
either preceded or followed by a character that could be part of a
variable name.  Thus str$N would be interpreted as component N of
structure str (see structures) and $Na would not be changed and would
probably result in an error.

Here are fuller explanations or examples of the use of these special
symbols.  In all the following we assume the symbols are in a macro
invoked by mymacro(x, title:"Hello Dolly",run(5),new:T).

$0:
  In the example $0 is replaced by 'x,title:"Hello Dolly",run(5),new:T'.
  $0 is particularly useful in defining "aliases" of standard commands.
  For example,
   Cmd> dir <- macro("list($0)")
  defines macro dir which is used identically to list().

  If $0 appears in a quoted string, all instances of '"' or '\' in any
  macro argument are prefixed by '\'.  Thus, in the example, "$0" is
  replaced by "x, title:\"Hello Dolly\",run(5),new:T".

$N:
  In the example, $N is replaced by '4'.

$V:
  In the example, $V is replaced by 'x,run(5)'.  If $V is used within
  double quotes, any instances of '"' or '\' are expanded as for $0.

$v:
  In the example, $v is replaced by '2'.

$K:
  In the example, $K is replaced by 'title:"Hello Dolly",new:T'.  This
  is useful for passing on all the keywords to a function invoked by the
  macro.  See pre-defined macro colplot for an example of the use of $K.
  If $K is used within double quotes, any instances of '"' or '\' are
  expanded as for $0.

$k:
  In the example, $k is replaced by '2'.
```

$A:
In the example $A is replaced by vector("x","title:\"Hello Dolly\"", "run(5)","new:T"). To print argument 3 to a macro as it appeared in the call, instead of print("$3"), you might use print($A[3]). In the example this would print 'run(5)'. Any instances of '"' or '\' in the arguments are replaced by '\"' and '\\'. If $A is part of a quoted string, it is not expanded.

$S:
In the example, $S is replaced by 'mymacro' whether or not it is in quotes.

$$:
The particular integer that is substituted for $$ remains the same throughout an invocation of the macro, but is incremented once for each macro in which $$ is used. Its principal usefulness is in creating temporary variable names that are specific to a specific execution of a macro. Thus '@A$$ <- 3' might become '@A52 <- 3' or '@A57 <- 3' depending on when the macro is executed. This makes it possible to write macros whose temporary variables have names that are different from names used in other macros. You should avoid using temporary names ending in two digits to avoid possible conflicts with names generated by $$.

In a macro expanded in-line, the value of $$ starts at 50. If it reaches 100, all macros abort, thus putting a limit on the depth to which macros can be nested or recursive macros can call themselves. In a macro expanded out-of-line, the value of $$ ranges from 0 to 49. In this case all macros abort if it reaches 50.

Note that each invocation of a macro expanded in-line in a loop will have the same value for $$ since it is expanded only once, the first time through the loop. This will usually be the case for macros expanded out-of-line, too.

Recursive use of macros

A macro may invoke itself directly or indirectly up to a maximum depth of 50, provided some test is included to avoid infinite recursion. In practice, current limitations of the parser limit the maximum depth to around 20. If greater depth is needed, you may be able to accomplish it using evaluate().

Tips for good macro writing

Because macro arguments are expanded literally, their place holders should usually be enclosed in parenthesis when used in expressions. Thus '($1)*($2)' is preferable to '$1*$2'.

If a macro argument is referred to more than once in a macro it should usually first be copied to a temporary variable. Thus a macro to compute a mean might best be defined by

```
Cmd> mean <- macro("@x$$ <- $1;sum(@x$$)/dim(@x$$)[1]")
```

rather than by the somewhat simpler

```
Cmd> mean <-  macro("sum($1)/dim($1)[1]")
```

If the latter definition is used, 'mean(boxcox(x[,vector(1,2,4)],.5))' would result in boxcox(x[,vector(1,2,4)],.5) being evaluated twice.

When a macro is not intended to return a value, end it with ';;'. For arcane reasons, this helps MacAnova allocate memory more efficiently.

Functions useful in macros

There are several of functions whose primary usefulness is within a macro. These include anymissing(), isdefined(), ismatrix(), isscalar(), isvector(), isfactor(), isreal(), ischar(), islogic(), isgraph(), and isstruc(), all of which are useful in checking whether an argument is suitable. Function keyvalue() is designed to allow you to interpret keyword phrases in the argument list. Functions modelvars(), modelinfo() varnames(), xvariables() and xrows() allow a macro to do sophisticated computations based on the results of a previous GLM command.

Here is a fragment that might be in a macro whose first argument should be a REAL scalar and second argument should a CHARACTER vector, and which should recognize keywords 'down' with T or F for value and 'power' with positive REAL scalar value. 'down' is optional with default F and 'power' is required.

```
if(!isscalar($1,real:T)){error("ERROR: $1 is not a REAL scalar")}
if(!isvector($2,char:T)){error("ERROR: $2 is not a CHARACTER vector")}
@down <- keyvalue($K,"down","logic")
@down <- if(isnull(@down)){F}else{@down}
@power <- keyvalue($K,"power","real")
if (isnull(@power)){error("ERROR: keyword 'power' is required by $S")}
@ok <- if (!isscalar(@power,real:T)){F}else{@power > 0}
if (!@ok){error("ERROR: value for 'power' not REAL scalar > 0")}
....
```

Note that if the value of a keyword doesn't have the specified type, keyvalue() generates an error which immediately terminates the macro.

See macroread() for examples of macros.

2.174 macroread

Keywords: macros, files, input

Usage: mymacro <- macroread(fileName,macroName [,quiet:T or F, echo:T or F, silent:T, notfoundok:T]), fileName and macroName CHARACTER scalars; fileName can also be of the form string:charVal, charVal a CHARACTER scalar or vector

mymacro <- macroread(FileName,macroName) searches the named file for a macro whose name matches macroName. If the macro is found, it is read and made available under the name 'aMacro'. FileName and macroName must be quoted strings or CHARACTER scalars. Most usually the name to the left of '<-' will be the same as the name of the macro read. The header

and comment lines are echoed to output. See topic macro files for a description of the required file format.

mymacro <- macroread(FileName) with no macro name reads the first macro on the file. The first non-empty, non-blank line in the file is assumed to be the start of the macro as described in topic macro files.

In a version with windows (Macintosh, Windows, Motif), when FileName is the null string "", you will be able to select the file using a dialog box.

Just reading a macro does not make it available; you must assign the value of macroread() using '<-'.

When you are reading a macro from one of the standard macro files (macanova.mac, design.mac and tser.mac) pre-defined macro getmacros is more convenient to use than macroread(). An example of its use is

```
Cmd> getmacros(covar)
```

This replaces to 'covar <- macroread("macanova.mac","covar")', except that you don't even need to know which file 'covar' is located in.

See below for discussion of keywords 'quiet', 'echo', 'silent' and 'notfoundok'.

If 'OUTOFLINE' or simply 'OUT' appears on the first header line of the macro, it will be marked to be always expanded out-of-line. Otherwise, the expansion of the macro will be determined by the value of option 'inline'. See macros, setoptions().

macroread(CONSOLE) reads from the regular input stream allowing you to type in the macro using the format described under topic macro files. On the Macintosh, type one line at a time in the dialog box that is opened. On Unix and DOS, type the necessary lines after the prompt. The value of CONSOLE is ignored. The first line must be of the form 'Name nLines MACRO', where nLines is the number of lines in the macro. On any machine, when macroread(CONSOLE) is used in a batch file, it reads the macro from the lines immediately following the macroread() command. See batch().

Any blank lines at the end of a macro are trimmed off when it is read.

There are 4 keywords, 'quiet', 'echo', 'silent' and 'notfoundok' which control what will be printed by macroread().

Keyword phrase	Meaning
quiet:T	Header and descriptive comments will not be printed
quiet:F	All header and descriptive comments will be printed
echo:T	Lines of the macro itself will be printed as they are read
silent:T	Only error messages will be printed; incompatible with quiet:F or echo:T

notfoundok:T Failure to find themacro is not considered an error so no error message is printed.

Without quiet:T or quiet:F, the header and comment lines not starting with '))' preceding the macro will be echoed to output.

Even without echo:T, header lines are printed when FileName is CONSOLE and the macroread() command is in a batch file. (In windowed versions, a macro will be echoed if FileName is CONSOLE whether or not the command is in a batch file.) Such echoing can be suppressed by 'echo:F'.

When notfoundok:T is an argument and the macro is not found, macroread() returns NULL as value. When used in a macro, this feature allows special action if macroName is not found.

macroread(file:FileName,...) is equivalent to macroread(FileName,...).

macroread(string:CharVar,...) where CharVec is a CHARACTER scalar or vector, does not read from a file. Instead, it "reads" CharVar as if each element were a line (or several lines if there are embedded end-of-line characters) read from a file. The first element or line of CharVar must be a header line with a name and number of lines. In particular, mymacro <- macroread(string:CLIPBOARD) would read the first macro on a replica of a data file in the special variable CLIPBOARD. In the Macintosh, Windows and Motif versions this would be taken from the Clipboard. In the Motif version, you can also "read" from special variable SELECTION in a similar way. See topic clipboard.

If either keyword 'file' or 'string' is used, they can appear in any position in the argument list, as can setName which must be the only non-keyword argument. For example, Cmd>
macroread(quiet:T,"mymacro",file:"myfile.dat") is equivalent to Cmd>
macroread("myfile.dat","mymacro", quiet:T).

A predefined CHARACTER variable MACROFILES contains the names of files containing macros. A pre-defined macro getmacros allows easy retrieval of macros from the files whose names are in MACROFILES. At startup, MACROFILES is initialized to vector("macanova.mac", "tser.mac", "design.mac"), but you can change it if desired. See getmacros and addmacrofile.

See also macros, macro(), read(), matread(), inforead(), macro files, files

2.175 macros

Keywords: macros, control, syntax

Usage:
```
mymacro <- macro(charVar [, dollars:T])
mymacro <- macroread(fileName [,"mymacro"])
getmacros(macro1 [, macro2 ...])  (reads macro from one
of files specified in MACROFILES)
macrowrite(fileName, mymacro)
```

A macro is a collection of commands grouped together to make it easy to execute them all at once. It is used (invoked) the same way as a function, by typing its name followed by 0 or more arguments in parentheses. For example, y <- boxcox(x,.5) invokes macro boxcox.

By default, when a macro is invoked it is expanded in-line, that is, its arguments are literally substituted into the text of the macro and the modified text is inserted in the line being executed exactly as if it had been typed in place of the macro call.

Because an in-line macro is inserted directly in the line, a particular instance of a macro is expanded only once, even if it is repeatedly encountered during a loop (see 'for' and 'while'). This makes it impossible to redefine an in-line macro in a loop and execute the new version the next time through. However, a macro that is expanded out-of-line (see below) is expanded every time it is encountered, allowing meaningful redefinition within a loop.

Macros may be read from a file by macroread() or created directly using function macro(). There also many pre-defined macros such as readcols and boxcox. Macros may be written to a file by macrowrite().

Macros are stored in the MacAnova workspace as variables similar to CHARACTER scalars. You can print the text of a macro by typing its name.

A macro can be a component of a structure (see structures) although it must be extracted in order to be used. Thus if structure boxcoxstr was created by boxcoxstr <- structure(boxcox), boxcoxstr$boxcox(x,.5) is illegal. You would have to use something like @tmp <- boxcoxstr$boxcox;@tmp(x,.5).

An alternative mode of macro expansion is out-of-line. In this mode, a modified copy of the macro text is created, substituting macro arguments in the text. This is then executed without being inserted in the line being executed. In a loop, a particular instance of an out-of-line macro will be expanded every time through the loop. The value of option 'inline' (default value is True) determines the default expansion mode. In addition a macro can be marked always to be expanded out-of-line by keyword phrase inline:F on macro(). See setoptions() and macro().

See also addmacrofile, getmacros, macro(), macrowrite(), macrousage(), macro files, macro syntax.

See macroread() for examples of macros.

2.176 macrousage

Keywords: macros, general
Usage: macrousage(Macro1 [,Macro2, ...]), Macro1, Macro2, ... , currently defined macros

macrousage(Macro) prints all comment lines (lines that start with "#") in Macro which must be a macro. These usally describe the usage of the macro, but that may not always be the case.

macrousage(Macro1, Macro2 ...) does the same for several macros.

macrousage(macroNames), where macroNames is a quoted string or CHARACTER vector specifying one or more macro names, prints the comment lines in each macro named.

Example:
```
  Cmd> macrousage(colplot, rowplot)
  Cmd> macrousage(listbrief(macros:T, keep:T)) # usage for all macros
```

See also help(), macro(), macros.

2.177 macrowrite

Keywords: macros, files, output
Usage: macrowrite(fileName, a, b, ... [, name:Name, header:F, comments:charVec, oldstyle:T]), a, b, ... macros, fileName and Name CHARACTER scalars, charVec a CHARACTER vector or scalar

macrowrite(FileName,a, b, ...) writes macros a, b, ... on the file. Filename must be a CHARACTER variable or quoted string and a, b, ... must be macros. a, b, ... are written in the form recognized by macroread. The default is to write each with no line count in the header and with a trailing line of the form %macroname%.

If FileName is variable CONSOLE or a CHARACTER variable whose value is "CONSOLE", the output is written to the screen rather than to a file. The value of variable CONSOLE is ignored.

If the macro was previously marked to be expanded out-of-line, "OUTOFLINE" is added to the header line,

Keyword 'new':
macrowrite(FileName, a, b, ..., new:T) removes all information currently in the file before writing new information. Without 'new:T', macros are

written at the end of the file.

Keyword 'comment':
macrowrite(FileName, a, comment:charVec) writes each element of CHARACTER vector or quoted string charVec, prefixed by ") ", as a comment line after the header. If header:F appears, no such comments are written. Keywords

Keyword 'header':
macrowrite(FileName,a,b,...,header:F) writes the macros without any header lines or any trailing %macroname%. They will not be readable by macroread(). Also, keyword 'comments' will be ignored.

Keyword 'oldstyle'
macrowrite(FileName, a, b, ..., oldstyle:T) writes each macro in the old style format. A line count will be included in the header line and no trailing %macroname% line will be written.

See also getmacros, macros, macro(), macroread(), matwrite(), matread(), macro files, files.

2.178 makecols

Keywords: combining variables
Usage: makecols(x,var1,var2, ...), where x is a REAL matrix, var1, var2, ... unquoted variable names
makecols(x,vector("var1","var2", ...))

makecols(x,name1,...,namek), where x is a REAL matrix and name1, ..., namek are unquoted names, creates new REAL vectors name1, name2, ... from the columns of x. Thus makecols is a sort of inverse to hconcat().

makecols(x,vector("name1","name2",...,"namek")) is an alternative usage.

If there are more names than columns of x, the extra names are ignored. If there are fewer names than columns, the extra columns are ignored.

Example:
```
  Cmd> makecols(x, x1, x2, x3, x4)
```
or
```
  Cmd> makecols(x, vector("x1","x2","x3","x4"))
```
puts the first 4 columns x in vectors x1, x2, x3 and x4.

makecols is implemented as a pre-defined macro.

See also readcols, hconcat().

2.179 makefactor

Keywords: glm, anova, character variables
Usage: `makefactor(vec), vec a REAL or CHARACTER vector`

makefactor(vec) computes a factor corresponding to the REAL or CHARACTER vector vec. If there are m unique values in vec, the output will be a factor with m levels in the same order as the values in vec.

Even if vec is REAL and consists of positive integers, it may be preferable to use makefactor rather than factor(), since the output from factor will not contain all m levels if max(vec) > m, or if some levels are missing in vec.

Examples:

```
  Cmd> makefactor(vector("A","A","B","C","B","D","C","D"))
```

and

```
  Cmd> makefactor(vector(1.3,1.3,2.6,3.9,2.6,5.2,3.9,5.2))
```

both produce the same vector as factor(vector(1,1,2,3,2,4,3,4)).

makefactor is implemented as a macro.

See also factor().

2.180 makestr

Keywords: structures, combining variables
Usage: `makestr(var1 [,var2,...,vark] [, KeyPhrases])`, where var1, var2, ... are arbitrary variables KeyPhrases can be compnames:Charvec, labels:lab, and silent:T, where Charvec and lab are CHARACTER scalars or vectors. Use structure() instead.

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

makestr() is identical to structure(). See structure() for more details on its use.

The use of makestr() is deprecated -- that is, it will continue to be available for the immediate future, but at some point may be disabled. Use structure() instead.

See also strconcat(), structures, keywords, changestr(), compnames().

2.181 manova

Keywords: glm, multivariate analysis, anova

Usage: manova([Model] [,print:F or silent:T, coefs:F, pvals:T, fstats:T, byvar:T, sssp:F or T]), Model a CHARACTER scalar

manova(Model) computes a MANOVA table of SS/SP (sums of squares and sums of products) matrices for the model in the CHARACTER variable Model. The response variable should be a matrix with rows as cases and columns the variables.

Type 'help(models)' for information on how to specify Model.

If the response is univariate (has only one column), manova() is equivalent to anova().

Normally, when each row of a SS/SP matrix will fit on a single line, all matrices are printed in their entirety. When a row would require more than one line, only the term names and the degrees of freedom are printed. This behavior can be modified by keywords 'sssp', 'byvar', 'fstats' and 'pvals'; see below. In any case the matrices are saved in the three-dimensional side effect array SS, with the first subscript indexing terms.

manova(Model,weights:Wts) does a weighted analysis. Wts must be a REAL vector with Wts[i] >= 0 and nrows(Wts) = nrows(response). The results are the same as if the i-th row of the response and all X-variables (variates and dummy variables and their products), including the constant vector were multiplied by sqrt(Wts[i]) and a least squares fit (without an intercept) computed.

manova() or manova(,weights:Wts) (no model supplied) uses the model used by the most recent GLM command such as manova(), anova(), or poisson(). See glm.

Unless marginal:T is an argument, SS/SP matrices are computed sequentially (so called SAS Type I quantities).

Side effect variables created are RESIDUALS, HII, DF, SS, DEPVNAME, TERMNAMES, and STRMODEL. When weights are specified, RESIDUALS = Response - Fitted and WTDRESIDUALS = sqrt(Wts)*RESIDUALS is an additional side effect vector. You should use WTDRESIDUALS rather than RESIDUALS in residual plots or other diagnostic procedures.

SS is a 3-dimensional array such that SS[j,,] is the sum of squares and products matrix for term j. If the appropriate error matrix for the k-th term is SS[j,,], the eigenvalues needed for several standard tests (Wilks, Roy, Pillai, Hotelling generalized T-squared) may be computed by releigenvals(SS[j,,],SS[k,,]) or you can compute some test statistics directly, for example,

```
  Cmd> T2 <- dferror*trace(solve(SS[k,,],SS[j,,])
```

or

```
  Cmd> lambda <- det(SS[j,,])/det(SS[k,,]+SS[j,,]).
```

Other Keywords

Keyword phrase	Default	Meaning
print:F	T	Suppress all output except warning and error messages. Side effect variables are set.
silent:T	F	Suppress all output except error messages. Side effect variables are set.
byvar:T	F	Computes a complete ANOVA table for each variable. The full SS/SP matrices are not printed although they are still available in array SS.
fstats:T	F	Prints F-statistics and P values. The denominator for each F is the mean square from the next following term with name "ERROR1", "ERROR2", ... If byvar:T is an argument, these are added to the ANOVA tables. If not, then for each term a table of SS, MS, F-statistics and P values for each variable is printed. The full SS/SP matrices are not printed but they are available in array SS.
pvals:T	F	Prints P values. If byvar:T is an argument, these are added to the ANOVA tables. If not, then for each term a table of SS, MS, and P values for each variable is printed. The full SS/SP matrices are not printed but are available in array SS.
sssp:T	none	Forces the printing of the full SS/SP matrices, even when each row would require more than one line. This option is ignored with any of fstats:T, pvals:T, or byvar:T.
sssp:F		Suppresses printing of the full SS/SP matrices, even if a row would fit on a single line. Only the term names and degrees of freedom are printed.
coefs:F	T	Suppresses the computation of coefficients and quantities needed for their standard errors. Use of coefs:F disables coefs(), secoefs() and some modelinfo() options. coefs:F is not legal with marginal:T.
marginal:T	F	Specifies that SS/SP matrices are computed marginally. When there are no empty cells, and sometimes when there are, the computed SS/SP are equivalent to SAS Type III SS/SP. marginal:T is not legal with coefs:F. See topic glm.

If byvar:T is an argument and option (not keyword) 'fstats' has value True, F-statistics will be printed unless 'fstats:F' is an argument. See setoptions() and glm.

If byvar:T is an argument and option 'pvals' has value True, P values will be printed unless 'pvals:F' is an argument. See setoptions() and glm.

Options (not keywords) 'fstats' and 'pvals' are ignored if byvar:T is not an argument.

Functions contrast(), coefs(), predtable(), and cellstats() work after a manova().

2.182 match

Keywords: ordering, variables, character variables
Usage: match(x,vec [,nomatch]), x REAL or CHARACTER, vec a vector of the same type as x and nomatch a REAL scalar

match(x,vec,noMatch) computes a vector, matrix, or array of the same size and shape as x by matching its elements to elements of vector vec. x may be either REAL or CHARACTER and vec must be the same type. Assuming for simplicity that x is a vector and y is the output vector, y[i] = j if x[i] == vec[j], y[i] = noMatch if x[i] does not match any element of vec, and y[i] is MISSING if x[i] is missing. It is an error for any values of vector vec to be MISSING.

If noMatch is omitted (match(x,vec)), its default value is length(vec)+1 and, when there are any non-matching elements, an advisory message is printed.

Examples:

```
match(vector(1.3,2.4,1.3,5,5.1),vector(2.4,1.3),-1) yields
  vector(2,1,2,-1)
match(vector(1.3,2.4,1.3,5,2.4,?),vector(2.4,1.3)) yields
  vector(2,1,2,3,1,MISSING)
match(vector("A","B","A","C","B"),vector("B","A")) yields
  vector(2,1,2,3,1)
a <- factor(match(x,sort(unique(x)))) transforms a REAL x to a factor
unique(x)[match(x,unique(x))] yields x when x is a vector.
match(scalarValue,vec,0) != 0 if and only scalarValue is in vec
```

See also unique().

2.183 matprint

Keywords: output, files, missing values

Usage: matprint(fileName, a, b, ... [, format:Fmt, nsig:n, sep:sepChar, quoted:T or bylines:T,missing:mVal, name:Name, comments:charVec, width:w, header:F, oldstyle:T]), a, b, ... arbitrary variables, Fmt, sepChar and Name CHARACTER scalars with sepChar only a single character, charVec a CHARACTER vector or scalar, mVal a REAL scalar, w >= 30 integer.

matprint(FileName,a,b,... [,new:T]) writes REAL, LOGICAL and CHARACTER variables a, b,... (scalars, vectors, matrices, or arrays) file FileName in a form which can be read by matread(). It can also write NULL variables and structures. GRAPH variables are legal arguments but are currently written as NULL variables.

matprint(a,b,...,file:FileName [,new:T]) is an alternative usage.

If new:T is present, anything already in the file is discarded before writing. Otherwise, writing is to the end of the file.

If width:w, with w an integer >= 30, is not an argument, the default value is taken from option 'width' (see setoptions()). This is the presumed line length and is used to determine the maximum number of values printed on one line.

For REAL and LOGICAL variables, by default matprint() uses the format that is used by print(), namely the format specified in option 'format'; this normally provides 5 significant digits in floating point form.

For CHARACTER variables, the default format, whenever possible, is "by fields", that is elements are written as fields separated by spaces. This is not feasible if there are any spaces or non-printable characters in the data. In that case, each element is quoted ("...").

Structures are written in a form that not only allows matread() to read all the components, but also can read individual components if desired. Since macros may be elements of structures, keyword phrase 'oldstyle:T' may be an argument. See macrowrite().

See data files for description of the file format.

matprint(FileName,Name1:a,Name2:b,...) gives names Name1, Name2,... to the data sets written in the file. Name1, Name2,... must not be keywords recognized by matprint, see below. Thus

```
matprint("Results.mat",values:releigenvals(h,e))
```

will write a matrix on file Results.mat with name 'values' on the first line of the header .

Keywords 'nsig', 'format', 'name', 'header', 'missing' 'width', oldstyle' and 'comments' are all recognized and can appear more than once. They affect the printing of objects that follow them, until they

are changed, except that the values of 'name' and 'comments' are used only once. Any of them that follow all items to be printed are treated as coming before all items. Thus, for example,

```
Cmd> matprint("data.txt",x,nsig:5,y,nsig:10)
```

and

```
Cmd> matprint("data.txt",nsig:10, x,nsig:5,y)
```

are equivalent. This does not apply to keywords 'file' and 'new' which can appear only once.

Keyword 'name':
matprint(FileName,name:charVar, a, b...) prints a with the name specified by quoted string or CHARACTER scalar charVar on the header. This is an alternative to using a keyword to specify a name and can be used when the name is not a legal MacAnova keyword name. Thus matprint("myfile", name:"Residuals",r) and matprint("myfile", Residuals:r) are equivalent. Keyword 'name' can be used several times in the argument list, and affects only the next item to be written to the file. If name:charVar is the last argument, it is treated as if it came before all items to be written to the file.

Keyword 'comment':
matprint(FileName, a, comment:charVec) writes each element of CHARACTER vector or quoted string charVec, prefixed by ") ", as a comment line after the header. If header:F appears, no such comments are written.

Keyword 'missing':
matprint(FileName,a,b,...,missing:realVal) recodes MISSING values with REAL number realVal. Thus, for example, matprint("mydata",x, missing:-99) substitutes -99 for every MISSING value. If 'missing' is not used, MISSING values will be coded as -99999.9999. In either case, for any variable with MISSING values, a comment line of the form ')MISSING value' is written before the data, where value is either -99999.9999 or the value specified by 'missing'. This enables matread() to recognize missing values and read them appropriately. Keyword 'missing' can be used several times, each affecting any variables later in the argument list. If it follows all variables to be printed, as in the example, it is as if it preceded them all. The value for 'missing' cannot itself be a MISSING value.

Note this use of 'missing' differs from print(), write() and setoptions() -- its value must be a REAL scalar, not a character string.

Keyword 'header':
matprint(FileName,a,b,...,header:F) writes the variables without any header lines. They will not be readable by matread() but will be readable by other programs that can read numbers separated by spaces. The only time you need header:T is when sep:"c" is an argument and you want to force the writing of a header.

Keyword 'width':
matprint(FileName,a,b,...,width:w) temporarily sets option 'width' to w, an integer >= 30. This affects how many items are printed per line.

Keyword 'sep':
matprint(FileName,a,b,...,sep:"c"), where c is an arbitrary character, writes items of data separated by c instead of by spaces. This also suppress the printing of header lines unless 'header:T' is an argument. This option is useful if you want to export data to a spreadsheet or other program that can read comma- or tab-separated items. For example, to write x with values separated by commas, use matprint("export.dat",x, sep:","). matprint("export.dat",x,sep:"\t") writes items separated by tabs.

Keywords 'quoted' and 'bylines':
When writing a CHARACTER variable you can also include keyword phrases quoted:T or bylines:T. matprint(fileName, charVar, quoted:T) outputs the data set in "quoted fields" format, that is with each element enclosed in double quotes ("..."). matprint(fileName, charVar, byline:T) outputs the data set in "by lines" format, with each element starting on a new line. However, if there are non-printable characters in the data, "quoted" fields format will be used. You can output a character variable in comma separated quoted fields as is required form some programs such as data bases, by matprint(fileName, charvar, quoted:T,sep:",").

On a version with windows (Macintosh, Windows or Motif), if FileName is "", you will be able to specify the file name and folder using a dialog box.

Keywords 'nsig' and 'format':
matprint() uses the same default format for each item written as does print() and has the same keywords 'nsig' and 'format'. See print() for information on 'nsig' and 'format'.

If an argument is LOGICAL, a comment line of the form ') LOGICAL' is added to the header lines. This is recognized by matread().

If FileName is variable CONSOLE or a CHARACTER variable whose value is "CONSOLE", the output is written to the screen rather than to a file. The value of variable CONSOLE is ignored.

You can change the default format for print(), fprint(), and matprint() by setoptions() using keywords 'nsig' or 'format'.

See also write(), fwrite(), write(), fprint(), matprint(), macrowrite(), matread(), files.

2.184 matread

Keywords: input, files, missing values

Usage: `y <- matread(fileName,setName [,quiet:T or F, echo:T or F, labels:Labels, silent:T, notfoundok:T]), fileName and setName CHARACTER scalars; fileName can also be of the form string:charVal where charVal is a CHARACTER scalar or vector.`

x <- matread(FileName,setName) searches a file for a data set whose name matches setName. If the data set is found, it is read and the data are saved in variable x. The data set must be a REAL, LOGICAL or CHARACTER vector, matrix, or array or a structure with REAL, LOGICAL, CHARACTER or macro components. FileName and setName must be CHARACTER variables or quoted strings. See topic data files for the required form for the data set.

If setName is omitted (x <- matread("mydata")), matread() will read the first dataset on the file, expecting that the first non-blank and non-empty line is a header line of the correct form (see below).

In a version with windows (Macintosh, Windows, Motif), if FileName is the null string "", you will be able to select the file using a dialog box.

Just reading a data set does not make it available; you must assign the value of matread() using '<-'.

Pre-defined macro getdata is somewhat easier to use, provided you have set variable DATAFILE to the name of the file. Since the default value of DATAFILE is "macanova.dat", another way to read 'irisdata' is

```
Cmd> x <- getdata(irisdata)
```

If you have a file of data sets you will be analyzing, say file "mydata.dat", redefine DATAFILE by

```
Cmd> DATAFILE <- "mydata.dat"
```

Then you can use macro getdata to read data sets from your file. See getdata.

If any data items in numerical data set are too large to be represented in the computer (for example "1e3000), they are set to MISSING.

If any data item in the file is not a proper number (for example 3."4a5"), it, together with numbers following it on the same line, are set to MISSING.

matread() and getdata() work only with files in a special format with header information. Use vecread() and readcols() to read data files that just consist of numbers.

matread(CONSOLE) reads from the regular input stream allowing you to type in the matrix using the format described under topic data files.

On the Macintosh, type one line at a time in the dialog box that is opened. On Unix and DOS, type the necessary lines after the prompt. The value of CONSOLE is ignored. The first line entered must be a header which includes a name and dimensions. On any machine, when matread(CONSOLE) is used in a batch file, it reads the data from the lines immediately following the matread() command. This allows even large data sets to be included directly in a batch file. See batch().

There are 4 keywords, 'quiet', 'echo', 'silent' and 'notfoundok' which control what will be printed by matread().

Keyword phrase	Meaning
quiet:T	Header and descriptive comments will not be printed
quiet:F	All header and descriptive comments will be printed
echo:T	Data lines will be printed as they are read
silent:T	Only error messages will be printed; incompatible with quiet:F or echo:T
notfoundok:T	Failure to find the data set is not considered an error so no error message is printed.

Without quiet:T or quiet:F, the header and comment lines not starting with '))' preceding the macro will be echoed to output.

Even without echo:T, data lines are printed when FileName is CONSOLE and the matread() command is in a batch file. (In windowed versions, data will be echoed if FileName is CONSOLE whether or not the command is in a batch file.) Such echoing can be suppressed by 'echo:F'.

When notfoundok:T is an argument and the data set is not found, matread() returns NULL as value. When used in a data set, this feature allows special action if data setName is not found.

matread(file:FileName,...) is equivalent to matread(FileName,...).

matread(string:CharVar,...) where CharVec is a CHARACTER scalar or vector, does not read from a file. Instead, it "reads" CharVar as if each element were a line (or several lines if there are embedded end-of-line characters) read from a file. The first element or line of CharVar must be a header line with a name and dimensioning information. In particular,

```
Cmd> x <- matread(string:CLIPBOARD)
```

would read the first data set on a replica of a data file in the special variable CLIPBOARD. In the Macintosh, Windows and Motif versions this would be taken from the Clipboard. In the Motif version, you can also "read" from special variable SELECTION in a similar way. See topic clipboard.

If either keyword 'file' or 'string' is used, they can appear in any position in the argument list, as can setName which must be the only non-keyword argument. For example,

```
Cmd> matread(quiet:T,"mydataset",file:"myfile.dat")
```

is equivalent to

```
Cmd> matread("myfile.dat","mydataset",quiet:T).
```

See also vecread(), readcols, macroread(), inforead(), files

2.185 matrices

Keywords: matrix algebra, operations, variables

Usage:

```
Matrix transposition
       x' or t(x)
Matrix multiplication
       x %*% y, x %c% y, x %C% y
Matrix inversion
       solve(a)
Linear equation solution
       solve(a, b) or a %\% b, rsolve(a,b) or
       b %/% a
Extract elements
       x[i,j], x[,j], x[i,], i, j integer scalars or
       vectors or LOGICAL vectors.
Eigen values and vectors
       eigen(a), eigenvals(a), releigen(a,b),
       releigenvals(a,b), trideigen(diag,subdiag...)
Other decompositions
       qr(x [,pivot:T]), cholesky(x),
       svd(x [all:T, right:T or F, left:T or F])
Other Functions of matrices
       trace(x), det(x), diag(x), nrows(x), ncols(x)
Create matrices
       matrix(x,nrows), hconcat(a,b,...),
       vconcat(a,b,...), dmat(vec), dmat(x, n)
```

A matrix is a two dimensional array, that is, it has two subscripts.

If x is a REAL, LOGICAL, or CHARACTER vector of length m*n, matrix(x,m) creates an m by n matrix from the elements of x.

A vector of length n is, in most contexts, equivalent to a n by 1 matrix. See vectors.

A generalized matrix is an array with more than two dimensions but which has no more than 2 dimensions greater than 1. With few exceptions, a generalized matrix can be used wherever a matrix can be used.

A generalized matrix with exactly two dimensions with lengths m > 1 and n > 1, is interpreted as a m by n matrix. Thus array(run(20),1,4,1,5) is considered for most purposes as if it were a 4 by 5 matrix.

A generalized matrix whose first dimension is n and all others are 1 is iterpreted as an n by 1 matrix or, in some contexts, as a vector of

length n. For example, array(run(7),7,1,1,1) is generally treated as either a 7 by 1 matrix or a vector of length 7.

A generalized matrix whose first dimension is 1 and there is a single dimension with length n > 1 is interpreted as a 1 by n matrix, that is, as a row vector. Thus array(run(5),1,1,5) is considered to be a 1 by 5 matrix.

A generalized matrix all of whose dimensions are 1 (example: array(17,1,1,1,1)) is interpreted as a 1 by 1 matrix or, in most contexts, a scalar.

If x is a generalized matrix, ismatrix(x) returns True and nrows(x) and ncols(x) return the numbers of rows and columns as just described.

If x is a generalized matrix, matrix(x) is equivalent to matrix(x, nrows(x)) and is an ordinary two dimensional matrix with the same elements as x.

You can compute the transpose of a matrix x by either x' or t(x). The transpose of a generalized matrix is a generalized matrix with the same dimensions in reverse order.

You can multiply two REAL matrices or generalized matrices with conforming dimensions and no MISSING values as follows:

Operator	Precedence	Meaning
x %*% y	11	x MatMult y
x %c% y	11	transpose(x) MatMult y
x %C% y	11	x MatMult transpose(y)

where MatMult is ordinary matrix multiplication. The result is always a matrix with two dimensions, even if either x and.or y is a generalized matrix. Either or both operands can also be structures. See structures.

You can "divide" one matrix by another (in the sense of multiplying by an inverse) if they have conforming dimensions and no MISSING values as follows:

Operator	Precedence	Meaning
x %/% y	11	x MatMult inverse(y) (same as rsolve(y,x)
x %\% y	11	inverse(x) MatMult y (same as solve(x,y)

Neither %/% or %\% can be used with structures.

Note: These 5 matrix operations are "left associative", that is, for example that x %*% y %\% z is equivalent to (x %*% y) %\% z, not x %*% (y %\% z).

Precedence level 11 is just above the precedence level of '*', '/' and '%%' and just below the precedence level of '^'.

Examples:

Expression	Interpretation	Required to be legal
a %*% b + 3	(a %*% b) + 3	ncols(a) = nrows(b)
3 / a %c% b^2	3 / (a %c% (b^2))	nrows(a) = nrows(b)
a / 3 %C% b	a / (3 %C% b)	ncols(b) = 1

See topics arithmetic, logic, and bit operations for the precedence levels of other operators.

NOTE: In version 3.35 and earlier of MacAnova, '%*%', '%c%', and '%C%' had the same precedence as '*', '/', and '%%'.

Here are some functions that are useful with matrices. All treat generalized matrices as matrices.

cholesky()	Compute Cholesky decomposition of x
det(x)	Compute the determinant of x
diag(x)	Extract the diagonal of x.
eigenvals(x) and eigen(x)	Compute eigenvalues and/or eigenvectors of x
hconcat(x,y,...)	Concatenate x, y, ... horizontally (y to the right of x, ...)
nrows(x), ncols(x)	Find the number of rows or columns of x
qr(x [,pivot:T])	Compute QR decomposition of x
releigenvals(a,b), releigen(a,b)	Compute eigenvalues and/or eigenvectors of a relative to b
rsolve(a, b)	Solve x %*% a = b; equivalent to b %/% a.
solve(x)	Invert x
solve(a,b)	Solve a %*% x = b; equivalent to a %\% b.
svd(x)	Compute singular value decomposition of x
swp(x,intvec)	Apply Beaton SWP operator to rows and columns of x specified by intvec
t(x) or x'	Transpose of x
trace(x)	Compute the trace of x
trideigen(diag,subdiag [,...])	Compute eigenvalues and/or eigenvectors of symetric tridiagonal matrix
vconcat(x,y,...)	Concatenate x, y, ... vertically (y below x, ...)

The following are also useful, but they do not treat a generalized matrix x exactly like matrix(x).

max(x)	Maximum of each column of x
min(x)	Minimum of each column of x
prod(x)	Product down columns of x
sum(x)	Sum down columns of x

These all operate on the first actual dimension of x, producing a result with the same number of dimensions as x, but with first dimension 1. If you want to treat a generalized matrix as if it were a matrix, use, say, sum(matrix(x)).

See also det(), trace(), swp(), eigen(), eigenvals(), releigen(), releigenvals(), dim(), nrows(), ncols(), svd(), cholesky().

2.186 matrix

Keywords: matrix algebra, variables, combining variables

Usage: `matrix(x,Rowdim [,KeyPhrases]), x a vector, Rowdim > 0 an integer dividing length(x)`
`matrix(x [,KeyPhrases]), x a generalized matrix.`
`KeyPhrases can be labels:structure(rowLabs,colLabs) and/or silent:T, where rowabs are CHARACTER scalars or vectors.`

matrix(x,Rowdim) creates a matrix (two dimensional array) with Rowdim rows containing the data in x. x can be a vector, matrix, or higher dimensional array, all of whose elements are used, with first subscript changing fastest, second subscript, if any, changing next, and so on.

Rowdim must be a positive integer exactly dividing the length of vector, matrix, or array x.

Example:

```
  Cmd> c <- matrix(vector(1,1,1, -1,1,0, -1,-1,2),3)
```

creates the following matrix:

$$c = \begin{matrix} 1 & -1 & -1 \\ 1 & 1 & -1 \\ 1 & 0 & 2 \end{matrix} .$$

matrix(x), with no Rowdim, is equivalent to matrix(x,nrows(x)). It is valid for any one- or two-dimensional x, or for any higher dimensional array with no more than two dimensions greater than 1, that is for any x such that 'ismatrix(x)' would be True. See ismatrix() and matrices.

Example: h <- matrix(SS[2,,]) creates a true p by p matrix from the 1 by p by p array SS[2,,], if SS is an array of SSCP matrices created as a side effect by manova(). See manova().

Although most operations, including matrix multiplication, matrix inversion, and eigenvalue computation, treat SS[2,,] and matrix(SS[2,,]) identically, there are a few that do not. Using matrix() can avoid some surprises.

On both usages, you can specify row and column labels for the output using keywords labels. See topic labels for details.

See also array(), nrows(), matrices.

2.187 matwrite

Keywords: files, output

Usage: matwrite(fileName, a, b, ... [, format:Fmt, nsig:n, sep:sepChar, quoted:T or bylines:T,missing:mVal, name:Name, comments:charVec, width:w, header:F, oldstyle:T]), a, b, ... arbitrary variables, Fmt, sepChar and Name CHARACTER scalars with sepChar only a single character, charVec a CHARACTER vector or scalar, mVal a REAL scalar, w >= 30 integer.

matwrite(FileName,a,b,... [,new:T]) writes REAL, LOGICAL and CHARACTER variables a, b,... (scalars, vectors, matrices, or arrays) file FileName in a form which can be read by matread(). It can also write NULL variables and structures. GRAPH variables are legal arguments but are currently written as NULL variables.

matwrite(a,b,...,file:FileName [,new:T]) is an alternative usage.

matwrite() differs from matprint() only in the default format used for REAL and LOGICAL variables. It uses the format used by write(), namely the format specified by option 'wformat'. This normally provides 9 significant digits. It is provided to make it easier for you to write data sets with increased precision without having explicitly to provide a format.

See matprint() for information on keywords 'missing', 'sep', 'name', 'header', 'width', 'quoted', 'bylines', 'oldstyle', and 'comments', and print() for information on keywords 'nsig' and 'format'.

See topic data files for a description of the file format.

You can change the default format for write(), fwrite(), and matwrite() by setoptions() using keyword 'wformat'.

See also print(), fprint(), write(), fwrite(), matwrite(), macrowrite(), matread(), files.

2.188 max

Keywords: descriptive statistics

Usage: max(x), x REAL or LOGICAL or a structure with REAL or LOGICAL components. max(x1,x2,...), x1, x2, ... REAL or LOGICAL vectors, all the same type.

max(x) computes the maximum of the elements of a REAL or LOGICAL vector x. If x is LOGICAL, True is interpreted as 1.0 and False as 0.0 and hence max(x) is 1.0 if any element of x is True, and 0.0 if all elements are False.

If x is a m by n matrix, max(x) computes a row vector (1 by n matrix) consisting of the maxima of the elements in each column of x.

If x is an array with dimensions n1, n2, n3, ..., max(x) computes an array with dimensions 1, n2, n3, ... such that y[1,j,k,...] = max(x[i,j,k,...], i=1,...,n1). This is consistent with what happens when x is a matrix. Note: MacAnova3.35 and earlier produced a result with dimensions n2, n3, ...

max(NULL) is NULL.

max(a,b,c,...) is equivalent to max(vector(a,b,c,...)) if a, b, c, ... are all vectors. They must all have the same type, REAL or LOGICAL, or be NULL. max(NULL,NULL,...,NULL) is NULL.

If all the elements of a vector x are MISSING, max(x) is MISSING.

If x is a structure, max(x) computes a structure, each of whose components is max() applied to that component of x.

Example:

If x is a n by m matrix, r <- x/max(x) computes the matrix of the ratios of x[i,j] to the maxima for each column. See arithmetic.

If x is a n by 4 by 5 array, r <- x/max(n) computes an array with r[i,j,k] = the ratio of x[i,j,k] to the maximum of all x[i,j,k] with the same values for j and k. That is, it treats x analogously to a 4 by 5 array of vectors of length n. See arithmetic.

See also min().

2.189 min

Keywords: descriptive statistics

Usage: min(x), x REAL or LOGICAL or a structure with REAL or LOGICAL components. min(x1,x2,...), x1, x2, ... REAL or logical vectors, all the same type.

min(x) computes the minimum of the elements of a REAL or LOGICAL vector x. If x is LOGICAL, True is interpreted as 1.0 and False as 0.0 and hence min(x) is 0.0 if any element of x is False, and 1.0 if all elements are True.

If x is a m by n matrix, min(x) computes a row vector (1 by n matrix) consisting of the minima of the elements in each column of x.

If x is an array with dimensions n1, n2, n3, ..., y <- min(x) computes an array with dimensions 1, n2, n3, ... such that y[1,j,k,...] = min(x[i,j,k,...], i=1,...,n1). This is consistent with what happens when x is a matrix. Note: MacAnova3.35 and earlier produced a result with dimensions n2, n3, ...

min(NULL) is NULL.

min(a,b,c,...) is equivalent to min(vector(a,b,c,...)) if a, b, c, ... are all vectors. They must all have the same type, REAL or LOGICAL, or be NULL. min(NULL, NULL, ..., NULL) is NULL.

If all the elements of a vector x are MISSING, min(x) is MISSING.

If x is a structure, min(x) computes a structure, each of whose components is min() applied to that component of x.

Example:
If x is a n by m matrix, r <- x - min(x) computes the matrix of the residuals of x[i,j] from the column minimums.

If x is a n by 4 by 5 array, r <- x - min(n) computes an array with r[i,j,k] = the residual of x[i,j,k] from the minimum of all x[i,j,k] with the same values for j and k. That is, it treats x analogously to a 4 by 5 array of vectors of length n. See arithmetic.

See also max().

2.190 modelinfo

Keywords: glm

Usage: modelinfo([all:T] keyword1:T or F, keyword2:T or F ...[,nomodelok:T]) where keywords are one or more of strmodel, termnames, xvars, y, weights, coefs, xtxinv, parameters, scale, colcount, aliased, bitmodel, link, distrib, sigmahat.

modelinfo(keyword1:T, keyword2:T, ...) computes one or more vectors or matrices associated the most recent GLM (generalized linear or linear model) command such as regress(), anova(), poisson(), or glmfit(). This gives you direct access to such things as the design variables or X-variables (xvars:T), the estimated coefficients of the X-variables (coefs:T), and the inverse of the X'X matrix (xtxinv:T).

You can't use modelinfo() after fastanova(), ipf(), or screen().

Any component requested that is not available is set to NULL. In particular this happens for components coefs and xtxinv after anova() when the model is balanced, or after any GLM command with coefs:F, and for component parameters after any GLM for which a sample size or other parameter is specified such as for logistic() and probit().

If more than one of the keywords specifying type of output has value True, modelinfo() returns a structure with component names the same as the keywords. Otherwise it returns a vector or matrix.

modelinfo(all:T) is equivalent to modelinfo(xvars:T,y:T,coefs:T, xtxinv:T,colcount:T,weights:T,parameters:T,strmodel:T,bitmodel:T, termnames:T, scale:T, sigmahat:T,aliased:T). To suppress any particular components, say strmodel, bitmodel, and termnames, use modelinfo(all:T, strmodel:F,bitmodel:F,termnames:F). Use of all:F is an error.

Normally, it is an error if there is no active GLM model. However, if nomodelok:T is an argument, when there is no active model modelinfo() returns NULL without printing an error message. Thus you can test for the existence of an active model by

```
Cmd> if(isnull(modelinfo(strmodel:T,nomodelok:T))){..do something..}
```

See isnull().

Here is a list of the permissible keyword phrases and descriptions of what they return. With xvars:T or coefs:T you can also use keyword 'missing'. With any of them you can also use keyword 'nomodelok'.

xvars:T
The matrix of X-variables associated with the active model. If there is no active model, but STRMODEL is defined and there are no other keywords, it works identically to xvariables(). If 'missing:Value' is an argument, where Value is a REAL scalar, the X-variable values for a case with any missing data will be set to Value. In particular, you can used 'missing:?' to set the X-variables for a case with MISSING data to MISSING. The default value is 0. See xvariables() for more information.

y:T
The dependent variable in the model as a vector or matrix. modelinfo(y:T) is thus equivalent to modelvars(0).

weights:T
A REAL vector containing the weights associated with each case. If there are no missing data and no weights were specified, either explicitly or implicitly, this is a vector of 1's. Otherwise it contains either the weights specified by keyword 'weights' or 'wts' in anova(), manova() or regress() or the implicit weights from the final iteration of poisson(), logistic(), or robust(). The weight for any case with MISSING data is always zero.

parameters:T
A REAL vector containing the sample sizes or other distribution parameters after logistic() or probit(), or after glmfit() with keywords n or parameters.

colcount:T
A REAL vector containing the numbers of X-variables assocated with each term in the active model. The numbers of the first column associated with each term can be obtained by autoreg(1, modelinfo(colcount:T)). If no X-variables are aliased with earlier X-variables, these values are the degrees of freedom associated

with each term.

aliased:T
A LOGICAL vector whose length is the number of X-variables in the model. The i-th variable is True if and only if the i-th X-variables as returned by xvars:T is aliased with previous X-variables. If there was no aliasing every element should be False.

coefs:T
The vector of coefficients of the X-variables in the fitted model. Coefficients corresponding to aliased X-variables (those that are apparently linearly dependent on previous X-variables) are set to zero. After manova() with a p-dimensional response matrix, the coefficients form a matrix with p columns. Note: If there are factors in the model, some of the coefficients computed will differ from the coefficients computed by coefs() and secoefs().

xtxinv:T
The inverse of the X'X matrix computed from the X-variables. The row and column corresponding to any aliased X-variable is set to zero. If the previous GLM command specified weights either explicitly (keyword 'weights' or 'wts' on anova(), regress(), manova()) or implicitly (poisson() or logistic()) the matrix computed is the inverse of X'WX, where W is the diagonal matrix of the weights. After robust(), the matrix computed is the inverse of X'X, ignoring the implicit weights. The weights may be obtained by keyword phrase weights:T (see above).

scale:T
A REAL factor or factors to multiply the square roots of the diagonals of the inverse of the X'X matrix so as to obtain estimated standard errors for the estimated coefficients. After logistic(), poisson(), probit(), or glmfit(), scale will be the default value, unless changed by keyword 'scale' on the GLM command. After other GLM commands, including robust(), scale will be sqrt(SSerror/DFerror), where DFerror and SSerror come from the final line of the ANOVA table. After manova(), scale will be the vector consisting of the square roots of diagonal elements of SSerror/DFerror, where SSerror is the error matrix.

sigmahat:T
The robust estimate of sigma printed after the robust() ANOVA table. It is not usable after any other GLM command. Note that this is not a suitable value to use in computing standard errors.

strmodel:T
A CHARACTER variable containing the current model taken from STRMODEL.

bitmodel:T
A REAL vector or matrix with as many rows as there are terms in the model, including the CONSTANT term, if any, but excluding the final

error term. This encodes the model in a special form. See below for details.

termnames:T
A CHARACTER vector of the terms in the model taken from TERMNAMES. This includes the name (usually "ERROR1") of the final error term, and thus the length of the result is 1 greater than the number of rows in modelinfo(bitmodel:T).

Each row of modelinfo(bitmodel:T) corresponds to a term in the model and consists of one or more integers between 0 and 4294967295 = 2^32 - 1, the bits of whose binary representation encode the variates and/or factors in that term. If there are 33 to 64 or 65 to 95 variates and factors, the result is a matrix with 2 or 3 columns. Thus each row of the output has room for Nvar bits, where Nvar is the number of variates and/or factors in the model.

The bits of each row should be considered to be numbered from 1 to Nvar. Bit 1 is the least significant and bit 32 is the most significant bit of the first element; bit 33 is the least significant and bit 64 is the most significant bit of the second element, if any; and bit 65 is the least significant and bit 96 is the most significant bit of the third element, if any. Bit i of row j of the result is 1 if and only if the i-th variable or factor is in the j-th term of the model, following the order in which variables and factors first appear in the model. All the elements in a row corresponding to the CONSTANT term (usually row 1) are zero.

See topics bit operations and nbits() for information on how to extract information from the result of modelinfo(bitmodel:T).

See also varnames(), modelvars(), xvariables(), models.

2.191 models

Keywords: glm, anova, regression

Usage:

```
Regression:  regress("y=x1+x2+...+xk")
One-way ANOVA: anova("y=A"), factor A
Randomized block ANOVA: anova("y=Repl+A"), factors Repl
and A
Nested ANOVA: anova("y=A/B") or anova("y=A+A.B")
Two-way factorial:  anova("y=A*B") or
anova("y=A+B+A.B"), factors A and B
Completely randomized Split plot ANOVA:
anova("y=A+E(Repl.A)+B+A.B"), factors A, B, and Repl
Analysis of covariance:  anova("y=x+A"), factor A,
variate x
Transform variables on the fly:  regress("log10(y)=sqrt(x)")
Polynomial regression:  regress("y=P3(x)")
Periodic regression:  regress("y=C2(2*PI*hour/24)")
```

All of the GLM (generalized linear or linear model) commands such as regress(), anova(), or poisson() require you to specify a model in a quoted string or CHARACTER variable.

A model can be specified as
"Response = Term" or "Response = Term1 + Term2 + ..."
where 'Response' is the name of the dependent or response variable and each term is of the form Name or Name1.Name2. ... .Namek, where Name, Name1, Name2 ... are the names of variables. A period or dot (.) between the variable names is interpreted as a product operator indicating that all combinations of the variable values are included in the model.

Model Variables

Variables in model terms, including those computed on the fly (see below), may be either factors (vectors of positive integers created using factor()) or variates. No more than one variate may appear in a single term. Up to 95 variables may appear in a model, including no more than 31 factors.

Factors and variates must be vectors or matrices with one column. They must all have the same number of rows as the response variable.

Any factors must have been created using function factor() or been selected from such a variable using subscripts. For balanced designs with factor levels in a reasonable order a factor may often be computed by factor(rep(run(r),s)), factor(rep(run(s), rep(r,s))), or something similar. See factor(), rep().

The constant term may be specified as 1, but is always included by default, that is "y = Model" is equivalent to "y = 1 + Model". You can omit a constant term by "y = Model - 1" or move it to the end by "y = Model - 1 + 1".

Computing Variables "on the fly"

You can transform or otherwise compute model variables "on the fly." In place of the name of a variable, including the response variable, you can use {Expr}, where Expr is a MacAnova expression such as x^2 or log10(y). If the same expression, say {sqrt(x)}, appears more than once in a model, it is evaluated only once and only one model variable is introduced. In comparing expressions, leading and trailing spaces are ignored, so that { sqrt(x) } is considered the same as {sqrt(x)}; however, other differences in the presence or placement of spaces will cause expressions to be considered different variables. Thus {sqrt(x)} is not recognized to be the same as {sqrt(x)}. The only limitation on Expr is that it may not directly or indirectly execute another GLM command.

Since subscripted factors remain factors (see subscripts), when groups is a factor, anova("{y[-3]} = {groups[-3]}") computes a one factor analysis of variance omitting case 3.

Examples

a and b are factors and y, x1, x2, and x3 are REAL vectors

Model	Description
"y = a + b + a.b"	Two factor model with both main effects and interaction
"y = a + a.b"	Two factor model with b nested in a
"y = x1 + x2 + x3"	Three variable multiple regression
"{sqrt(y)} = x1 + {x1^2}"	2nd order polynomial regression of square root of y on x.

Shortcuts for Polynomial and Periodic Regression

You can use special short cuts of the form Pn(expr) and Cn(expr) to specify a polynomial term or a periodic term, respectively, where n is an integer between 1 and 95 and expr is a MacAnova expression. For example, P4(x-10) expands to ({x-10}+{(x-10)^2}+{(x-10)^3}+{(x-10)^4}) and C2(2*PI*x/24) expands to ({cos(2*PI*x/24)}+{sin(2*PI*x/24)}+ {cos(2*(2*PI*x/24))}+{sin(2*(2*PI*x/24))}).

Pn(expr) and Cn(expr) can be used wherever a variable name can be used on the right side of '=', except not in a {...} expression. Thus the last example in the preceding list could have been written "{sqrt(y)} = P2(x1)". They can be "dotted" with a factor. For example, P2(x).a expands to {x}.a + {(x)^2}.a.

If you are doing a regression on a subset of cases uses subscripts, the subscripts must be applied to x, not Cn(x) or Pn(x). For example,

```
Cmd> regress("{y[-run(3)]} = P3(x[-run(3)])")
```

fits a cubic polynomial omitting the first 3 rows of x and y.

See below for other shortcuts you can use to specify models.

Combining Variables

Parts of terms can be replaced by 'submodels', enclosed in parentheses, for example,

```
Cmd> anova("y = a + b + c + d + (a + b).(c + d)")
```

is equivalent to

```
  Cmd> anova("y = a + b + c + d + a.c + b.c + a.d + b.d")
```

The product of a factor or variate with itself (a.a) is equivalent to the variate or factor itself. For example,

```
  Cmd> anova("y = (a + b).(a + c)")
```

is equivalent to

```
  Cmd> anova("y = a + b.a + a.c + b.c")
```

The order of factors and variates in a term is immaterial. That is a.b is equivalent to b.a.

The order of terms in a model is very important since fitting a model is done sequentially, term by term. For example, although "y = a + a.b" is a model with b nested in a, "y = a.b + a" is computationally equivalent to "y = a.b", since after fitting all combinations of a and b, there is nothing left for 'a' to fit.

If a term in a model is duplicated, only the first occurrence is retained. For example, (a + b).(a + b) expands to a.a + b.a + b.a + b.b which is equivalent to a + a.b + a.b + b which is trimmed to a + a.b + b (which is computationally equivalent to a + a.b).

Order of terms in expanded models

If M1, M2, ..., Mk, N1, N2, ..., Nl are terms or submodels, (M1 + M2 + ... + Mk).(N1 + N2 + ... + Nl) is equivalent to M1.N1 + M2.N1 + ... + Mk.N1 + M1.N2 + M2.N2 + ... + Mk.N2 + ... + Mk.Nl

If M1, M2, ..., Mk are terms or submodels, M1.M2.M3. ... .Mk is expanded as (...((M1.M2).M3) ...).Mk.

Short cut formulas for combining terms or submodels

In the following, M1, M2, ... are terms or submodels.

M1*M2 is an abbreviation for M1 + M2 + M1.M2

M1*M2* ... *Mk is an abbreviation for (...((M1*M2)*M3) ...)*Mk. In particular, M1*M2*M3 is an abbreviation for (M1*M2)*M3, that is for M1 + M2 + M1.M2 + M3 + M1.M3 + M2.M3 + M1.M2.M3

M1^N is an abbreviation for M1.(1+M1). ... .(1+M1), where there are N factors. N must be a digit between 1 and 31. This contains the same terms as M1*M1*...*M1 (N factors) but in a different order. Note that M1^N is usually not equivalent to and does not contain the same terms as M1. ... .M1 (N dot factors).

M1/M2 is an abbreviation for M1 + M1.M2

M1 - M2 is an abbreviation for a model containing all the terms in M1, omitting any term in M2. In particular Model - 1 specifies a model with no constant term or intercept and Model - 1 + 1 specifies a model with a contant term that is fit after all other terms in Model.

M1 -* M2 is an abbreviation for a model containing all the terms in M1,

but omitting any terms containing all the variables in any term of M2.

Examples of use of shortcuts. Note the order of the exanded terms.
"y = a*b" is equivalent to "y = a + b + a.b"
"y = a/b" is equivalent to "y = a + a.b"
"y = a*b*c" is equivalent to "y = a + b + a.b + c + a.c + b.c + a.b.c"
"y = (a+b+c)^2" is equivalent to "y = a + b + c + a.b + a.c + b.c"
"y = (a+b+c)^3" is equivalent to "y = a + b + c + a.b + a.c + b.c + a.b.c"
"y = a*b*c - a.b.c" is equivalent to "y = a + b + a.b + c + a.c + b.c"
"y = a*b*c -* (a.b + a.c)" is equivalent to "y = a + b + c + b.c"

Note although "y=a*b*c" and "y=(a+b+c)^3" contain the same terms when expanded, they are in a different order.

Error Terms

In the output from commands such as anova() or poisson() that produce an analysis of variance or deviance table, there is always one line, usually labeled "ERROR1", following all the terms explicitly or implicitly specified in Model. It consists of the sum of squares or deviance associated with all the degrees of freedom not included in the model. If the model fitted uses up all the degrees of freedom this line will still be present, but will have 0 degrees of freedom.

You can also label other terms as ERROR. If a term is of the form E(Term) (for example, E(a.b.c)), it will be labeled "ERRORn" in the ANOVA table, where n is 1, 2, ..., . The final error line will still be printed but will be labeled "ERRORm", where m-1 is the number of error terms you specified. E(1) is not legal, nor is it legal to specify a term as an error term more than once (E(a.b) + E(a.b)). Moreover, once a term is designated as an error term, it cannot be deleted by '-' or '-*'. Term in E(term) must be a single factor or a pure product of factors. Thus E(a.b+a.b.c) is illegal.

A '#' in Model marks the end of the model, allowing models to be self-documenting as in anova("y = a + b #additive model").

Any GLM command sets the CHARACTER variable STRMODEL to the specified model as a "side effect" of the analysis. If no model is specified on a subsequent GLM commands (for example anova()), it is taken from this variable. Alternatively, if you set STRMODEL directly, for example

```
Cmd> STRMODEL <- "y = x1 + x2 + x3"
```

then the value of STRMODEL will be used by the next GLM command if it has no model as argument. Note, however, when you assign a value to STRMODEL, MacAnova discards the internal information saved by the most recent GLM command that is used by functions such as secoefs() and contrast().

Examples of GLM Models

```
Cmd> anova("y = a + b + a.b") # or anova("y = a*b")
```

will produce a two-way analysis of variance with interaction for the

response in y, provided vector y is defined and a and b are factors with the same length as y.

```
Cmd> anova("y = a + a.b") # or anova("y = a/b")
```
where a and b are factors will produce a nested analysis of variance with b nested within a.

```
Cmd> anova("y = blk + a + E(a.blk) + b + a.b")
```
would be appropriate for the analysis of a two factor split plot experiment with the whole plot treatments in a randomized block design. Do not attempt to use the name 'rep' for a blocking factor, since 'rep' is the name of a built-in operation.

2.192 modelvars

Keywords: glm

Usage: modelvars(varList [,Model]), varList a vector of integers >= 0, Model a CHARACTER scalar modelvars(y:T or x:T or variates:T or factors:T or all:T [, Model]) modelvars(nx:T or nvariates:T or nfactors:T or hasconst:T [, Model])

modelvars(VarList,Model), where VarList is a vector of non-negative integers, say vector(i1,i2,i3,...), returns a vector or matrix whose columns are the variables in the model specified by the scalar CHARACTER variable or quoted string Model. Variable 0 is the dependent variable (the variable before '=' in Model) and, if i > 0, variable i is the i-th variate or factor appearing on the right hand side of Model (after '=').

See topic 'models' for information on specifying Model.

modelvars(y:T,Model) returns a vector or matrix containing the dependent variable of Model. This usage yields the same result as modelvars(0, Model).

modelvars(x:T,Model) returns a vector or matrix containing the independent variates and factors of on the right hand side of Model. This yields the same as modelvars(run(nv), Model), where nv is the number of variates and factors. When there are no variates and factors ("y=1"), NULL is returned. See keyword 'nx' below for determining the total number of variates and factors.

modelvars(factors:T,Model) returns a vector or matrix containing the factors on the right hand side of Model. When there are no factors in the model, NULL is returned. See keyword 'nfactors' below for determining the total number of factors.

modelvars(variates:T,Model) returns a vector or matrix containing the variates on the right hand side of Model. When there are no variates in the model, NULL is returned. See keyword 'nvariates' below for determining the total number of variates.

modelvars(all:T,Model) returns a matrix containing the dependent variable followed by the independent variates and factors of Model. Equivalent to this is modelvars(run(0,nv),Model).

When Model is omitted (modelvars(VarList) or modelvars(keyword:T)), variables are taken from internal copies of the variables in the current active model. This allows retrieval of the dependent variable and/or model variables even if they were temporary variables (their names started with '@'). When there is no active model but variable STRMODEL exists, modelvars(keyword:T) and modelvars(VarList) are equivalent to modelvars(keyword:T,STRMODEL) and modelvars(VarList,STRMODEL). Here keyword is one of 'x', 'y', 'factors', 'variates', or 'all'.

Examples:

```
modelvars(vector(1,2,0),"y=x+a") is equivalent to hconcat(x,a,y)
modelvars(x:T,"y=x+a+a.x") is equivalent to hconcat(x,a)
modelvars(all:T,"y=x1+x2") is equivalent to hconcat(y,x1,x2)
```

Note: Any variables that are factors are returned unchanged. This is very different from xvariables(), which computes dummy X-variables associated with a factor.

Counting Factors and Variables

You can also use modelvars() to determine how many factors and variates there are in a model or to check whether the constant term is in the model. This can be useful in a macro using the results of a GLM command to do further analyses.

modelvars(nx:T [,Model]) returns the number of independent variates and factors on the right hand side of Model, that is, what would be computed by ncols(modelvars(x:T [,Model])). When there are no variates and factors ("y=1"), 0 is returned.

modelvars(nfactors:T [,Model]) returns the number of factors on the right hand side of Model, that is, what would be computed by ncols(modelvars(factors:T, [,Model])). When there are no factors 0 is returned.

modelvars(nvariates:T [,Model]) returns the number of variates on the right hand side of Model, that is, what would be computed by ncols(modelvars(variates:T [,Model])). When there are no variates 0 is returned.

modelvars(hasconst:T [,Model]) is True if and only if the constant term is in the model.

Examples:

```
modelvars(nfactors:T,"y=x+a+b+a.x") returns 2
modelvars(nvariates:T,"y=x+a+b+a.x") returns 1
modelvars(nx:T,"y=x+a+b+a.x") returns 3
modelvars(hasconst:T, "y=x") returns True
modelvars(hasconst:T, "y=x-1") returns False
```

See also models, varnames(), xvariables().

2.193 more

Keywords: output, general
Usage: more(x [, nsig:n, format:Fmt, missing:M]), where x is a macro or is a REAL, CHARACTER, or LOGICAL variable, n > 0 is an integer, Fmt and M are CHARACTER scalars

more(x) displays object x using a 'paging' program that displays a screenful at a time. On Unix by default it uses Unix program 'more'. If variable PAGER exists and is a CHARACTER scalar, then it is assumed to specify a paging program. For example, on some Unix systems, if the value of PAGER is "less -x4", then more invokes Unix program 'less' with tab stops set every 4 positions.

You may use any of keywords 'nsig', 'format', and 'missing' as on print(), but not 'name' and 'file'.

The lines written to the screen are not written to a spool file, nor are they redisplayed after a plot. See spool().

More is implemented as a pre-defined macro and is not available in all versions of MacAnova.

2.194 movavg

Keywords: time series
Usage: movavg(Phi,A [,reverse:T, limits:vector(i1,i2), start:startVals]), Phi REAL vector, A REAL vector or matrix

movavg(theta,a) applies the moving average operators specified by the columns of the REAL matrix theta to the columns of the REAL matrix a. If ncols(theta) = 1, theta is applied to every column of a and if ncols(a) = 1, each column of theta is applied to a. The result is a matrix with nrows(a) rows and max(ncols(theta), ncols(a)) columns. If both theta and a have more than one column, they must both have the same number of columns.

Specifically, assuming for simplicity that both theta and a are vectors so that the result x is a vector, then

```
    x[i] = a[i] - sum(theta[k]*a[i-k],1<=k<=nrows(phi)),
```

with a[1] taken to be 0 for 1 < 1.

A common usage is movavg(1,a), where a is a vector or matrix. This computes the first differences a[1,] - 0,a[2,]-a[1,], ...,a[n,]-a[n-1,].

```
Second differences can be computed by movavg(vector(2,-1),a).

movavg(theta,a,reverse:T) applies the moving average operator in
reverse:
    x[i] = a[i] - sum(theta[k]*a[i+k],1<=k<=nrows(phi))
with a[1] = 0 for 1 > nrows(a).

movavg(theta,a,limits:vector(i1,i2),start:StartVals[,reverse:T]) is the
same except that x[i] is computed as just described only for i1 <= i <=
i2, with the remaining values copied from rows 1 to i1-1 and rows i2+1
to nrows(a) of matrix StartVals.  StartVals must be the same size and
shape as a.  Unlike what happens with autoreg(), the values computed for
rows i1 to i2 are unaffected by the values of StartVals.

Example (theta and theta1 vectors of same length):
  Cmd> m <- nrows(theta); n <- 300
  Cmd> movavg(theta,rnorm(n+m))[-run(m)]
     generates a moving average series with normal innovations.
  Cmd> movavg(theta,matrix(rnorm(10*(n+m),10))[-run(m),]
     generates 10 independent moving average series
  Cmd> movavg(hconcat(theta,theta1),rnorm(n+m))[-run(m)]
     generates two moving average series with the same innovations

movavg() is the inverse of autoreg() and vice versa, in that
   movavg(phi,autoreg(phi,x))   and   autoreg(phi,movavg(phi,x))
both reproduce x, except for rounding error.

See also autoreg().
```

2.195 nameof

Keywords: variables, character variables
Usage: nameof(var1, var2, ...)

```
nameof(arg1, arg2, ..., argk) returns a CHARACTER vector containing the
names of the arguments.  If an argument is the result of a computation
or is a quoted string, its name will be descriptive.  For example, if x
is a matrix, the value of
  Cmd> nameof(x,cos,17,3+5,describe(x)$mean,matrix(run(10),5),"Hello")
is equivalent to
  Cmd> vector("x","cos","NUMBER","NUMBER","VECTOR","MATRIX","STRING")

See also compnames(), varnames(), rename().
```

2.196 nbits

Keywords: operations, transformations, glm
Usage: nbits(x), where x consists of 1 or more integers between 0 and 4294967295

nbits(x), where x is an integer with value between 0 and 4294967295 (2^32-1), computes the number of non-zero bits in the binary representation of x. For example, nbits(123455) is 11 since 123455 has binary representation 00000000000000011110001000111111b.

If x is not an integer or x < 0 or x > 4294967295, a warning message is printed and the result is set to MISSING.

If x is a REAL vector, matrix or array or a structure all of whose components are REAL, nbits(x) is a variable or structure of the same size and shape as x, each element of which is the number of bits in the corresponding element of x.

nbits() is useful with the output of modelinfo(bitmodel:T). For example, vector(sum(nbits(modelinfo(bitmodel:T)'))) computes a vector containing the number of variables or variates in each term of the model used by the most recent GLM (generalized linear or linear model) command such as regress(), anova(), or poisson(), while sum(vector(nbits(modelinfo(bitmodel:T)[3,]))) computes the number of variables or variates in term 3 of that model

See also bit operations, modelinfo().

2.197 ncols

Keywords: variables
Usage: ncols(x) where x is a matrix or generalized matrix

ncols(x) returns the number of columns of matrix argument x. If x has more than two dimensions, no more than two dimensions may exceed 1 and it is treated as described in topic 'matrices'.

Examples: Let x by array(run(24),1,6,1,4) and y be run(7). Then
```
   vector(nrows(x),ncols(x))   is vector(6,4)
   vector(nrows(y),ncols(y))   is vector(7,1)
   vector(nrows(y'),ncols(y')) is vector(1,7)
```

If x is a structure, ncols(x) is a structure. If xi is the i-th component of x, the i-th component of ncols(x) is ncols(xi).

See also nrows(), dim(), ndims(), length(), structures.

2.198 ncomps

Keywords: variables, structures
Usage: ncomps(Str) where Str is a structure

ncomps(Str) returns the number of components in structure Str. It is an

```
error if Str is not a structure.

See also ndims(), dim(), length(), isstruc(), structures.
```

2.199 ndims

Keywords: variables, null variables
Usage: `ndims(x)`

```
ndims(x) computes the number of dimensions of x.  If x is a vector,
ndims(x) is 1; if x is a matrix, ndims(x) is 2, even if has column
dimension 1.

If x is a NULL variable, ndims(x) is 0.

Examples:  Let x by array(run(24),1,6,1,4) and y be run(7). Then
   vector(ndims(x),ndims(y),ndims(y'),ndims(y'')) is vector(4,1,2,2)

If x is a structure, ndims(x) is a structure.  If xi is the i-th
component of x, the i-th component of ndims(x) is ndims(xi).

See also length(), dim()
```

2.200 nrows

Keywords: variables
Usage: `nrows(x), x a matrix or generalized matrix`

```
nrows(x) returns the number of rows of matrix argument x.  If x has more
than two dimensions, no more than two dimensions may exceed 1 and it is
treated as described in topic 'matrices'.
Examples:  Let x by array(run(24),1,6,1,4) and y be run(7). Then
   vector(nrows(x),ncols(x))   is vector(6,4)
   vector(nrows(y),ncols(y))   is vector(7,1)
   vector(nrows(y'),ncols(y')) is vector(1,7)

If x is a structure, nrows(x) is a structure.  If xi is the i-th
component of x, the i-th component of nrows(x) is nrows(xi).

See also ncols(), dim(), ndims(), length(), structures.
```

2.201 outer

Keywords: matrix algebra
Usage: `outer(x1, x2, ...), x1, x2, ...  REAL`

outer(x1, x2) returns a matrix or array which is the "outer product" of x1 and x2 which must be REAL. The dimensions of the result are the joined dimensions of the arguments. That is, outer(x1, x2) is equivalent to array(vector(x1)*vector(x2)', vector(dim(x1), dim(x2))). More specifically outer(x1,x2)[i,j,k,...,l,m,n,...] = x1[i,j,k,...] * x2[l,m,n,...].

outer(x1, x2, x3) is mathematically equivalent to outer(outer(x1,x2), x3) and in general outer(x1, x2, x3, ..., xk) is mathematically equivalent to outer(outer(...(outer(x1,x2), x3), ...), xk). The elements of the result are all possible k-way products of elements from each of the arguments.

One use for outer() is constructing multidimensional contrasts that are products of 1 dimensional contrasts. Thus if c1, c2 and c3 are vectors of main effect contrast coefficients for each factor for a 3 factor design, outer(c1,c2,c3) defines a contrast among the three-way interaction effects.

See also contrast(), array(), matrices

2.202 padto

Keywords: time series
Usage: padto(x,n), x a REAL vector or matrix, n > 0 an integer

padto(x,n) creates a new matrix or vector from vector or matrix x by adding n - nrows(x) rows of all zeros so as to bring the total number of rows to n. If n < nrows(x), the last nrows(x) - n rows of x are deleted to bring the number down to n. n must be a positive integer.

The principal use of padto() is to add zeros to a time series after subtracting the mean or other estimate of trend but before computing its Fourier transform, as in rft(padto(x-sum(x)/nrows(x),S)), where S is the number of frequencies desired. If x has several columns, they all get padded simultaneously.

2.203 partacf

Keywords: time series
Usage: partacf(vec [, inverse:T]), vec a REAL vector.

partacf(rho), where rho is a REAL vector, computes the partial autocorrelations corresponding to the autocorrelation function in the REAL vector rho. Row k of rho should contain the lag k autocorrelation. If rho is a matrix, partacf(rho) is a matrix of the same shape whose j-th column contains partial autocorrelations corresponding to autocorrelations in column j of rho. If any column of rho is not a

valid autocorrelation function, that is, it does not define a positive definite Toeplitz matrix, a warning message is printed. The Levinson-Durbin algorithm is used.

partacf(pacf,inverse:T) is the inverse function to partacf. Each column of REAL matrix pacf is considered to be the partial autocorrelation function of a time series. The corresponding column of the result is the corresponding autocorrelation function. All the elements of pacf must be less than 1 in absolute value.

See also yulewalker().

2.204 paste

Keywords: output, missing values

Usage: paste(arg1, arg2, ... [, format:Fmt, sep:C, intwidth:Iw, charwidth:Cw, missing:S]), Fmt, C, and S CHARACTER scalars, Iw and Cw integers > 0
paste(arg, multiline:T [, format:Fmt, sep:Cs, linesep:Cl, missing:S]), where Cs and Cl are CHARACTER scalars consisting of a single character.

paste(arg1, arg2, ...) returns a CHARACTER variable concatenating the arguments. Thus the value of

```
paste("The answer is", run(7), "; ok?")
```

is the string "The answer is 1 2 3 4 5 6 7 ; ok?" .

An important use of paste() is in constructing labels for graphs (for example, title:paste("Variable",j,"vs variable",i)). It is also useful for producing informative messages to be printed in a macro and can even be used to prepare nicely formatted lines for output.

The default behavior is to print the arguments separated by single spaces, with exact integers printed as such, non-integers printed using the default print format (see print(), setoptions()) with leading spaces trimmed off, and missing values printed as "MISSING". If you have used setoptions() to replace "MISSING" by a different default string, the replacement will be used.

paste(arg1, arg2, ..., sep:S), where S is a CHARACTER variable or quoted string, uses S to separate arguments rather than a space. Thus paste(run(5),sep:",") produces the string "1,2,3,4,5" and paste("A","B","C","D","E",sep:"") produces the string "ABCDE". No separator is ever put before the first item in the output variable. You can have several instances of 'sep:S' with different separators, each affecting later arguments until changed.

paste(arg1, arg2, ..., intwidth:w), where w is a positive integer prints each exact integer using at least w positions, padding on the left with spaces if necessary. Thus paste(12,intwidth:5) produces " 12".

paste(arg1, arg2, ..., format:Fmt), where Fmt is a CHARACTER variable or quoted string representing either a f-format ("10.5f" or "f10.5") or g-format ("11.7g" or "g11.7") (see print()) prints any non-integer REALs using format Fmt If the width is omitted (".7f") leading spaces will be stripped off. If the width is not omitted ("10.7f"), any non-integer REAL variables printed will be formatted so as to use at least this width. If 'intwidth:w' has not previously appeared, this width will also be used for integers.

paste(arg1, arg2, ..., charwidth:w), w is a positive integer, uses at least w character positions for any CHARACTER argument, padding on the right with spaces if necessary, unless justify:"r" or justify:"c" is an argument.

paste(arg1, arg2, ..., justify:C), where C is "right", "left", or "center" (or simply "r", "l", or "c"), specifies that any strings are to be right justified, left justified or centered, respectively. This has no effect unless charwidth:w is specified and a string is shorter than w. The default is justify:"left".

paste(arg1, arg2, ..., missing:String), where String is a quoted string or CHARACTER scalar such as "?", uses String to represent a missing value instead of "MISSING". If a format has been specified with width > 0 longer than the length of String, String will be padded on the left to make it have this width.

You can use any of the keywords more than once anywhere in the argument list, with each usage affecting the formatting of subsequent items until changed by a new keyword phrase. Putting it after all non-keyword arguments, as illustrated above, is equivalent to putting it before them. For example, paste(sep:",",charwidth:5,a,b,c) is equivalent to paste(a,b,c,sep:",", charwidth:5).

paste(arg,multiline:T) has somewhat different behavior. The result is much as before, except, if arg is other than a row vector, the result is a CHARACTER vector, with one element for each value of the first dimension that is greater than 1. Thus, if arg is a matrix or vector, each element of the output is a character representation of a row of arg. When you use 'multiline:T', there must be exactly 1 non-keyword argument. If arg is LOGICAL, it is first translated to REAL with True and False becoming 1 and 0, respectively.

paste(arg,multiline:T,linesep:Char), where Char is a string with just one character such as ";" or "\n" (the end-of-line character), combines the rows into a single CHARACTER scalar with each row separated by Char. If arg is a row vector (just 1 row), Char is ignored.

Along with 'multiline:T', you can use keywords 'sep', 'format', and 'missing', but not 'intwidth' and 'charwidth'. If it appears, the value for 'sep' must be a string with just one character, for example, " " (the default) or "\t". Any leading or trailing blanks in each numerical field are trimmed off when 'sep' is used.

```
Function paste() does not recognize keyword 'nsig'.

Examples:
  Cmd> paste(sep:"-","tick","tock",sep:", ","ding",sep:"-","dong")
  (1) "tick-tock, ding-dong"

The first 'sep:"-"' could also come at the end.

  Cmd> paste("PI is",PI,format:".10f")
  (1) "PI is 3.1415926536"

  Cmd> paste(format:"10.2f",sqrt(2),format:".15g",sqrt(2))
  (1) "      1.41 1.4142135623731"

  Cmd> paste("Blocks",DF[2],SS[2],SS[2]/DF[2],format:"7.3f",\
        (SS[2]/DF[2])/mse,charwidth:8,format:"13.6g",intwidth:2)
  (1) "Blocks    4       48.0368       12.0092  18.782"

  Cmd> paste(x, multiline:T) # 3 by 5 matrix x
  (1) "10.322 9.5278 10.636 10.411 9.6343"
  (2) "9.9979 10.606 8.1604 MISSING 8.5926"
  (3) "8.6147 11.212 9.4683 7.7964 10.489"

  Cmd> paste(x,multiline:T,linesep:"\n",sep:",",\
        missing:"-99",format:"0.4f")
  (1) "10.3222,9.5278,10.6357,10.4106,9.6343
  9.9979,10.6057,8.1604,-99,8.5926
  8.6147,11.2120,9.4683,7.7964,10.4889
  "

See also print().
```

2.205 plot

Keywords: plotting

Usage: plot(x,y [,impulse:T, lines:T] [other graphics keyword phrases]), where x is a REAL vector or scalar, y is a REAL vector or matrix

plot(x,y) makes a scatterplot of the data in vector x and vector or matrix y using characters such as asterisks or diamonds as plotting symbols. If y has several columns, they are plotted with symbols asterisk, diamond, cross, square, X, triangle, asterisk, dot, small cross, diamond,..., thereafter cycling through the plotting symbols.

plot(Struc), where Struc is a structure with at least two REAL components, is equivalent to plot(Struc[1], Struc[2]). Thus plot(x,y) and plot(structure(x,y)) are equivalent. Any components beyond the first two are ignored.

```
plot(graph,x,y) or plot(graph,Struc), where graph is a GRAPH variable,
draws the plot encapsulated in graph, adding to it the new information.
See topic graphs for details on adding information to a plot.

plot(x,y,impulses:T) makes an "impulse" plot of y vs x, drawing vertical
lines from the x = 0 line to each point.  If y has several columns, a
different line type is used for each column.  However, since the lines
will probably be superimposed, it may be hard to interpret the resulting
plot.

If keyword phrase lines:T' is an argument after y, the points will be
connected by lines similarly to lineplot().

If option 'dumbplot' has been set False (see setoptions()), the plot
will be a low resolution plot unless 'dumb:F' is an argument.

See topic graphs for the use of a scalar or length 2 vector for x, and
for the use of keywords title, xlab, ylab, xmin, xmax, ymin, ymax,
xaxis, yaxis, keep, show, dumb, new, file, linetype and thickness.

Use addchars(), addlines(), and addpoints() to add information to a
plot.  Use showplot() to re-display a plot.

Example:
  Cmd> plot(yhat1:yhat[,1],resid1:RESIDUALS[,1],\
       title:"Residuals vs yhat")
  Cmd> plot(X:1,run(20)^(.2*run(5)'),ylab:"Powers of X",\
       title:"X^.2, X^.4, X^.6, X^.8, and X", file:"ps.out",new:T)

See also chplot(), lineplot(), addpoints(), addlines(), addchars(),
showplot(), colplot, rowplot, tek, tekx, vt, vtx.
```

2.206 poisson

Keywords: glm, regression, categorical data

Usage:

```
poisson([Model] [, print:F or silent:T, incr:T,
offsets:vec, pvals:T, maxit:m, epsilon:eps, coefs:F]),
vec a REAL vector, m an integer > 0, eps REAL > 0
```

```
poisson(Model) computes a log linear regression fit of the model
specified in the CHARACTER variable Model.  If y is the response
variable in the model it must be a REAL vector with y[i] >= 0.
Estimation is by maximum likelihood on the assumption that y[i] is
Poisson.  If any y[i] is not an integer a warning message is printed.

See topic 'models' for information on specifying Model.

poisson() sets the side effect variables RESIDUALS, WTDRESIDUALS, SS,
DF, HII, DEPVNAME, TERMNAMES, and STRMODEL.  The elements of
WTDRESIDUALS are the final weighted residuals in the iteratively
```

reweighted least squares fit to log(response).

If, say, Model is "y=x1+x2", an iterative algorithm fits log(y) as a linear function of x1 and x2. A two line Analysis of Deviance table is printed, with line 1 the diffence between the deviance from a model with all coefficients 0 and the deviance of the estimated model, and line 2, labeled "ERROR", the deviance of the estimated model. Under appropriate assumptions, the latter can be used to test the goodness of fit of the model.

poisson(Model,inc:T) computes the full Poisson model and all partial models -- only a constant term, the constant and the first term, and so on. It prints an Analysis of Deviance table, with one line for each term, plus the deviance of the complete model labeled as "ERROR" . Each term's deviance is the reduction in deviance associated with that term.

If you omit Model (poisson()), the model from the most recent GLM command such as poisson() or anova(), or the model in CHARACTER variable STRMODEL is assumed.

Computations are carried out using iteratively reweighted least squares.

Other keyword phrases

Keyword phrase	Default	Meaning
maxiter:m	50	Positive integer m is the maximum number of iterations that will be allowed in fitting
epsilon:eps	1e-6	Small positive REAL specifying relative error in objective function (2*log likelihood) required to end iteration
print:F	T	Suppress all output except warning and error messages. Side effect variables are set.
silent:T	F	Suppress all output except error messages. Side effect variables are set.
coefs:F	T	Suppresses computation of quantities necessary for coefs() and secoefs() to compute estimated coefficients and their standard errors. Thus it effectively disables coefs() and secoefs() as well as some of the options available in modelinfo().
offsets:OffVec	none	Causes model to be fit to log(p) to be 1*Offvec + Model, where OffVec is a REAL vector the same length as response y. Note OffVec is in log units.

The default value for pvals can be changed by setoptions(pvals:T). See setoptions().

When OffVec a linear combination of X-variables in the model with specified coefficients, the overall model deviance from poisson(Model,n:Denom, offsets:OffVec) tests the hypothesis that the true coefficients are the same as the specified ones. Thus, if the elements of y are independent poisson random variables, the error deviance from

```
    poisson("y=1",offsets:rep(log(lambda0),length(y)))
```

is a test statistic for H0: elements of y have identical expectation lambda0.

poisson(Model,...) is equivalent to glmfit(Model,dist:"poisson", link:"log",...).

2.207 polyroot

Keywords: time series, complex arithmetic
Usage: `polyroot(coefs), coefs a REAL matrix`

polyroot(Coef) computes the real and possibly complex roots of the polynomials specified by the columns of REAL matrix Coef. If c[i] is coef[i,j], then the polynomial whose roots are found is

```
  x^n - c[1]*x^(n-1) - c[2]*x^(n-2) - ... - c[n-1]*x - c[n],
```

where n = nrows(Coef). Note that the leading coefficient (of x^n) is 1, and the coeficients are associated with descending powers of x.

If Coef is n by m, the result returned is a n by 2*m matrix with the real and imaginary parts of the roots associated with column j of Coef in columns 2*j-1 and 2*j, that is in the standard fully complex form. See topic complex.

To find the roots of a polynomial d[1]*x^n+d[2]*x^(n-1)+...+d[n]*x+ d[n+1], use polyroot(-d[-1,]/d[1,]).

The form of the argument to polyroot is adapted to its use in evaluating autoregressive and moving average operators. If phi is a REAL vector and x is a vector of white noise, autoreg(phi,x) generates a stationary autoregressive series if and only if the roots computed by polyroot(phi) are inside the unit circle, that is, max(creal(cpolar(polyroot(phi)))) < 1. Similarly, movavg(theta,x) generates an invertible moving average model if and only if all the roots computed by polyroot(theta) lie inside the unit circle.

See also autoreg(), movavg(), complex.

2.208 power

Keywords: probabilities, glm, anova
Usage: power(noncen,ngrp,alpha,nrep [,design:"rbd"]), noncen >= 0, 0 < alpha < 1, integers ngrp > 0 and nrep > 0; some or all arguments may be vectors

power(noncen,ngrp,alpha,nrep) computes the power of an F-test with significance level alpha in a balanced one-way analysis of variance (completely randomized design) for ngrp groups of size nrep or ngrp treatments with nrep replications with noncentrality parameter noncen. The noncentrality parameter is the sum of the squared treatment effects divided by the error variance. Note that this differs from the definition of the noncentrality parameter for power2().

power(noncen,ngrp,alpha,nrep,design:"rbd") computes power for a randomized block design with nrep blocks and ngrp >= 2 treatments.

power(noncen,1,alpha,n) computes the power of a single-sample two-tail t-test of H0: mu = 0 when mu = sqrt(noncen).

Some or all of the arguments of power may be vectors, in which case all non-scalars must be the same length, which will also be the length of the result. Thus you can compute the power of randomized block designs with 2 to 20 blocks, k treatments and noncentrality parameter 2 by

```
    power(2, k, .05, run(2,20),design:"rbd")
```

This is exactly equivalent to

```
    power2(2*run(2,20),k-1,.05,(k-1)*(run(2,20) - 1)
```

See also power2() and samplesize().

2.209 power2

Keywords: probabilities, glm, anova, regression
Usage: power2(noncent2,numDF,alpha,denomDF), noncent2 >= 0, 0 < alpha < 1, numDF >0, denomDF > 0; some or all arguments may be vectors

power2(noncen2,numDF,alpha,denomDF) computes the power for an F test with numDF numerator degrees of freedom, denomDF denominator degrees of freedom, a significance level of alpha, and a noncentrality parameter of noncen2. The noncentrality parameter is the sum of (i-th sample size times the squared effect of the i-th treatment) divided by the error variance. Alternatively,

```
   noncen2 = numDF*(E[Numerator MS]/E[denominator MS] - 1);
```

Some or all of the arguments of power2 may be vectors, in which case all non-scalars must be the same length, which will also be the length of the result. Thus you can compute a power as a function of noncen2 by, say

```
   power2(.2*run(0,100),10,.05,20)
```

Note that power2 uses a different definition of noncen than does power() or samplesize(). For example, you will get the same answer from

```
  Cmd> power2(nrep*noncen,ngrp-1,alpha,(nrep-1)*ngrp)
```

and

```
  Cmd> power(noncen,ngrp,alpha,nrep)
```

If nrep was computed as samplesize(noncen,ngrp,alpha,pwr), the value of power2(nrep*noncen,ngrp-1,alpha,(nrep-1)*ngrp) should be approximately equal to pwr.

power2() is useful for interaction and related effects where the error degrees of freedom are not simply sample sizes minus 1.

See also power() and samplesize().

2.210 predtable

Keywords: glm, anova

Usage: predtable() or predtable(Term), Term a CHARACTER scalar of the form "A.B. ...", where A, B are factors in current GLM model, or term:k, where k is a positive integer.

predtable() computes a table of fitted values (estimated cell expected values) based on the computations of the most recent GLM (generalized linear or linear model) command such as anova() or poisson(). The table has a dimension for each factor in the model, in the order the variables appear in the model. If there are variates in the model, the fitted values are computed with each variate set to its unweighted mean value and thus are what are sometimes called the covariate adjusted cell means.

predtable(Term) where Term is a quoted string or CHARACTER scalar of the form "Name1.Name2.Name3....", where Name1, Name2, ... are names of factors in the current GLM model, returns an estimated marginal table for the specified factors. If there are k factor names in Term, the value will be an array with k dimensions (vector if k = 1, matrix if k = 2), with the dimensions ordered in the same order as in Term, not the order they appear in the model if that is different. You cannot use predtable(term [,...]) after anova() with a balanced design unless Term includes all the factors in the model.

Example:

```
  Cmd> predtable("a.b") # same as predtable("b.a")'
```

predtable(term:k) is equivalent to predtable(TERMNAMES[k]), computing the marginal table matching term k in the model.

```
Example:
  Cmd> predtable(term:3)
```

predtable() and predtable(Term) are equivalent to glmtable(seest:F) and glmtable(Term, seest:F). You can use keywords phrases wtdmeans:T, x:values, estimate:F, seest:F and sepred:T as described under glmtable(). You can't use sepred:T when estimating marginal means.

For GLM functions involving a binomial response variable (logistic(), probit(), glmfit() with dist:"binomial"), the values computed are the estimated probabilities p of "success" associated with each cell. In this case, you can also use keyword phrase n:N, where N is a REAL variable, to specify the number of trials for each cell. N can be a scalar, a vector whose length matches the size of the table, or a matrix or array whose dimensions match those of the table. The resulting table is a table of N*p.

```
Example:
  Cmd> logistic("y=a+b",n:100); predtable(n:100)
```

Caution: When the marginal table for any term in the model contains empty cells, especially when a factor is nested in another with different numbers of levels, the estimated means may not be what you want.

After fitting a non-linear model by logistic(), probit(), poisson(), or glmfit(), when Term doesn't contain all the factors in the model, predtable(Term) first computes the estimated marginal table in the linear scale (logit, probit, or log) and then transforms it back into the scale of the response. This means that the computed marginal table is not the marginal means of the fitted table. For example, if b is a factor with 3 levels, after logistic("y=a*b", n:40), sum(predtable("a.b"))/3 is not the same as predtable("b").

When keyword phrase coefs:F was an argument on the most recent GLM command, predtable() is not available.

See also anova(), anovapred, glmpred(), glmtable(), regpred(), glm.

2.211 print

Keywords: output, missing values

Usage: print(a, b, ...[,format:Fmt or nsig:m, header:F, labels:F, missing:missStr, name:setName, width:w, height:h] [, file:fileName [,new:T]]]), Fmt, missStr, fileName, setName CHARACTER scalars, m > 0, w >= 30, h >= 12 integers

print(a,b, ...) prints objects (variables, expressions, macros) a, b, ... By default, print() formats REAL items using the format identified by 'format' on getoptions() output. This normally is floating point

with 5 significant digits. Objects that are macros or CHARACTER variables are printed enclosed in quotes (""), with any internal quotes escaped with '\' (for example "\"Hello\"") and non-printable characters printed as escaped octal integers (for example, "\033" or "\177")

print(string) where string is a single quoted argument or CHARACTER variable prints string just as it is, without enclosing quotes and without any internal quotes and non-printable characters escaped. This is useful for printing messages in a macro.

print(a,b,...,file:FileName [,new:T]) where FileName is a quoted string or CHARACTER variable, writes the output to the specified file rather than to the screen. If new:T is an argument, any information in the file is discarded before writing. Otherwise, output is appended to the end of the file. If FileName is the variable CONSOLE or a CHARACTER variable whose value is "CONSOLE", the output is written to the screen rather than to a file. The value of variable CONSOLE is ignored.

Keywords 'nsig', 'format', 'name', 'header', 'labels', and 'missing' are all recognized and can appear more than once. They affect the printing of objects that follow them, until they are changed except that a value for 'name' is used only once. Any of them that follow all items to be printed are treated as coming before all items. Thus, for example,

```
  Cmd> print(x,nsig:5,y,nsig:10)
```

and

```
  Cmd> print(nsig:10,x, nsig:5,y)
```

are equivalent. This does not apply to keywords 'file' and 'new' which can appear only once.

print(nsig:d,a,b,...) or print(a,b,...,nsig:d) prints numbers with d significant digits in floating point format with width d+7.

print(format:Fmt,a,b,...) or print(a,b,...,format:Fmt), where Fmt is a quoted string or CHARACTER variable, prints numbers according to specifications given in Fmt. Fmt must be of the form "w.df" or "fw.d" (fixed point) or "w.dg" "gw.d" (floating point) where w (field width) and d (decimals or significant digits) are integers, for example "6.3f" or "g15.7". See below for details.

print(name:charVar, a, b...) prints a with the name specified by quoted string or CHARACTER scalar charVar on the header. Alternatively, if the name is a legal MacAnova name no more than 10 characters long you can use a keyword to specify the name. Thus print(name:"Residuals",r) and print(Residuals:r) are equivalent. A name specified using keyword 'name' is used for only one output variable; however, you can have several instances of name:charVec, each affecting the next variable output.

print(header:F,a,header:T,b,...,) prints a without an identifying name, and b, ... with one.

print(a,b,...,width:w) temporarily sets option 'width' to w, an integer >= 30. This affects how many items are printed per line.

print(a,b,...,height:h) temporarily set option 'height' to h, an integer >= 12. This affects the number of lines in any graphs being printed as "dumb" plots and how often output will be paused.

print(labels:F,a,labels:T,b,...,) suppresses printing row and column labels of a, and enables their printing for b. When an object printed does not have coordinate labels, the labels printed are just the index or indices of the first element in each line. See labels.

print(missing:MissStr1,a,b,...), where MissStr is a quoted string or CHARACTER variable such as "?" or "NA", specifies that all missing values are to be printed using MissStr. If 'missing' is not used, missing values are printed as "MISSING" (or using a different default if you changed it by setoptions()). Note that this differs from the use of 'missing' on matprint() and matwrite() for which the value must be a REAL scalar.

print(x,file:FileName,new:T,header:F,labels:F,missing:"?") writes x to the file in a form that can be read by vecread().

Details on value of 'format' keyword

If Fmt is "w.df" or "fw.d" (fixed point), or "w.dg" or "gw.d" (floating point), integer w specifies a field width of at least w characters. For fixed point format, integer d is the number of digits that will follow the decimal point. For floating point format, d is the number of significant digits printed. If w is omitted (".3f" / "f.3" or ".7g" / "g.7"), it is implicitly set to d+7 ("10.3f"/ "f10.3" or "14.7g" / "g14.7"). If w > 27, width = 27 is assumed and if d > 20, digits = 20 is assumed.

With fixed point output, trailing zeros are kept; for floating point output they are trimmed off. For example, 10.30 is printed as '10.30000' with "8.5f" or "f8.5" format and as '10.3' with "8.5g" or "g8.5" format.

For floating point output, exponential form, 9.3e+07 for example, is used if required to represent the number.

You can change the default format for print(), fprint(), and matprint() by setoptions() using setoptions keywords 'nsig' or 'format'.

Examples:

print(nsig:5,a), print(format:"12.5g",a), and print(a,nsig:5), are equivalent.

print(nsig:5,file:"myfile",a,new:T) writes a to file "myfile", starting fresh.

```
Cmd> print("Quoted because > 1 argument",vector("a","Escaped\1\2"))
STRING:
(1) "Quoted because > 1 argument"
VECTOR:
(1) "a"
```

```
  (2) "Escaped\001\002"

  Cmd> print("Not quoted because only 1 argument")
  Not quoted because only 1 argument
```

Other keywords are used to label output. Thus the output produced by 'print(eigenvals(x))' will be labeled 'VECTOR', but that produced by 'print(Values:eigenvals(x))' will be labeled 'Values'. Keywords may have no more than 10 characters.

This function is particularly useful inside macros and compound lines.

See also setoptions(), write(), fprint(), fwrite(), matprint(), matwrite(), paste(), error().

2.212 probit

Keywords: glm, regression, categorical data
Usage: probit([Model], n:Denom [, print:F or silent:T, incr:T, offsets:vec, pvals:T, maxit:m, epsilon:eps, coefs:F]), Denom REAL scalar or vector > 0, vec a REAL vector, m an integer > 0, eps REAL > 0

probit(Model,n:Denom) computes a probit regression fit of the model specified in the CHARACTER variable Model. If y is the response variable in the model it must be a REAL vector with y[i] >= 0. Denom must either be an REAL scalar >= max(y) or a REAL vector of the same length as y with Denom[i] >= y[i]. Estimation is by maximum likelihood on the assumption that y[i] is binomial with Denom[i] trials (Denom trials for scalar DENOM). If any y[i] or n[i] is not an integer a warning message is printed.

See topic 'models' for information on specifying Model.

probit() sets the side effect variables RESIDUALS, WTDRESIDUALS, SS, DF, HII, DEPVNAME, TERMNAMES, and STRMODEL.

If, say, Model is "y=x1+x2", an iterative algorithm is used to predict invnor(E[y/Denom]) = probit(E[y/Denom]) - 5 as a linear function of x1 and x2. A two line Analysis of Deviance table is printed. Line 1 is the difference 2*L(1) - 2*L(0), where L(0) is the log likelihood for a model with all coefficients 0 (all probabilities = 0.5) and L(1) is the maximized log likelihood for the model fit. Line 2 is 2*L(2) - 2*L(1) where L(2) is the maximized log likelihood under a model fitting one parameter for every y[i]. Under appropriate assumptions, the latter can be used to test the goodness of fit of the model using a chi-squared test.

To get the coefficients for a classic probit analysis, you should increase the estimated constant by 5.

probit(Model,n:Denom,inc:T) computes the full probit model and all partial models -- only a constant term, the constant and the first term, and so on. It prints an Analysis of Deviance table, with one line for each term, representing a difference 2*L(i) - 2*L(i-1) where L(i) is the maximumized log likely for a model including terms 1 through i, plus the deviance of the complete model labeled as "ERROR1". Each line except the last can be used in a chi-squared test to test the significance of the term on the assumption that the true model includes no later terms.

If you omit Model (probit(,n:Denom ...)), the model from the most recent GLM command such as poisson() or anova(), or the model in CHARACTER variable STRMODEL is assumed.

Computations are carried out using iteratively reweighted least squares.

Other keyword phrases

Keyword phrase	Default	Meaning
maxiter:m	50	Positive integer m is the maximum number of iterations that will be allowed in fitting
epsilon:eps	1e-6	Small positive REAL specifying relative error in objective function (2*log likelihood) required to end iteration
print:F	T	Suppress all output except warning and error messages. Side effect variables are set.
silent:T	F	Suppress all output except error messages. Side effect variables are set.
coefs:F	T	Suppresses computation of quantities necessary for coefs() and secoefs() to compute estimated coefficients and their standard errors. Thus it effectively disables coefs() and secoefs() as well as some of the options available in modelinfo().
offsets:OffVec	none	Causes model to be fit to probit(p) to be 1*Offvec + Model, where OffVec is a REAL vector the same length as response y. Note OffVec is in probit units.

The default value for pvals can be changed by setoptions(pvals:T). See setoptions().

When OffVec a linear combination of X-variables in the model with specified coefficients, the overall model deviance from probit(Model,n:Denom, offsets:OffVec) tests the hypothesis that the true coefficients are the same as the specified ones. Thus, if the elements of y are independent binomial random variables with n = 10, the error deviance from

```
Cmd> probit("y=1",10,offsets:rep(invnor(p0),length(y)))
```

is a test statistic for H0: elements of y/10 have identical expectation p0.

probit(Model,n:n,...) is equivalent to glmfit(Model,n:n,dist:"binomial", link:"probit",...).

2.213 prod

Keywords: summary statistics
Usage: prod(x), x REAL or LOGICAL or a structure with REAL or LOGICAL components. prod(x1,x2,...), x1, x2, ... REAL or logical vectors, all the same type.

prod(x) computes the product of the elements of a REAL or LOGICAL vector x. If x is LOGICAL, True is interpreted as 1 and False as 0 and hence prod(x) has value 1 if all elements of x are True and 0 if any is False.

If x is a m by n matrix, prod(x) computes a row vector (1 by n matrix) consisting of the product of the elements in each column of x.

If x is an array with dimensions n1, n2, n3, ..., y <- prod(x) computes an array with dimensions 1, n2, n3, ... such that y[1,j,k,...] = prod(x[i,j,k,...], i=1,...,n1). This is consistent with what happens when x is a matrix. Note: MacAnova3.35 and earlier produced a result with dimensions n2, n3, ...

prod(NULL) is NULL.

prod(a,b,c,...) is equivalent to prod(vector(a,b,c,...)) if a, b, c, ... are all vectors. They must all have the same type, REAL or LOGICAL or be NULL. prod(NULL, NULL, ..., NULL) is NULL.

If all the elements of a vector x are MISSING, prod(x) is 1.0.

If x is a structure, prod(x) computes a structure, each of whose components is prod() applied to that component of x.

Example:

If x is a n by m matrix, r <- prod(x)/x computes a matrix with r[i,j] = product of all x[l,j] with l != i. See arithmetic.

If x is a n by 4 by 5 array, r <- prod(x)/x computes an array with r[i,j,k] = the product of all r[l,j,k] with l != i. That is, it treats x analogously to a 4 by 5 array of vectors of length n. See arithmetic.

See also sum().

2.214 putascii

Keywords: output
Usage: putascii(vec [,keep:T]), vec a vector of integers > 0 and <= 255 putascii(vec, file:fileName [, new:T])

putascii(Vec) prints the characters corresponding to the elements of the vector Vec, considered as ASCII codes. All the codes must be integers between 1 and 255, inclusive. Thus putascii(run(64,126)) prints
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~

putascii(Vec1,Vec2,...) is equivalent to putascii(vector(Vec1,Vec2, ...)), if Vec1, ..., are REAL vectors.

The primary use of putascii() is to send control sequences to a terminal. See macro vt for an example of its use. On all machines, any leading 7's are explicitly translated to beeps or bells. Thus
```
  Cmd> putascii(vector(7,7,7,69,82,82,79,82))
```
rings the bell three times and prints ERROR.

putascii(Vec,file:FileName [,new:T]) or putascii(Vec1, Vec2, ..., file:FileName [,new:T]), where FileName is a CHARACTER variable or quoted string, writes the ASCII codes on file FileName. If new:T is an argument, any information in the file will be destroyed; otherwise the codes are added at the end of the file.

putascii(Vec,keep:T) or putascii(Vec1,Vec2,...,keep:T) returns a CHARACTER variable whose characters have the ASCII codes. Thus, for example, putascii(run(4,8), keep:T) has value "\004\005\006\007\010". Use of new:T with file:FileName is illegal.

See also print(), write(), fprint(), fwrite().

2.215 qr

Keywords: matrix algebra
Usage: qr(x [,pivot:T, ronly:T]), x a REAL matrix

qr(x) computes the elements of a QR decomposition of matrix x as a structure with two components 'qr' and 'qraux' in the form computed by Linpack subroutine dqrdc. Component qraux is a vector of length p and component qr is a n by p matrix. The elements of qr on and above the diagonal constitute the upper triangular matrix R of the QR decomposition, and the remaining elements, together with the elements of qraux contain enough information to compute Q. No pivoting is performed with this form of qr.

qr(x,pivot:T) or simply qr(x,T) does the same computation, with possible reordering of columns. The result is a structure with three components, 'qr' and 'qraux' as before, plus 'pivot'. The latter is a vector

containing the column numbers in x corresponding to the successive columns of q and r.

qr(x,ronly:T) returns a p by p upper triangular matrix consisting of the R matrix in the QR decomposition, computed without pivoting. Thus both parts of the QR decomposition can be computed by Q <- x %*% solve(R <- qr(x,ronly:T)). The columns of Q are orthogonal and x - Q %*% R will be zero except for rounding error.

See also cholesky().

2.216 quitting

Keywords: general
Usage: quit or bye or end or stop or exit quit(F) or bye(F) or end(F) or stop(F) or exit(F)

To terminate a MacAnova run, type 'quit'. On a version with Windows (Macintosh, Windows or Motif), you will be asked if you want to save the workspace and any command/output windows.

quit() or quit(T) has the same effect as 'quit'.

quit(F) means you want to quit unconditionally with no opportunity to save the workspace or windows. On a version without Windows, it is no different from quit(T).

It is an error for quit to be part of a compound command (enclosed in {...}) or for there to be anything other than a comment starting with '#' after it on the command line.

'stop', 'end', 'bye' and 'exit' mean the same as 'quit'.

Example:

```
  Cmd> quit  # or bye or stop() or end(T), etc.
```

In a windowing version (Macintosh, Windows or Motif), you can also select Quit on the File Menu. You will be asked whether you wish to save the workspace or the command/output window(s). On a Macintosh, if you hold down the Option key while selecting Quit or hitting Return after typing 'quit', you quit unconditionally without a chance to save files.

See also launching.

2.217 rank

Keywords: ordering

Usage: sort(x [,down:T]), x REAL or CHARACTER or a structure with all REAL or all CHARACTER components.
rank(x [,down:T, ties:"ignore" or "average" or "minimum"]), x REAL or CHARACTER or a structure with all REAL or all CHARACTER components.

rank(x) computes the ranks of the data in variable x, the minimum value being assigned rank 1. In case of ties, the rank corresponding to the average of the ranks for the tied cases is computed. x can be REAL or CHARACTER.

When x is CHARACTER, x[i] is considered greater than x[j] if x[i] follows x[j] in alphabetical order using the ASCII collating sequence. See sort() for the explicit ordering of characters.

rank(x,down:T) or simply rank(x,T) acts the same except that the data are ranked in decreasing order, with rank 1 assigned to the maximum element.

rank(x,ties:Method) or rank(x,ties:Method,T or down:T) treats ties as described below. Method must be one of "average", "minimum" or "ignore" (or simply "a", "m" or "i"). rank(x) and rank(x,T) give the same results as rank(x, ties:"average") and rank(x, ties:"average",T), respectively. When x is CHARACTER, keyword 'ties' is ignored and is assumed to have value "i".

If x is a matrix, the result is a matrix each of whose columns contains the ranks of the elements of the corresponding column of x.

If x is an array with dimensions n1, n2, ..., the result is an array of the same size and shape with all the elements with fixed values of subscripts 2, 3, ... defining a "column" whose ranks are computed. An array with dimension > 2 is always treated as an array and not as a matrix, even if there are at most two dimensions greater than 1.

If x is REAL and there are MISSING values in a column, their rank is also MISSING and the maximum rank of the non-missing values is the number of non-missing values.

It is also acceptable for x to be a structure, whose non-structure components are all REAL or all CHARACTER. In that case, rank() returns a structure of the same form, each of whose non-structure components is the result of applying rank() to the corresponding component of x.

Treatment of Ties with REAL Data

Suppose k elements in a vector (column) are tied, that is they all have the same value and no other element has this value, and suppose the ranks these elements would have if their values were very slightly changed so as to break the ties while preserving other ordering would be r, r+1, r+2, ..., r+k-1. The following describes the ranks computed for

the tied values for each of the three possible methods.

Value for 'ties'	Computed ranks
"average" or "a" (default)	All ranks = (r+(r+1)+...+(r+k-1))/k = r+(k-1)/2
"minimum" or "m"	All ranks = r
"ignore" or "i"	r, r+1, r+2, ..., r+k-1; which ranks go in which position is unpredictable

Examples: Suppose x is vector(27,22,25,26,22,21,?,24) (1 tie)
rank(x) and rank(x,ties:"a") compute vector(7,2.5,5,6,2.5,1,?,4)
rank(x,down:T) and rank(x,T) compute vector(1,5.5,4,3,5.5,7,?,4)
rank(x,ties:"m") computes vector(7,2,5,6,2,1,?,4)
rank(x,ties:"i") computes vector(7,2,5,6,3,1,?,4)

See also grade(), sort().

2.218 rankits

Keywords: transformations, descriptive statistics, ordering
Usage: rankits(x [,ties:"ignore" or "average" or "minimum"]),
x REAL or a structure with REAL components.

rankits(x) computes the vector of rankits (normal scores) for the data in the REAL vector x. An important use is plot(rankits(x),x) which produces a rankit or normal scores plot of the values in x. What is computed is equivalent to

```
invnor((rank(x,ties:"ignore") - .375)/(n + .25))
```

where n is the number of non-MISSING values. The value corresponding to a MISSING value is MISSING.

rankits(x,ties:method), where method is "ignore", "average", or "minimum" (or "i", "a", "m") computes

```
invnor((rank(x,ties:method) - .375)/(n + .25))
```

See rank() for a detailed discussion of the three methods. It is hard to think of a situation when you would want to use "minimum" with rankits().

If x is a matrix, the result is a matrix each of whose columns contains the rankits for the corresponding column of x.

If x is an array, rankits(x) is an array of the same size and shape with all the elements with fixed values of subscripts 2, 3, ... defining a "column" whose rankits are computed. An array with dimension > 2 is always treated as an array and not as a matrix, even if there are at most two dimensions greater than 1.

It is also acceptable for x to be a structure, whose non-structure components are all REAL. In that case, rankits() returns a structure of the same form, each of whose non-structure components is the result of applying rankits() to the corresponding component of x.

See also halfnorm().

2.219 rational

Keywords: transformations
Usage: rational(x, a, b) or rational(x, a) or rational(x,,b), x REAL or a structure with REAL components, a and b REAL vectors

rational(x,a,b) computes a rational function of the REAL vector, matrix, or array x, with coefficients for the numerator and denominator polynomials in REAL vectors a and b. The result is REAL with the same size and shape as x. If a or b is omitted, it is construed as representing the constant 1. For example, rational(x,a), rational(x,a,), and rational(x,a,1) are equivalent and compute a polynomial in x, and rational(x,,b), and rational(x,1,b) are equivalent and compute the reciprocal of a polynomial in x.

The rational function computed is
(a[1] + a[2]*x + a[3]*x^2+ ...)/(b[1] + b[2]*x + b[3]*x^2+ ...) .

x can also be a structure, all of whose non-structure components are REAL, in which case the result is a similar structure.

An important use of rational() is in writing macros to compute mathematical functions that are not directly available in MacAnova but which can be approximated using rational functions. For example

```
Cmd> cumnor1 <- macro("@x <- $1; @posx <- @x > 0
@posx - (@posx - .5)*rational(abs(@x),1,\\
   vector(1,.196854,.115194,.000355,.019527))^4", dollar:T)
```

creates macro cumnor1 which uses an approximation due to Hastings to cumpute cumulative normal probabilities. See also macros i0 and i1 in file MacAnova.mac distributed with MacAnova. See topics macro() and macros.

2.220 rbin

Keywords: random numbers
Usage: rbin(N, n, p), N positive integer, n scalar or vector of positive integers, p scalar or vector of probabilities.

rbin(N, n, p) returns a vector of N independent binomial pseudo-random variables with sample size n and probability p. N must be a positive integer.

n must be a positive integer scalar or a vector of N positive integers. p must be a REAL scalar between 0 and 1 or a vector of N values between 0 and 1. If n or p is a scalar, it is used for every element of the result. Otherwise, n[i] and/or p[i] are used for the i-th element of the result.

If the random number generator has not been initialized by setseeds(), setoptions() or previous use of rbin(), rnorm(), rpoi() or runi(), the generator's "seeds" will be initialized automatically using the current time and date, and their values will be printed out.

rbin() can be used to generate other random variables by using a random vector as p and/or n. For example,

```
Cmd> y <- rbin(N, n, invbeta(runi(N),a, b)) # a, b > 0
```

will generate a pseudo-random beta-binomial sample variables with parameters n, a and b. See the User's Guide for details.

The generation algorithm is adapted from Voratas Kachitvichyanukul and Bruce Schmeiser, "Binomial random variate generation", Commun.ACM, 31 (1988) 216-222, published as Algorithm 678, Trans. Math. Software 15, 394-397.

See also setseeds(), getseeds(), setoptions(), rnorm(), rpoi(), runi(), invbeta(), cumbin().

2.221 read

Keywords: macros, input, files, missing values

Usage: x <- read(fileName, setName or macroName [,quiet:T or F, echo:T or F, labels:Labels, silent:T, notfoundok:T]), fileName and setName CHARACTER scalars; fileName can also be of the form string:charVal where charVal is a CHARACTER scalar or vector.

read() can be used instead of either matread() and macroread(). It recognizes the same keywords. In fact, the only difference from matread() and macroread() is that matread() gives a warning message when reading a macro and macroread() gives a warning message when reading a data set. read() makes no such complaint. See matread() and macroread() for details.

See also getdata, getmacros, vecread(), readcols, data files, macro files.

2.222 readcols

Keywords: input, files
Usage: readcols(FileName, name1, name2, ..., namek[, Keyword phrases])
readcols(FileName, vector("name1", ..., "namek")[, keyword phrases])
FileName a quoted string or CHARACTER scalar, name1, ... unquoted variable names. Keyword phrases may be quiet:T, echo:T, skip:skipChar, or stop:stopChar. FileName can also be of the form string:charVal where charVal is a CHARACTER scalar or vector.

readcols(FileName,name1,name2,...,namek) uses vecread() to read numerical data from file FileName and puts the columns in variables name1, name2, ..., namek which should not be quoted. The file should consist of k columns of numbers separated by spaces, commas or tabs, with MISSING values indicated by '?' or '.'. See data files and vecread() for a complete description of the file format, including skip and stop characters.

readcols(FileName,varNames), where varNames is a CHARACTER vector of the form vector("name1",...,"namek") is an alternative usage.

For both usages, FileName must be a quoted string or CHARACTER scalar and the number of variable names provided must divide the total number of data values. In a version with windows (Macintosh, Windows, Motif), if FileName is "", you can select the file using a dialog box.

Filename can also be for the form string:CharVector. The CHARACTER vector CharVector is "read" instead of a file. See vecread().

readcols recognizes the same keyword phrases as vecread(). These are skip:skipChar, stop:stopChar, quiet:T or F, and echo:T or F. See vecread().

Examples:

```
  Cmd> readcols("halddata",x1,x2,x3,x4,y)
  Cmd> readcols("halddata",vector("x1","x2","x3","x4","y"))
```

These are completely equivalent.

See also vecread(), macros.

2.223 redo

Keywords: control
Usage: redo() or redo(charVar) where charVar is CHARACTER scalar

redo() creates a macro REDO from the entire preceding command line

(automatically saved as variable LASTLINE) and then executes REDO(). Subsequently the same line can be re-executed one or more times by REDO().

redo(charVar) does the same, except macro REDO is created as macro(charVar), where charVar is a CHARACTER variable or quoted string.

Caution: do not attempt to use redo() immediately following a line containing redo() or REDO(), as this leads to uncontrolled recursion.

redo is implemented as a pre-defined macro.

See also edit, macros, syntax.

2.224 REDO

Keywords: control
Usage: REDO()

REDO() re-executes the command line re-executed by the most recent use of pre-defined macro redo. It is a macro created by redo and thus has no fixed text. See redo.

2.225 regcoefs

Keywords: glm, anova, regression, confidence intervals
Usage: regcoefs(Model [,pvals:T] [,byvar:F]) or regcoefs([pvals:T] [,byvar:F]), where Model is a CHARACTER scalar

regcoefs(Model) returns a matrix with appropriately labeled rows and columns of the regression coefficients, their standard errors and t-statistics from a least squares fit to the regression model specified by Model. There can be no factors in Model. If Model is omitted, the most recent GLM model is used.

regcoefs(Model,pvals:T) or regcoefs(pvals:T) also computes two-tail P values corresponding to the t-statistics on the basis of Student's t-distribution with degrees of freedom from the last element of side effect variable DF.

If the response variable is multivariate, the result is a structure, each of whose components is a labeled matrix of coefficients, standard errors and t-statistics. regcoefs(Model,byvar:F) or regcoefs(byvar:F) returns a single labeled matrix, with separate columns for the coefficients, standard errors, ... for each variable.

Because of the row and column labels, after any GLM command with a model

withut factors, typing regcoefs([pvals:T]) produces a table similar to that produced by regress(). After non-linear fits such as logistic() or poisson(), the P-values will not necessarily be appropriate.

See also glm, regress(), secoefs().

2.226 regpred

Keywords: glm, regression
Usage: regpred(vals), vals a REAL vector or matrix.

regpred(vals) computes the fitted (predicted) value, the standard error of estimation, and the standard error of prediction for the current regression model when the X variables take values given in the REAL vector or matrix vals.

The result is a structure with components 'estimate', 'SEest',and 'SEpred'. SEpred is missing except after regress(), anova() or manova() or their weighted versions.

If vals is a vector, it must have length nvars, where nvars is the number of independent variables in the model, not counting the constant, if any.

If vals is a matrix, it must be m by nvars, with rows corresponding to m vectors of values for the independent variabls and each component of the result is a vector of length m. Thus, after regress("y=x1+x2+x3") regpred(hconcat(x1,x2,x3)) computes the predicted values and standard errors at all the cases in the data set.

If the error degrees of freedom are 0, all standard errors are set to MISSING.

You can also use regpred() after non-regression GLM commands as long as there are no factors in the model. SEpred is omitted from the output.

After anova(), manova() and regress() regpred(vals) is equivalent to glmpred(vals,sepred:T). After other GLM commands, regpred(vals) is equivalent to glmpred(vals). You can use keyword phrases estimate:F, seest:F, sepred:F and n:N. See glmpred() for details.

See also regress(), anova(), glm, yhat.

2.227 regress

Keywords: glm, regression
Usage: regress([Model] [,print:F or silent:T, pvals:T, coefs:F, marginal:T])

regress(Model) performs a least squares fit of the regression model given in the quoted string or CHARACTER variable Model. It prints out the regression coefficients, their standard errors, and t statistics, plus other summary statistics (see below). If option pvals is True, regress() also prints P values for each coefficient based on Student's t distribution.

Examples (y, x, x1, x2, and x3 all REAL vectors of length 10):

regress("y = x")	Simple linear regression of y on x
regress("y = x - 1")	Linear regression through origin of y on x
regress("y = {run(10)}")	Simple linear regression of y on vector(1,2,3,4,5,6,7,8,9,10)
regress("y = x1 + x2 + x3")	3 variable multiple regression of y on x1, x2 and x3
regress("{sqrt(y)} = x + {x^2}")	Quadratic polynomial regression of sqrt(y) on x

regress(Model,weights:Wts) performs a weighted least squares fit, using REAL vector Wts as case weights. The elements of Wts must not be negative. The results are what you would get by multiplying the response vector and all independent variables, including the contant vector, by sqrt(Wts) and then doing a least squares fit.

No ANOVA table is printed by regress(). To see one, type 'anova()' as the next GLM command after regress().

Model is of the form "Response = Var1 + Var2 + ... + Vark", where Response, Var1, ..., Vark are either variable names or have the form {expr}, where expr is a MacAnova expression. All variables or evaluated expressions must be REAL with the same number of rows. The variables to the right of '=' must be vectors or n by 1 matrices. If any right hand side variable is actually a factor, it is treated as a quantitative variate whose values are the levels of the factor. The associated sum of squares had only 1 degree of freedom regardless of the number of levels of the factor. Regression through the origin is specified by including a "-1" in the model. See also models.

regress() or regress(,weights:Wts) with no model specified computes a least squares regression using the same model as was used by the most recent GLM command such as regress(), anova(), or poisson(). See glm.

Other printed output from regress() includes multiple R-squared, the overall F-statistic for the model excluding the constant term, the mean squared error and the Durbin Watson statistic.

Side effect variables are RESIDUALS, HII, SS, DF, DEPVNAME, TERMNAMES, STRMODEL, COEF, and XTXINV. When weights are used, RESIDUALS = response - fit and WTDRESIDUALS = sqrt(Wts)*RESIDUALS is an additional side effect variable. If any independent variables are of the form {expr}, the corresponding element of TERMNAMES will be "{expr}".

Coefficients and/or standard errors may be retrieved by coefs() or

secoefs().

Other Keywords		
Keyword phrase	Default	Meaning
print:F	T	Suppress all output except warning and error messages. Side effect variables are set.
silent:T	F	Suppress all output except error messages. Side effect variables are set.
pvals:T	F	Print P values. Default is T when option pvals is T.
marginal:T	F	Specifies that the elements of the side effect variable SS are computed marginally. When none of the X-variables are aliased, the computed SS are equivalent to SAS Type III SS. See topic glm for details. SS will be printed by anova() with no intervening GLM command.

Keyword phrase 'coefs:F' is not legal with regress().

See also anova().

2.228 regs

Keywords: glm, regression
Usage: regs(x,y), x and y REAL matrices with the same number of rows

regs(x,y) is a macro which computes the regression of y on the columns of x. Thus, if data is a n by 7 matrix, say, you can compute the regression of the last column on the first 6 by regs(data[,-7], data[,7]).

Macro regs creates temporary variables @X1, @X2, ... and @Y and then invokes regress(). If y has more than 1 column, regs invokes manova(). To see the ANOVA table, immediately follow regs by anova(). Note that because the model for regress() or manova() uses temporary variables, STRMODEL cannot be used as an implicit model to a subsequent regress, anova, or manova command.

2.229 releigen

Keywords: matrix algebra
Usage: releigen(h,e), h and e symmetric REAL matrices with no MISSING values, e positive definite.

releigen(h,e) computes an eigenvector/eigenvalue decomposition of the symmetric matrix h relative to the symmetric positive definite matrix e. The result is a structure with two components, 'values' and 'vectors', containing the relative eigenvalues and eigenvectors, respectively.

Suppose releigs has been computed as releigen(h,e) where both h and e are p by p. Then if u is releigs$vectors and v is releigs$values, u = is a p by p matrix with columns u1, u2, ..., up and v is a vector of length p with elements v1 >= v2 >= ... >= vp, such that h %*% uj = vj * e %*% uj. The uj's are normalized so that uj' %*% e %*% uj = 1, uj' %*% e %*% uk = 0, j != k. If dv is the diagonal matrix dmat(vector(v1, v2,...,vp)) then these properties are summarized by h %*% u = e %*% u %*% dv , u' %*% h %*% u = dv, u' %*% e %*% u = Ip. Ip is the identity matrix.

If h and e are MANOVA hypothesis and error matrices, respectively, the relative eigenvalues may be used to compute several standard multivariate hypothesis tests, and the elements of the relative eigenvectors are the coefficients of the MANOVA canonical variables associated with the hypothesis.

See also det(), eigen(), eigenvals(), trideigen() and releigenvals().

2.230 releigenvals

Keywords: matrix algebra
Usage: releigenvals(h,e), h and e symmetric REAL matrices with no MISSING values, e positive definite.

releigenvals(h,e) computes the eigenvalues of the symmetric matrix h relative to the symmetric positive definite matrix e. If h and e are p by p, then releigenvals(h,e) returns vector(v1,v2,...,vp) or relative eigenvalues with v1 >= v2 >= ... >= vp. See releigen() for a summary of the properties of relative eigenvlues.

See also det(), eigen(), eigenvals() and trideigen().

2.231 rename

Keywords: general
Usage: rename(var, newName), where newName is an undefined variable

rename(Var, Name) where Name is an undefined variable (no value has been assigned to it) changes the name of Var to Name. Var may be an existing variable, a constant such as 3.5, "hello", or T, or an expression such as 'sqrt(20)' or '3*cos(x) + 4*sin(x)'. When Var a constant or expression, rename(Var, Name) is equivalent to Name <- Var. Unless Name

is a CHARACTER scalar (see below), it must not be a function or an existing variable or macro.

rename(Var, nameVar) where nameVar is a quoted string or scalar CHARACTER variable is equivalent to rename(Var, <<nameVar>>), that is the new name is the value of nameVar. Thus, for example, rename(x, y) and rename(x, "y") are equivalent. The value of nameVar must not be the name of any existing variable, macro or function and must be a legal name (see syntax).

Var must not be a function or a "special" variable such as CLIPBOARD or SELECTION.

Examples:
  rename(PI, pi) and rename(PI,"pi") both change the name of PI to pi.
  pi <- 10; rename(PI, pi) is an error since pi is now a defined
  variable. rename(sqrt(2),sqrt2) is equivalent to sqrt2 <- sqrt(2),
  provided sqrt2 has not previously been defined.

You can achieve the same effect as rename(Var,Name) by
  Name <- Var; delete(Var)
However, if Var is not a constant or expression, doing it this way entails the temporary existence in memory of two copies of Var. If Var is large, there may not be enough room. The use of rename() avoids this problem. For this reason, in a macro that uses another macro to compute a value, it may sometimes be helpful to use rename() instead of assignment.

See also nameof(), compnames(), varnames()

2.232 rep

Keywords: combining variables, variables
Usage: rep(x, n) or rep(x, repFac), where x is REAL, LOGICAL, or CHARACTER, and n is an integer > 0 or repFac is a vector of integers >= 0

rep(X, n), where n is a positive integer, replicates the values in X n times to form a vector of length n*length(X). It is equivalent to vector(X,X,X,...,X), where there are n repetitions of X. X must be a REAL, LOGICAL or CHARACTER vector, matrix, or array.

rep(X, Repfac), where Repfac is a vector of integers with length(Repfac) = length(X) and Repfac[i] >= 0, replicates the j-th element of vector(X) Repfac[j] times. The result has length length(X)*sum(Repfac). At least one element of Repfac must be non-zero.

Examples:
  Cmd> rep(vector(1,3,5),4) # returns vector(1,3,5,1,3,5,1,3,5,1,3,5)
  Cmd> rep(vector(7,6,5),vector(3,0,2)) # returns vector(7,7,7,5,5)

```
  Cmd> a <- factor(rep(run(4),5)); b <- factor(rep(run(5),rep(4,5)))
computes factors suitable for a 4 by 5 design with no replications.
  Cmd> rep(run(5), vector(1,2,3))# ERROR! (args have different lengths)

WARNING: Because rep() is a function you cannot use the name 'rep' as a
variable name, say for a replication factor in a design.  Use 'Rep' or
'reps' instead.

See also vector(), run(), models, glm.
```

2.233 resid

Keywords: glm, residuals, anova, regression
Usage: `resid() or resid(Model)`

```
resid(), with no argument, computes a REAL matrix of various quantities
useful in the analysis of residuals.  It uses side effect variables
RESIDUALS, HII, etc. produced by the most recent GLM (generalized linear
or linear model) command such as regress(), anova(), or poisson().

It is an error if any of the needed side effect variables do not exist.

resid(Model) first executes manova(Model, silent:T) to compute the
required side effect variables before computing the residual-related
quantities.  Model should be a CHARACTER variable or string specifying a
linear ANOVA or MANOVA model.  Any factors in the model will be treated
as factors.  If you want them treated as variates, use resid(Model,T).

Each row of the result corresponds to a case.  When the dependent
variable Y is univariate, there are 5 columns, as follows:
   Col. 1   Y = observed response
   Col. 2   Studentized residuals = RESIDUALS/SE(RESIDUALS)
   Col. 3   HII = leverage
   Col. 4   Cook's distance
   Col. 5   t-statistics = externally studentized residuals =
            RESIDUALS/SE*(RESIDUALS), where SE* for each case is
            a standard error based on the model fit excluding that case.

When Y is multivariate of dimension p, there are 4*p + 1 columns -- the
p values for Y, the p standardized residuals, HII, the p Cook's
distances, and the p externally studentized residuals.

If a case has missing values, most entries for that case will be MISSING
and there are no useful numbers.

After non-linear GLM commands such as poisson() and logistic(), the
results are based on the last stage of the iteratively reweighted least
squares algorithm used to fit the model.  Residuals are standardized by
the error mean square in the linear scale.  They should still be valid
for diagnosing departures from the model.
```

The output of resid is modeled on what is printed by the 'resid' command in program Multreg.

resid is implemented as a pre-defined macro.

See also glm, yhat, resvsindex, resvsrankits, resvsyhat.

2.234 restore

Keywords: files, general
Usage: restore(FileName [,delete:F, list:T])

restore(FileName) restores the variables previously put in file FileName by save() or asciisave(), where FileName is a quoted string or CHARACTER variable. Unless option 'restoredel' has been set to False (see setoptions()), all current variables are deleted before restoration.

In a version with windows (Macintosh, Windows, Motif), if FileName is "" you will be prompted for the name of the file. Restore Workspace on the File menu is equivalent to 'restore("")'.

restore(FileName, delete:F) does not delete existing variables before restoring. However, a variable or macro with the same name as a variable or macro in the file is replaced by the one in the file. If option 'restoredel' has been set to False, it may be overridden by delete:T.

restore(FileName [,delete:F], list:T) lists information on variables as they are restored.

Unless options:F was used when the save file was created or the file was created by a partial save (for example, save("myfile",x,y,a)), values for options such as nsig, format, and seeds will be restored to the previous values at the time the save was done (see setoptions()).

If all:T was used when the file was created, then private information required by certain commands like secoefs() will be restored.

If MacAnova determines that the file being restored is a binary file that was created by save() in an incompatible version of MacAnova, the restore is aborted. Current variables are lost unless delete:F is an argument or option 'restoredel' has been set to False.

Use asciisave() to save information if you are planning to restore the information on a different type of computer or different version of MacAnova.

In a version with windows (Macintosh, Windows, Motif), Restore Workspace on the File menu equivalent to restore("").

If variables HOME and/or VERSION exist and are CHARACTER scalars, they

are restored to their original values after restore(). See files, launching.

Files created by earlier versions of MacAnova for the same computer can normally be restored correctly except that binary files created by MacAnova2.4x on a Macintosh cannot be restored.

See also asciisave(), save(), setoptions(), files.

2.235 resvsindex

Keywords: plotting, glm, residuals, anova, regression

Usage: resvsindex([varNo [,Chars]] [,graphics keyword phrases]), varNo a positive integer <= ncols(RESIDUALS), Chars a CHARACTER or REAL vector or scalar

resvsindex() makes a plot against case number of the studentized residuals from the most recent GLM (generalized linear or linear model) command such as regress(), anova(), or poisson() against case number. If the most recent command was manova(), only column 1 of residuals is plotted.

If option 'dumbplot' has been set False (see setoptions()), the plot will be a low resolution plot unless 'dumb:F' is an argument.

resvsindex(i) makes a plot vs case number of the studentized residuals associated with the i-th variable when the previous linear model command was manova().

resvsindex(i,Chars) is like resvsindex(i) except the plotting symbols are specified by Chars. Char must be a CHARACTER or REAL vector or scalar and is interpreted like the third argument to chplot(), except that a value of 0 is equivalent to run(nrows(RESIDUALS)), resulting in points being labeled with case number. If Chars is used, i must be specified, even if the response is univariate (in which case i must be 1).

You can use all the plot() keywords, including 'title', 'xlab', 'ylab', and 'file'. See graphs.

The quantities plotted are the internally studentized residuals from the fit and standard errors computed using all the data.

In the case of a non-linear GLM such as poisson() or logistic(), the residuals plotted are computed from residuals in the transformed scale based on the last step of the iteratively reweighted least squares algorithm used in the fit.

resvsindex is implemented as macro.

See also resvsrankits, resvsyhat, resid.

2.236 resvsrankits

Keywords: plotting, glm, residuals, anova, regression
Usage: resvsrankits([varNo [,Chars]] [,graphics keyword phrases]), varNo a positive integer <= ncols(RESIDUALS), Chars a CHARACTER or REAL vector or scalar

resvsrankits() makes a "rankit" plot of the studentized residuals from the most recent GLM (generalized linear or linear model) command such as regress(), anova(), or poisson(). If the most recent command was manova(), only column 1 of residuals is plotted.

If option 'dumbplot' has been set False (see setoptions()), the plot will be a low resolution plot unless 'dumb:F' is an argument.

resvsrankits(i) makes a rankit plot of the studentized residuals associated with the i-th variable when the previous linear model command was manova().

resvsrankits(i,Chars) is like resvsrankits(i) except the plotting symbols are specified by Chars. Char must be a CHARACTER or REAL vector or scalar and is interpreted like the third argument to chplot(), except that a value of 0 is equivalent to run(nrows(RESIDUALS)), resulting in points being labeled with case number. If Chars is used, i must be specified, even if the response is univariate (in which case i must be 1).

You can use all the plot() keywords, including 'title', 'xlab', 'ylab', and 'file'. See graphs.

The quantities plotted are the internally studentized residuals based on fitted values and standard errors computed using all the data.

In the case of a non-linear GLM such as poisson() or logistic(), the residuals plotted are computed from residuals in the transformed scale based on the last step of the iteratively reweighted least squares algorithm used in the fit.

resvsrankits is implemented as macro.

See also resvsindex, resvsyhat, resid.

2.237 resvsyhat

Keywords: plotting, glm, residuals, anova, regression
Usage: `resvsyhat([varNo [,Chars]] [,graphics keyword phrases]), varNo a positive integer <= ncols(RESIDUALS), Chars a CHARACTER or REAL vector or scalar`

resvsyhat() makes a plot against predicted values of the studentized residuals from the most recent GLM (generalized linear or linear model) command such as regress(), anova(), or poisson(). If the most recent command was manova(), only column 1 of residuals is plotted.

If option 'dumbplot' has been set False (see setoptions()), the plot will be a low resolution plot unless 'dumb:F' is an argument.

resvsyhat(i) makes a similar plot using the residuals and predicted values associated with the i-th variable when the previous linear model command was manova().

resvsyhat(i,Chars) is like resvsyhat(i) except the plotting symbols are specified by Chars. Char must be a CHARACTER or REAL vector or scalar and is interpreted like the third argument to chplot(), except that a value of 0 is equivalent to run(nrows(RESIDUALS)), resulting in points being labeled with case number. If Chars is used, i must be specified, even if the response is univariate (in which case i must be 1).

You can use all the plot() keywords, including 'title', 'xlab', 'ylab', and 'file'. See graphs.

The quantities plotted are the internally studentized residuals based on fitted values and standard errors computed using all the data.

In the case of a non-linear GLM such as poisson() or logistic(), the residuals plotted are computed from residuals in the transformed scale based on the last step of the iteratively reweighted least squares algorithm used in the fit.

resvsyhat is implemented as macro.

See also resvsindex, resvsrankits, resid, yhat

2.238 reverse

Keywords: time series
Usage: `reverse(x), x a REAL or LOGICAL vector or matrix.`

reverse(x) reverses the order of the rows of x, where x is a REAL or LOGICAL matrix. The result has the same size, shape and type as x.

```
Example:
   reverse(vector(1,3,2,4,7,6)) is vector(6,7,4,2,3,1).
   reverse0(matrix(vector(1,2,3,4,5,6),3) yields matrix(vector(3,2,1,
   6,5,4), 3)
```

2.239 rft

Keywords: time series, complex arithmetic
Usage: `rft(rx [,divbyT:T]), rx a REAL matrix`

rft(rx) where rx is a REAL vector or matrix, computes the Hermitian form of the complex discrete Fourier transform of each column of rx, considered as a real series. Any MISSING values in rx are treated as if they were 0.

rft(rx,divbyt:T) does the same except the result is divided by the number of rows of rx.

hft(hconj(hx),divbyt:T) is the inverse of rft() in the sense that rx and hft(hconj(rft(rx)),divbyt:T) are equal except for rounding error.

The largest prime factor of nrows(rx) must not exceed 29.

See topic 'complex' for discussion of complex matrices in MacAnova.

See also cft(), hft(), hconj().

2.240 rnorm

Keywords: random numbers
Usage: `rnorm(n), n a positive integer`

rnorm(N) generates a vector of N pseudo-random normals with mean 0 and variance 1. N must be a positive integer.

If the random number generator has not been initialized by setseeds(), setoptions() or previous use of rbin(), rnorm(), rpoi() or runi(), the generator's "seeds" will be initialized automatically using the current time and date, and their values will be printed out.

See also setseeds(), getseeds(), setoptions(), rbin(), rpoi(), runi(), cumnor() and invnor().

2.241 robust

Keywords: glm, anova, regression

Usage: robust([Model] [,print:F or silent:T, trunc:c, maxiter:m, epsilon:eps, fstats:T, pvals:T, marginal:T]), Model a CHARACTER scalar, c and eps positive REAL scalars, m a positive integer.

robust(Model) computes a robust linear fit of the model specified in the quoted string or CHARACTER variable Model. An approximate analysis of variance (ANOVA) table is printed, as well as a robust estimate of sigma. In the case of normal errors with possible outliers arising from contamination, the square of the estimated sigma is approximately unbiased for the variance of the uncontaminated errors.

See topic 'models' for information on specifying Model.

When Model is omitted (robust() or robust(,...), the most recent GLM model is assumed, or, if there have not been previous GLM commands, the model specified in variable STRMODEL, if any.

Side effect variables created by robust() are DF, SS, HII, RESIDUALS, and WTDRESIDUALS. WTDRESIDUALS is not really weighted residuals; instead it is set to the vector of modified residuals sigmahat*psi(residuals/sigmahat). See below for details.

Inference based on the ANOVA table should be used with caution. Especially when there are cases with high leverage (large value of HII), or when the ratio of model degrees of freedom to error degrees of freedom is too large, the approximation can be misleading. Interpret with similar caution the results of secoefs() and contrast() after robust().

Keyword phrase	Default	Meaning
trunc:c	.75	Positive REAL c is used as a "truncation point" in the fitting algorithm. See discussion below. The larger the value, the less "robustifying."
maxiter:m	50	Positive integer m is the maximum number of iterations that will be allowed in fitting
epsilon:eps	1e-6	Small positive REAL specifying relative error in objective function required to end iteration
print:F	T	Suppress all output except warning and error messages. Side effect variables are set.
silent:T	F	Suppress all output except error messages. Side effect variables are set.
fstats:T	F	Print approximate F-statistics and P values

pvals:T	F	Print approximate P values; default is T with fstats:T
marginal:T	T	Specifies that SS are computed marginally. When there are no empty cells, and sometimes when there are, the computed SS are equivalent to SAS Type III SS. See topic glm for details.

The default values for fstats and pvals can be changed by setoptions(fstats:T) and/or setoptions(pvals:T). See setoptions().

Keyword phrase 'coefs:F' is not legal for robust().

The algorithm used is based on Sec. 7.8 of "Robust Statistics" (Wiley 1981) and Sec. 14 of "Robust Statistical Procedures" (SIAM 1977), both by Peter J. Huber. Coefficients and scale sigma are estimated by minimizing a certain function of sigma and the residuals r = y - fit (Huber 1981, eq. 7.7.9 and 7.7.14).

Effectively the algorithm deemphasizes cases whose apparent residual r[i] satisfies abs(r[i]) > c * sigma, where c is a constant "truncation point". The default value of c is .75 but it can be set by keyword 'trunc'.

After running robust(), WTDRESIDUALS contains the modified residuals rmod = sigmahat*psi(rhat/sigmahat), where sigmahat is the estimated scale, rhat is the vector of estimated residuals, and psi(z) = z when abs(z) < c, psi(z) = c for z >=c, psi(z) = -c, z <= -c.

Following a suggestion of Huber, the ANOVA table is computed from pseudo-data obtained as fithat + K*rmod, where fithat = y - rhat is the vector of estimated predicted values and K is a constant (see p. 40 of Huber 1977). K = (1 + (k/n)(1-mu)/mu)/mu, where n is the sample size, k is the number of model degrees of freedom, and mu = m/n where m is the number of non-truncated rhat[i], that is, which satisfy abs(rhat[i]/sigmathat) < c. The robust estimate of sigma is sqrt(mse/beta)/K, where mse is the error mean square from the ANOVA table and beta is a constant computable by beta = cumchi(c^2,3) + 2*c^2*(1 - cumnor(c)) = E[psi(z^2)] for N(0,1) z.

If you want ANOVA results with a different order of terms, you do not need to redo robust(); you can use anova() with the pseudo-data in the preceding paragraph as dependent variable. You can determine m needed to compute K as the number of cases for which WTDRESIDUALS and RESIDUALS are equal except for rounding error.

2.242 rotate

Keywords: time series, combining variables
Usage: rotate(x, k), x a REAL vector or matrix, k an integer.

rotate(x, k) "rotates" by k rows each column of the REAL or LOGICAL vector or matrix x. When k > 0, rows pushed down off the end are shifted to the start and when k < 0, rows pushed up before the start are moved to the end. k must be a single integer and is interpreted modulo nrows(x). Thus rotate(x,k) and rotate(x,k %% nrows(x)) are equivalent.

More explicitly, for -m < k < m, where m = nrows(x), rotate() moves rows as follows:

```
 0 <= k <= m-1 (down column shift)   -(m-1) <= k <= 0 (up column shift)
Row j -> Row j+k   for j=1,...,m-k   Row j -> Row j+k+m for j=1,...,-k
Row j -> Row j+k-m for j=m-k+1,...,m Row j -> Row j+k   for j=-k+1,...,m
```

Examples:

```
  rotate(vector(1,3,2,6,5,4),2) is vector(5,4,1,3,2,6)
  rotate(matrix(vector(1,3,2,7,6,5,4,8),4),-2) is matrix(vector(2,7,1,3,
  4,8,6,5),4)
```

Caution: Do not confuse rotate() with rotation() which does rotation of factor loadings.

2.243 rotation

Keywords: multivariate analysis

Usage: rotation(loadings [, method:name, reorder:T, verbose:T]), where loadings is REAL matrix and name is CHARACTER scalar

rotation(Loadings) or rotation(Loadings, method:"varimax") computes the "rotation" of Loadings using the Varimax criterion. That is, an orthogonal matrix R is found so that Loadings %*% R maximizes a criterion. The value returned is Loadings %*% R. At present, Varimax is the only method implemented. Loadings must be a REAL matrix with nrows(Loadings) >= ncols(Loadings). The result is a REAL matrix with the same dimensions as Loadings.

rotation(Loadings, verbose:T) prints the value of the Varimax criterion before and after rotation.

rotation(Loadings, reorder:T [,verbose:T]) enables post-processing that mulitiplies each column of the result to make its sum positive and reorders columns in decreasing order of the column sums of squares.

Function rotation() is designed to be used with the matrix of loadings from a factor analysis, but may also be used with matrices of canonical variable coefficients, possibly after rescaling.

The rotation matrix can be computed as

```
  R <- solve(Loadings %c% Loadings, Loadings %c% rotation(Loadings))
```

The algorithm used is iterative, using a default convergence criterion of epsilon = .00001, and performing a maximum of 100 iterations. These

values can be modified by including keyword phrases 'epsilon:value' and/or 'maxiter:n', where value is a small positive number and n is a positive integer.

Caution: Do not confuse rotation() with rotate() which shifts the rows of its first argument up or down, wrapping around the end.

2.244 round

Keywords: transformations
Usage: round(x [, ndec]), x REAL, ndec an integer

round(x) rounds the elements of the REAL variable x to the nearest integer, producing a vector, matrix, or array with the same shape as x.

round(x,n) where n is an integer is equivalent to 10^(-n)*round(x*10^n). If n > 0, this rounds to n decimal places. If n < 0, this rounds to the nearest multiple of 10^abs(n). round(x,0) is equivalent to round(x).

If x is a structure, so is round(x) or round(x,n). If xi is the i-th component of x, the i-th component of round(x) or round(x,n) is round(xi) or round(xi,n).

Example: round(3141.593,2) is 3141.59 and round(3141.593,-2) is 3100, the nearest multiple of 100 = 10^2.

round(x, p) can also be used when x is a CHARACTER variable and p, if present, is a quoted string or CHARACTER scalar or REAL scalar. The result is a CHARACTER variable of the same shape as x describing the transformation. For example, both round(vector("X1","X2"),3) and round(vector("X1","X2"),"3") return vector("round(X1,3)","round(X2,3)"). This can be useful for creating labels for a transformed variable.

See also floor(), ceiling(), structures, labels.

2.245 rowplot

Keywords: plotting
Usage: rowplot(x [, graphics keyword phrases]), x a REAL matrix

rowplot(x) makes an "interaction" plot of the data in the matrix x. The plotting positions are the column numbers and the values in x. Points within each row are joined by lines. Any keywords useable in chplot may follow x. Rowplot is implemented as a pre-defined macro.

If option 'dumbplot' has been set False (see setoptions()), the plot will be a low resolution plot unless 'dumb:F' is an argument.

```
Example:
  Cmd> rowplot(run(20)^(.2*run(5)'),\
       title:"X^vector(.2, X^.4, X^.6, X^.8, X)'")

See also colplot.
```

2.246 rpoi

Keywords: random numbers
Usage: `rpoi(n,lambda), n positive integer, lambda scalar >= 0 or non-negative vector of length n.`

```
rpoi(N,lambda) generates a vector of N independent Poisson pseudo-random
variables with mean lambda.  N must be a positive integer.

lambda must be a REAL scalar >= 0 or a REAL vector of length N with
lambda[i] >= 0.  If lambda is a scalar, it is used for every element of
the result.  Otherwise, element of the result will be Poisson with mean
lambda[i].

When used together with invgamma(), rpoi() can be used to generate
pseudo-random negative binomial random variables.

  Cmd> y <- rpoi(N, m*invgamma(runi(N),alpha)) # m > 0, alpha > 0

will generate negative binomial random variables with shape or index
alpha and mean alpha*m, with probabilities
 p[y=x]={(1-m)^alpha}*alpha*(alpha+1)...*(alpha+x-1)*m^x/x!, x=0,1,... .

If the random number generator has not been initialized by setseeds(),
setoptions() or previous use of rbin(), rnorm(), rpoi() or runi(), the
generator's "seeds" will be initialized automatically using the current
time and date, and their values will be printed out.

The algorithm is based on CD and AW Kemp, Appl Statist 40 (1991)
143-158.

See also setseeds(), getseeds(), setoptions(), runi(), rnorm(),
rbin(), invgamma(), cumpoi().
```

2.247 rsolve

Keywords: linear algebra
Usage: `rsolve(A , B), A a square REAL matrix, B a REAL matrix, nrows(A) = ncols(B).`

```
rsolve(a, b) computes the solution x to the system of linear equations
```

```
x a = b, where a is a REAL square matrix and b is a REAL matrix with
ncols(b) = nrows(a).  rsolve(a, b) produces the same result as
solve(a',b')', and is mathematically but not computationally equivalent
to b %*% solve(a).

If a is singular, an informative message is printed and the operation
aborts.

MISSING values are not allowed in a or b.

Expression b %/% a is equivalent to rsolve(a, b).

See also solve(), swp().
```

2.248 run

Keywords: combining variables, variables
Usage: `run(first,last,incr) or run(first,last) or run(last)`

```
run(First,Last,Incr) will generate a sequence of numbers from First to
Last with step size given in Incr.  The i-th number is computed as
First+(i-1)*Incr (unless it is very close to 0 or Last;see below).  It
is an error if Incr is 0, unless First = Last.

run(First,Last) is equivalent to run(First,Last,1) if Last > First, and
to run(First,Last,-1) if Last < First.

run(n) is equivalent to run(1,n) if n is an integer.  This is the most
common usage.

If a is a vector of length 3, run(a) is equivalent to run(a[1],a[2],
a[3]).

If Incr is (Last-First)/n, where n is an integer, there will be n+1
values, with the (n+1)-th being exactly Last, even if First+n*Incr is
slightly less or greater than Last because of rounding error.
Similarly, if 0 is between First and Last and Incr is -First/n, the
(n+1)-th value will be exactly 0 even if First+n*Incr is not exactly 0
because of rounding error.  In both situations, the value is rounded to
the target value (Last or 0) if abs((First+n*Incr-target)) <
1e-15*abs(Last-First).
```

2.249 runi

Keywords: random numbers
Usage: `runi(n), n a positive integer`

```
runi(count) generates a vector of count pseudo-random uniforms on the
```

interval 0 to 1.

If the random number generator has not been initialized by setseeds(), setoptions() or previous use of rbin(), rnorm(), rpoi() or runi(), the generator's "seeds" will be initialized automatically using the current time and date, and their values will be printed out.

See also setseeds(), getseeds(), setoptions(), rnorm(), rbin() and rpoi().

2.250 samplesize

Keywords: probabilities, glm, anova
Usage: samplesize(noncen,ngrp,alpha,Pwr [, design:"rbd"] [,maxn:n]), noncen, alpha, Pwr positive, ngrp, n positive integers

samplesize(noncen,ngrp,alpha,Pwr) computes the group sample size (number of replicates) required in a balanced one-way analysis of variance (completely randomized design with equal group sizes) of ngrp groups at significance level alpha to achieve power Pwr. The noncentrality parameter noncen is the sum of the squared treatment effects divided by the error variance.

samplesize(musq/sigmasq,1,alpha,Pwr) computes the sample size computed required to achieve power Pwr in a single-sample two-tail t-test of H0: mu = 0 when mu = sqrt(musq) and variance is sigmasq.

samplesize(noncen,ngrp,alpha,Pwr,"design:"rbd") computes the number of blocks required to achieve power Pwr in a randomized complete block design with ngrp >= 2 treatments.

If the required sample size > 256, a warning message is given and the value 256 is returned. This default maximum sample size can be changed to m by including 'maxn:m' as a final argument, where m is a positive integer.

See also power() and power2().

2.251 save

Keywords: files, general, variables, null variables
Usage: save(FileName [,all:T, v335:T, nulls:F, options:F, ascii:T])
save() repeats previous save() or asciisave()

save(FileName) saves the MacAnova "workspace", that is, all the current variables and option values, in a file with name given in the quoted string or CHARACTER variable FileName. The file will be in a binary

format that is specific to the type of computer. See files.

save(FileName,ascii:T) is equivalent to asciisave(FileName), that is, the file written will be an ASCII text file instead of a binary file. This option can be used together with others described below.

In a version with windows (Macintosh, Windows, Motif), FileName can be "", in which case you will be prompted for the file name.

save(), with no file name, saves the workspace in the same file as specified in a previous asciisave() or save(). If the previous save was ASCII, so will be the current save. Moreover, if the previous save() specified a older version (v24:T or v31:T), the same option will be used, unless explicitly changed. If there was no previous save() or asciisave(), omitting the file name is an error.

save(FileName, var1, var2, ...) saves only variables or macros var1, var2, ... on the file. If any of the variables saved is specified in keyword form, the keyword is used for the name. The items saved can be restored without deleting everything by restore(FileName,delete:F).

save(FileName,options:F) suppresses the otherwise automatic saving of the current options values, including random number seeds. If options:F is not used, when restore() is used to recover the information on the file, the options are automatically reset to the saved values.

save(FileName,all:T), saves, in addition to variables, macros and option values, a variety of internal information computed by the last GLM command, in a form such that it will be recovered by restore(). This option is probably useful only when you have just completed a GLM (generalized linear or linear model) computation (regress(), anova(), poisson(), etc.) that took a long time to compute. The special information saved is used by coefs(), secoefs() and modelinfo(). If the model was complex and had many cases, 'all:T' can greatly increase the size of the save file.

save(FileName, nulls:F [, var ...]) does not save NULL variables.

There have been minor differences in save file format among different versions of MacAnova. For instance, If any NULL variables or variables with MISSING values are saved, save files written by version 3.36 or later cannot be restored by earlier versions. However, you can ensure compatibility by the use of keywords.

save(FileName,v24:T [, var ...]) or save(FileName,old:T [, var ...]) saves in the format recognized by versions 2.4x and earlier of MacAnova. Keywords all and options are ignored and options are not saved.

save(FileName,v31:T [, var ...]) saves in the format recognized by versions 3.0 and 3.1x of MacAnova.

save(FileName,v335:T [, var ...]) saves in the format recognized by version 3.35 of MacAnova. NULL variables are not saved.

Note: Prior to version 4.01, names starting with '_' were not permitted. Variables with such names can be restored by earlier versions, but they cannot be used.

save() also saves the current time and date. These are reported by restore() when the workspace or variables are restored.

In addition, save() saves information about the computer MacAnova is running on and the compiler it was compiled with, plus information about the internal representation of linear models. This information is used by restore() to determine if it is safe to restore variables. If you are planning to restore the file on another computer or different version of MacAnova, use asciisave().

save() differs from asciisave() in that asciisave() saves the information in the form of a "text" file, more or less human readable, that can be transferred between different types of computers. Files created by asciisave() are often bigger than the corresponding file created by save().

Save() and asciisave() are useful when a session must be interrupted and you don't want to lose what you have done. On an unstable system or when using MacAnova remotely over a noisy phone line, a useful insurance measure is to save your current variables every few minutes. The following creates a macro that makes this easy:

```
    snapshot <- macro("save(\"snapshot.sav\",all:T)")
```

Then simply typing 'snapshot()' saves a copy of your workspace, options, and internal variables related to GLMs on file snapshot.sav. They can be restored by 'restore("snapshot.sav")'.

Examples:

```
  Cmd> save("chckpnt.bin") # save entire workspace, but not GLM stuff
  Cmd> save("saveFile",all:T) # save everything, including GLM stuff
  Cmd> asciisave("saveFile",x,y,residuals:RESIDUALS, v31:T)
  Cmd> save("saveFile",x,y,residuals:RESIDUALS, v31:T, ascii:T)
```

The last two are identical and save only x, y, and RESIDUALS on ASCII file saveFile readable by MacAnova 3.1. When restored, RESIDUALS will be recreated with name 'residuals'.

On a Macintosh, Save Workspace on the File menu (Command+K) is equivalent to 'save()', except that if there hasn't been a previous save() or asciisave(), you are prompted for a file name using the usual scrolling dialog box. When you Quit from MacAnova, you are normally asked if you want to save the workspace.

See also asciisave(), restore(), setoptions(), files

2.252 screen

Keywords: glm, regression

Usage: screen([Model] [, method:"cp" or "rsq" or "adjrsq", mbest:m, forced:fn, s2:mse, penalty:pen, keep:items]), m positive integer, fn vector of positive integers, mse and pen positive REAL scalars, items CHARACTER scalar or vector with elements "p", "cp", "rsq", "adjrsq", "model" or "all".

screen(Model) performs a screening of all possible regression models based on subsets of the X variables given in the CHARACTER argument Model. screen() prints out the best subsets, together with the values of Mallow's Cp, multiple R squared (coefficient of determination), and adjusted R squared. If not specified otherwise by keywords 'method' and 'mbest', screen() finds the 5 models with the smallest values of Mallow's Cp statistic.

See topic 'models' for information on specifying Model.

In addition to Model several optional keywords are recognized:

Keyword	Type of value	Default	Meaning
method	CHARACTER variable	"cp"	Criterion ("cp", "rsq", or "adjrsq") for subset selection
mbest	Positive integer	5	Number of subsets to be found
forced	REAL Vector of positive integers or independent variable names	none	List of independent variables to be forced into all subsets
s2	Positive REAL number	MSE	Replacement for full model MSE in computing Cp
penalty	Positive REAL number	2	Multiplier of p in computing Cp
keep	CHARACTER vector or scalar	none	Specifies items to return as value. Permissible items are "p", "cp", "rsq", "adjrsq", "model", and "all" (all preceding).
print	T or F		print:T forces printing when 'keep' is used.

Keyword 'silent', 'fstats', and 'pvals' are illegal for screen(), and 'print:F' is illegal unless keyword 'keep' is used.

Keyword 'coefs' is illegal with screen().

The best mbest subsets are printed or, when 'keep' is used, returned. When the method is "adjrsq" or "cp", the mbest models with the largest

value of adjusted R^2 or the smallest value of Mallow's Cp statistic are printed. When the method is "rsq", for each possible number m, m = 1, 2, ..., k, of variables, screen() selects the mbest models with largest value of R^2, where k is the number of variables in the model. Thus in this case, up to (k-1)*mbest + 1 models would be selected.

The vector forced contains the names (or positions) of any variables that should be forced into the model. By default, no variables are forced.

The CHARACTER variable method sets the selection criterion. The possible values are "adjrsq" or "a" for adjusted r^2, "rsq" or "r" for r^2, and (default) "cp" or "c" for Mallow's Cp.

Keyword s2 sets the error variance for used in computing Cp. If not specified, the mean square error from the complete model is used.

REAL variable penalty modifies the way Cp is computed. Larger values increase the "penalty" of including additional variables. The default value is 2.

Examples, all assuming Model is "y=x1+x2+x3+x4+x5+x6+x7"

Screen with defaults mbest = 5,method = "cp",and penalty = 2

```
Cmd> screen(Model)
```

Screen for 10 best models using adjusted R^2 with variable x3 forced into the model:

```
Cmd> screen(Model,mbest:10,forced:"x3",method:"adjrsq") # or forced:3
```

Screen using Cp over models with x6 forced in and penalty factor of 3:

```
Cmd> screen(Model,forced:"x6",penalty:3)
```

Screen using defaults, saving p, cp, and the models, and printing results:

```
Cmd> result <- screen(Model,keep:vector("p","cp","model"),print:T)
Cmd> regress(result$model[1]) # compute regression with best model
```

See also regress() and anova().

2.253 secoefs

Keywords: glm, anova, regression, confidence intervals

Usage: secoefs([Term] [, errorTerm:ErrorTerm, byterm:F, se:F or coefs:F]), Term a CHARACTER scalar, a positive integer, or a factor or variate in the current GLM model, ErrorTerm a CHARACTER scalar or positive integer. byterm:F only when Term, se:F and coefs:F omitted

secoefs(Term) returns the model effects or regression coefficients and

their standard errors for the term specified in the CHARACTER variable Term. These are determined from information computed by the most recent GLM (generalized linear or linear model) command such as regress(), anova(), or poisson(). Interactions produce matrices or arrays with the leftmost subscript corresponding to the leftmost factor in Term. The result is a structure with components 'coefs' and 'se'.

Term is usually a quoted string or CHARACTER variable such as "a.b" which exactly matches a term in the most recent model, that is, "a.b" is not the same as "b.a". An interaction term produces a matrix or array with the leftmost subscript corresponding to the leftmost factor in Term.

If any variables in Term originally specified in the form {expr}, where expr is a MacAnova expression, you must include the enclosing '{' and '}'.

For a term which consists of a single factor or variate, Term can be its unquoted name.

Alternatively, Term can be a integer between 1 and the number of terms, excluding the final error term. Thus, for example, unless the model contained "-1", secoefs(1) gets the estimated intercept or grand mean and its standard error.

secoefs() (no Term specified) computes coefficients and standard errors for all terms in the model. The result is a structure with one component for each term in the model, with the components themselves being structures with components 'coefs' and 'se'.

secoefs(byterm:F) is the same as secoefs() except that the resulting structure has two components, 'coefs' and 'se', each of which is a structure with one component per term (unless there is only one term in the model).

secoefs(Term,coefs:F) and secoefs(coefs:F) (or secoefs(,coefs:F)) suppress the computation of the coefficients, returning a structure or matrix containing only standard errors. secoefs(Term,se:F) and secoefs(se:F) are equivalent to coefs(Term) and coefs(), respectively. This feature can be used to compute a structure containing t-statistics for every coeficient by coefs()/secoefs(coefs:F) or secoefs(se:F)/secoefs(coefs:F). Alternatively, you could compute such a structure by

```
@tmp <- secoefs(byterm:F); tstats <- @tmp$coefs/@tmp$ses
```

secoefs(Term,Varno) or secoefs(,Varno) computes coefficients and standard errors only for variable number Varno in the case of a multivariate dependent variable. If present, Varno must be the second argument and any keywords must follow it.

For all forms, an optional keyword phrase argument errorterm:ErrTerm or errorterm:ErrTermNo, where ErrTerm is a CHARACTER variable or quoted string specifying a term in the model and ErrTermNo is a positive

integer, specifies that the MS from the indicated term is to be used in computing standard errors.

If tcrit is a critical value, say, invstu(1-alpha/2,errorDF), you can compute the lower 1-alpha confidence limits for the coefficients

```
Cmd> @tmp <- secoefs(byterm:F);@tmp$coefs - tcrit*@tmp$se
```

and similarly for upper limits (replace - by +).

Secoefs() does not work after fastanova(), ipf(), or screen(), or if 'coefs:F' was an argument to the most recent GLM command..

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

- secoefs(a), secoefs("a"), or secoefs(1) will compute the main effect coefficients and their standard errors for factor a
- secoefs(a,coefs:F), secoefs("a",coefs:F), or secoefs(2,coefs:F) will compute the standard errors of main effect coefficients
- secoefs("a.b") or secoefs(4) will produce matrices of the a by b interaction coefficients and their standard errors.
- secoefs() will produce all coefficients and their standard errors in a structure with components CONSTANT, a, b, and a.b.
- secoefs(byterm:F) will produce all coefficients and their standard errors in a structure with components coefs and se

This will produce the a by b interaction effects and their standard errors.

See also coefs().

2.254 select

Keywords: combining variables, variables

Usage: select(k, x), k vector of positive integers or LOGICAL vector, x a matrix.

select(k, x) computes vector(x[1,k[1]],x[2,k[2]],...,x[n,k[n]]), where n = nrows(k). Thus select() selects the k[i]-th element of the i-th row of x. The length of the result is nrows(k).

k must be a REAL vector of positive integers or a LOGICAL vector and x must be a matrix with nrows(x) >= nrows(k) and ncols(x) >= max(k),

If k is a LOGICAL vector, False is translated to 1, and True is translated to 2. Thus if x is a matrix with two columns, select() can be used to select column 1 or column 2 of x depending on whether k[i] is False or True.

If k[i] is MISSING, select(k,x)[i] is MISSING when x is LOGICAL or REAL and is "" when x is a CHARACTER variable.

If x is REAL or LOGICAL and k has no MISSING values, select(k, x) is equivalent to vector(x[hconcat(run(nrows(k)), k)]).

See subscripts.

2.255 sethistory

Keywords: general
Usage: sethistory(lines), lines is a CHARACTER vector with length <= value of option 'history'

sethistory(Lines), where Lines is a CHARACTER vector, resets the internal list of previous commands to Lines with Lines[1] the earliest command available. Subsequent keyboard or menu retrieval of previous commands will retrieve elements of Lines.

It is an error if length(Lines) > nHist, where nHist is the value of option 'history'. See setoptions(), getoptions().

See gethistory() for an example using gethistory() and sethistory() to maintain a history of commands executed between MacAnova sessions.

sethistory() is not implemented in the limited memory DOS version or in any version that does not allow keyboard or menu retrieval of previous commands.

2.256 setoptions

Keywords: control, missing values, output, random numbers
Usage: setoptions(option1:value [,option2:value ...])
option1, option2, ... option names. Legal option names are 'angles', 'batchecho', 'dumbplot', 'errors', 'format', 'fstats', 'height', 'inline', 'labelabove', 'labelstyle', 'maxwhile', 'missing', 'nsig', 'prompt', 'pvals','restoredel', 'seeds', 'update', 'warnings', 'wformat', and 'width' plus 'font' and 'fontsize' on Macintosh, 'scrollback' on Macintosh, Windows and Motif, and 'history' on Macintosh, Windows, Motif and some Unix and DOS/Windows versions. setoptions(str), where str is of the form structure(option1:value, ...)

setoptions(keyword:value [,keyword:value...]) changes the values of various items specified by the keywords.

Legal option names are 'angles', 'batchecho', 'dumbplot', 'errors', 'format', 'fstats', 'height', 'inline', 'labelabove', labelstyle', 'maxwhile', 'missing', 'nsig', 'prompt', 'pvals', 'restoredel', 'seeds', 'update', 'warnings', 'wformat', and 'width'.

Option name 'lines' is a synonym for 'height' for compatibility with previous versions.

On versions allowing you to recall previous commands (Macintosh, Windows, Motif, Unix, extended memory DOS), option 'history' is also available.

On windowed versions (Macintosh, Windows, Motif), option 'scrollback' is also legal.

On computers such as a Macintosh, with changeable fonts, options 'font' and 'fontsize' are also legal.

See below for details on these options.

setoptions(Options), where Options is a structure with component names matching any or all of the legal option names, sets the options from the component values. Thus if Options was created by 'Options <- getoptions()', setoptions(Options) resets the options to what they were at the time getoptions() was invoked. See also getoptions().

setoptions(defaults:T) restores all the options to their default values. It is an error if there is more than one argument. If a prompt was set at start up, it is restored. When setoptions(defaults:T) occurs in a batch file, the current batch prompt is restored to what it was when the batch file was opened, either the file name or the value of keyword 'prompt' on the batch() command. See batch().

On a Macintosh, you can use the Options menu to change most of the options. This works by generating and executing an appropriate setoptions() command.

In the following, the permissible values for an option are given in parentheses after the option name. A quoted string can be replaced by a CHARACTER scalar and T or F can be replaced by a LOGICAL scalar.

Options that may be set

angles ("radians", "degrees" or "cycles")
  setoptions(angles:units) specifies the angular units assumed for sin(), cos(), tan(), asin(), acos(), atan(), cpolar(), hpolar(), crect(), and hrect(). The default value for units is "radians". When units is "degrees", 360 is equivalent to 2*pi radians; when it is "cycles", 1 is equivalent to 2*pi radians).

batchecho (T or F)
  setoptions(batchecho:F) sets the default behavior of batch() to not echo input commands, while setoptions(batchecho:T) sets it to echo input commands (the usual behavior). If batchecho has been set to False, you must use batch(FileName,echo:T) to have commands in a batch file echoed. See batch().

dumbplot (T or F)
  setoptions(dumbplot:T) sets the default behavior of all plotting

commands (plot(), chplot(), lineplot(), addpoints(), ...) to make "dumb" plots, that is plots using only characters such as could be produced on a printer with no graphics capability. When option dumbplot has been set to True, you can still get high resolutions graphs using keyword phrase 'dumb:F' on the plotting command. If you use spool() to save your output to disk, with dumbplot:T copies of all your plots will be spooled.

errors (integer >= 0)
setoptions(errors:n) sets the maximum number of errors tolerated to n. n = 0 means errors will not be counted. When executed from a batch file, the specified value of errors is used only until entry from the keyboard is next needed. Exceeding the limit while executing commands in a batch file, terminates reading from all current batch files. At startup, n is set to 0 (ignore errors). If it is not reset, use of batch() temporarily sets the error limit to 1; it reverts to 0 when commands from the batch file are completed.

font (quoted string)
setoptions(font:Font) sets the font for the command window, and any additional command windows that you may subsequently open. Font should be the name of an available font such as "Courier" or "Monaco". Font can also include a font size, for example, "Courier 12".

setoptions(font:"default") restores the default font and size.

If you change the font, you will almost certainly want to choose a non-proportional font such as "Courier" or "Monaco", that is a font for which all characters, including spaces, have the same width. If not, columns will not line up and, in general, output will be hard to read.

This option is available only on computers such as a Macintosh, on which you can change fonts. If you regularly prefer a font other than the default font, you might want to set this option in your startup file, .macanova.ini or MacAnova.ini. See customize.

fontsize (integer > 0)
setoptions(fontsize:N) changes to N the font size for the command window and any additional command windows that might subsequently be opened. This option is available only on computers such as a Macintosh, on which you can change fonts.

format (quoted string)
setoptions(format:Format) sets the default format for printing. For example, if Format is "12.5g", most output will be in floating point form with 5 significant digits and a maximum width of 12 characters.

Format must be of the form "w.dg" (floating point with d significant digits and width w characters, including sign and exponent), or "w.df" (fixed point format with d digits after the decimal and minimum width w). If w is omitted (for example, ".5g") it is taken to be d+7. Examples are "10.5f" (fixed point with 5 decimal places and total

width of 10) and ".15g" (floating point with 15 significant digits and width 22).

The format type specifier ('f' or 'g') can also be at the start of Format. Thus format:"f10.6" is equivalent to format:"10.6f".

If w > 27, the width is taken to be 27 and if d > 20, the number of decimals or significant digits is taken to be 20. See print() for more discussion of format specification.

fstats (T or F)
setoptions(fstats:T) sets the default behavior of anova(), robust() and fastanova() so that F-statistics are printed unless keyword phrase 'fstats:F' is an argument.

setoptions(fstats:F) suppresses the default printing of F-statistics.

This option has an effect manova() only when 'byvar:T' is an argument.

height (integer >= 0)
setoptions(height:nLines) sets the number of lines that are assumed to fit on the screen to nLines. Under Unix and DOS, whenever there are nLines-1 lines of output on the screen, MacAnova will print "Hit RETURN to continue or q RETURN to go to next command line:" and then pause. If command line editing is available, the message is "Press 'q' to quit, 'j' or 'n' to see next line, any other key to continue". The value of option 'lines' also affects the default size of stemleaf displays and "dumb" plots. setoptions(height:0) suppresses pausing output and sets the default size of stemleaf displays and "dumb" plots to 24. On a Macintosh, the value of option 'lines' may also be reset when the output window is resized and the only effect of the value is on the number of lines in a "dumb" plot. On Unix and DOS, it can be predefined using the -l command line option. See stemleaf(), graphs and launching. This option can be temporarily set aside by keyword 'height' on graphics functions with 'dumb:T' and on print(), write(), matprint(), matwrite() and error().

history (integer >= 0)
setoptions(history:20) limits the number of command lines that are saved and may be retrieved to 20. The default value for history is 100 (50 on Macintosh).

setoptions(history:0) causes saving of lines to cease.

This option is meaningful only in versions allowing you to recall previous commands. This includes the Macintosh, Windows and Motif versions, most Unix versions, and the extended memory DOS version (DJGPP). See topics macintosh, wx, unix, and dos-windows.

inline (T or F)
setoptions(inline:F) changes the default method of macro expansion from expansion in-line (and only once in a loop) to expansion out-of-line (will be expanded every time through the loop). See topics

```
  macro(), macroread(), macros, macro files.  The default can be reset
  to in-line by setoptions(inline:T).  This does not affect the
  expansion of macros created by macro(macroText, inline:F) or those
  read by macroread() or read() as out-of-line macros.

labelabove (T or F)
  setoptions(labelabove:T) changes the default coordinate labelling use
  when printing variables without labels (see topic 'labels').  Instead
  of all the subscripts being printed in the left margin, the values for
  the last subscript (only subscript in the case of vectors) are printed
  above the data being printed.  This has no effect on the printed
  labels of CHARACTER varables which are always at the left.  The
  default value is F.  The effect is the same as if the variable had
  labels which were all of the form rep("@",dimsize).

labelstyle ("(", "[", "{", "<", "/", "\\")
  setoptions(labelstyle:"["), say, changes the style used in printing
  coordinate labels for unlabelled variables (see topic 'labels'), so
  that '[' and ']' are used instead of the default '(' and ')'.

lines (integer >= 0)
  setoptions(lines:nLines) does the same as setoptions(height:nLines).
  Retained for compatibility with earlier versions.

maxwhile (integer >= 10)
  setoptions(maxwhile:n) changes the default upper limit on the number
  of repetitions of a while(){...} loop to n.  The default is 1000.

missing (quoted string of no more than 20 characters)
  setoptions(missing:"NA") changes the default string used to print
  MISSING values to "NA" instead of the usual "MISSING".  Any string up
  to 20 characters long can be used instead of "NA".  You can use
  keyword "missing" on print() and write() to override the default
  string.  See print(), write().  This option does not affect the
  internal representation of MISSING or what gets written by matprint()
  and matwrite().

nsig (0 < integer <= 20)
  setoptions(nsig:d) is equivalent to 'setoptions(format:"w.dg")' where
  d is a positive integer and w = d+7.  It specifies that all numbers
  should be printed with in floating point format d significant digits
  in a field of with d+7 characters.

prompt (quoted string of no more than 20 characters)
  setoptions(prompt:newPrompt) changes the prompt from "Cmd> " to
  newPrompt.  Example: setoptions(prompt:"Next? ").

pvals (T or F)
  setoptions(pvals:T) sets the default behavior of all the GLM commands
  except for manova(), robust(), and screen() so that P values are
  printed unless keyword phrase 'pvals:F' is an argument.

  setoptions(pvals:F) suppresses the default printing of P values.
```

```
restoredel (T or F)
  setoptions(restoredel:F) sets the default value for keyword 'delete'
  on restore() to False, meaning existing variables will not be deleted
  unless overwritten.  The default can be overridden by delete:T on
  restore().

  setoptions(restoredel:T) sets the default value of 'delete' to T (the
  normal default value), meaning that existing variables will be deleted
  by restore() unless delete:F is an argument.

scrollback (T or F)
  setoptions(scrollback:T) or setoptions(scrollback:F) set the default
  value of keyword 'scrollback' for any other command as help() that
  recognizes it.  Thus, after setoptions(scrollback:T), output from
  help() will automatically scroll back to the beginning (unless
  scrollback:F is an argument to help().  This does not take effect
  until after the next prompt.

seeds (vector of 2 integers > 0 or both 0)
  setoptions(seeds:vector(n1,n2)) is equivalent to setseeds(n1,n2) where
  n1 and n2 are positive integers.  After setoptions(seeds:vector(0,0)),
  the first use of runi(), rnorm(), rpoi() or rbin() causes the seeds to
  be set to pseudo-random values determined from the time of day.

update (T or F)
  setoptions(update:T) enables and setoptions(update:F) disables
  updating of the text window or screen after a high resolution plot.
  This has no effect on the Macintosh.  The default value of update is
  True in DOS versions and False in Unix versions.

warnings (T or F)
  setoptions(warnings:F) instructs MacAnova to suppress the printing of
  all warning messages (starting with "WARNING:").  This can be useful
  when doing arithmetic with and transformations of vectors or matrices
  with missing data.  However, since many warning messages are very
  important, it should be used with caution.  You should use
  setoptions(warnings:T) to restore the usual behavior as soon as
  possible.

wformat (quoted string)
  setoptions(wformat:Format) sets the default format for the various
  write commands (write(), fwrite(), and matwrite()).

width (integer >= 30)
  setoptions(width:n) sets the assumed number of characters on a line to
  be n, an integer no smaller than 30.  This number, together with the
  current formatting variables, determines how many items are printed
  per line and the width of "dumb" plots.  On a Macintosh, this may also
  be reset when the output window is resized.  On Unix and DOS a
  specified value may be pre-defined by the -w command line option.
  Some output does not respect this limit.  This option can be
  temporarily set aside by keyword 'height' on graphics functions with
```

'dumb:T' and on print(), write(), matprint(), matwrite() and error().
See graphs, launching.

2.257 setseeds

Keywords: random numbers
Usage: setseeds(seed1,seed2) or setseeds(vector(seed1,seed2)), seed1 and seed2 non-negative integers <= 2147483399

setseeds(seed1,seed2) initializes the random number generator used by runi(), rnorm(), rbin() and rpoi(). The seeds should be non-negative integers <= 2147483399.

setseeds(seeds), where seeds is a vector of the form vector(seed1, seed2), is equivalent to setseeds(seed1, seed2). In particular, any time after setting seeds by seeds <- getseeds(), setseeds(seeds) restarts the generator to the state it was when you used getseeds().

If either seed is zero, the seeds are intialized with values generated from the time of day; the seeds generated will normally be printed. You can suppress the printing by setseeds(0,0,quiet:T).

See also getseeds(), runi(), rnorm(), rpoi(), rbin() and setoptions().

2.258 shell

Keywords: control, general
Usage: shell(command), shell(command,keep:T) or shell(command,interact:T), command a quoted string or CHARACTER scalar. !command immediately after the prompt

shell(command) submits the CHARACTER vector or string command to the operating system for execution. It is implemented in the Unix, Motif, and DOS versions but not on the Macintosh. It does not work in Windows 3.1 or Windows 95 but may in Windows NT. Except in the limited memory DOS version (BCPP), the program run by command must not expect any input from the keyboard.

Thus, for example, in the Unix or Motif versions,
```
  Cmd> shell("ls -l *.dat")
```
causes the output from the Unix command 'ls -l *.dat' to be printed, giving a full listing of all files in the current directory with names ending with '.dat'. Under DOS, the same effect is obtained by
```
  Cmd> shell("dir *.dat").
```

shell(command, interact:T) does the same as shell(command) except that you can interact with the program that is started up. This option is

required if, for example, you are using shell() to run an editor to modify a file. In the limited memory DOS version (BCPP), you can always interact with the command. When in doubt as to whether a program expects keyboard input, use interact:T.

shell(command, keep:T) runs the command in non-interactive mode and returns its output as a CHARACTER vector, with each line of output an element. The command must not expect any input from the keyboard. This is not implemented in the limited memory DOS version (BCPP).

Example on Unix:

```
  Cmd> datafiles <-\
        shell(paste("cd ",DATAPATHS[1],"; ls *.dat"), keep:T)
```

returns a vector of file names of the form *.dat in the directory whose name is in DATAPATHS[1].

Operating System Escapes

Somewhat simpler, but less powerful because it cannot be included in a macro, is the use of the 'escape' character '!'. In the Unix, Motif and DOS versions, any line of input starting with '!' immediately after the prompt will be passed on to the operating system for execution (without the '!'). Specifically,

```
  Cmd> !command to be run ...
```

is equivalent to

```
  Cmd> shell("command to be run ...", interact:T).
```

For example,

```
  Cmd> !ls -l *.dat
```

and

```
  Cmd> shell("ls -l *.dat", interact:T)
```

are equivalent.

In a command line starting with '!', double quotes ('"') and curly brackets ('{' and '}') have no special significance. The only "special" character is a backslash ('\') and then only when it occurs at the end of line to indicate that the command is continued on the next line. Unlike the case when an ordinary line is continued with backslash, a trailing backslash is deleted and is not seen by the operating system. Unbalanced quotes or brackets are ignored.

Under DOS, but not Unix, you can change default directories by, for example,

```
  Cmd> !cd b:\data    # or shell("cd b:\\data")
```

Caution: If you want to execute in MacAnova a command that starts with '!' (for example !(x < y)), precede it with a space. Thus

```
  Cmd> !(x < y)
```

attempts to execute "(x < y)" as a shell command, probably causing an error, while

```
  Cmd>  !(x < y) #note the extra space
```

is computed in MacAnova. Conversely,

```
  Cmd>  !ls -l *.dat
```

is an error, because a space has been typed before '!'.

In the DOS extended memory version (DJGPP), shell(command) or shell(command, keep:T) sometimes hangs. shell(command,interact:T) or !command is more reliable.

In the Windows version, shell(command,interact:T) and !command ignores command and starts up DOS similar to selecting MS-DOS Prompt in the Program Manager window.

In the Motif version, shell(command,interact:T) and !command can be confusing: Output does not appear in the MacAnova command window, but in the "parent" window from which MacAnova was started up, and any input must be typed in parent window.

On a Macintosh, starting a line with '!' is an error.

2.259 showplot

Keywords: plotting
Usage: showplot([Graph] [,graphics keyword phrases])

showplot(Graph) redraws the graph whose information is encapulated in Graph, which must be a variable of type GRAPH. If Graph is omitted, GRAPH variable LASTPLOT is plotted instead.

If option 'dumbplot' has been set False (see setoptions()), the plot will be a low resolution plot unless 'dumb:F' is an argument.

You may used keywords xlab, ylab, title, xmin, ymin, xmax, ymax, xaxis, yaxis and dumb on showplot(). This allows you to change the original labeling and limits simply. See topic graphs for details on these keywords. It is an error if xmax <= xmin or ymax <= ymin.

As all plotting commands, showplot() updates LASTPLOT to reflect the graph being plotted. To suppress the creation or updating of LASTPLOT, use keyword phrase 'keep:F' as an argument. Occasionally when memory is limited, it may be necessary to use keep:F to view the graph.

See also plot(), chplot(), lineplot().

2.260 solve

Keywords: linear algebra
Usage: solve(A), square REAL matrix A solve(A, B), square REAL matrix A, REAL matrix B, with nrows(B) = nrows(A).

solve(a) computes the inverse of the square matrix a.

solve(a,b) computes the solution x to the linear equation a x = b, where

a is square and b is a REAL vector or matrix with the same number of rows as a. solve(a,b) is mathematically, but not computationally, the same as solve(a) %*% b.

For either usage, if a is singular, an informative message is printed and the operation aborts.

MISSING values are not allowed in a or b.

Expression a %\% b is equivalent to solve(a, b).

See also rsolve(), swp(), qr(), matrices.

2.261 sort

Keywords: ordering
Usage: sort(x [,down:T]), x REAL or CHARACTER or a structure with all REAL or all CHARACTER components.

sort(x) sorts the data in a REAL or CHARACTER vector x into ascending order.

sort(x, down:T) or simply sort(x,T) sorts the data in descending order.

If x is a matrix, sort(x) or sort(x,T) is a matrix each of whose columns contains the ordered elements of the corresponding column of x.

If x is an array, sort(x) or sort(x,T) is an array of the same size and shape with all the elements with fixed values of subscripts 2, 3, ... defining a "column" which is sorted. An array with dimension > 2 is always treated as an array and not as a matrix, even if there are at most two dimensions greater than 1.

With REAL data, any MISSING values in a column are sorted to the end of the column, regardless of the direction of the sort.

With CHARACTER data, elements are sorted using the ASCII collating sequence in which most punctuation and all numerals sort ahead of upper case letters which sort ahead of lower case letters. A space sorts ahead of all printable characters. Here is the explicit ordering starting with space:

```
    !"#$%&'()*+,-./0123456789:;<=>?
   @ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
   `abcdefghijklmnopqrstuvwxyz{|}~
```

It is also acceptable for x to be a structure, whose non-structure components are all REAL or all CHARACTER. In that case, sort() returns a structure of the same form, each of whose non-structure components is the result of applying sort() to the corresponding component of x.

See also grade(), rank().

2.262 split

Keywords: combining variables, structures
Usage: split(x,A [,compnames:CharVar]), x REAL, A a factor or vector of integers or LOGICAL vector, CharVar a CHARACTER scalar or vector split(x,bycols:T or byrows:T [,compnames:CharVar]), x a REAL matrix

split(Data,Factor) creates a structure by splitting Data along its first subscript into separate components according to the values of Factor all of whose elements must be positive integers. If N = max(Factor), the result has N components, some of which may be empty. Thus if the values in Factor are group or treatment numbers, each component of the result consists of the data correponding to a particular group or treatment.

It is also acceptable for Factor to be a LOGICAL vector, in which case False and True correspond to factor levels 1 and 2, respectively. Thus, split(y, x <= 0) would create a structure with two components.

Data must be REAL or LOGICAL and the components of the result are the same type. Each component of the result will be a vector, matrix or array, depending on whether Data is a vector, matrix, or array. A warning is printed if any component of the result contains no elements. If Factor[i] is MISSING, all the corresponding data are omitted. It is an error for all the elements of Factor to be MISSING.

If Factor is a variable rather than an expression, say groups or @groups, the components will be named 'groups1', 'groups2', etc. Similarly if Factor is specified in a keyword phrase such as dose:rep(run(4),5), components will be named 'dose1', 'dose2', etc.

split(Data,bycols:T) or simply split(Data) splits the data along the last subscript, creating a structure with one component corresponding to each value of the last subscript. The most important case is when Data is a m by n matrix, in which case the result each of the n components of the result is a vector containing the data from a column of Data. Components will be named 'col1', 'col2', ... If Data is a vector, the result is a structure with a single component named 'col1'.

split(Data,byrows:T) splits the data along the first subscript, creating a structure with one component corresponding to each value of the first subscript. Thus if Data is a m by n matrix, the result is a structure with m components, each a row vector of size n (1 by n matrix). Components will be named 'row1', 'row2', ...

For all these usages, an additional argument of the form compnames:CharVec, is recognized, where CharVec is a CHARACTER vector. The elements of CharVec are used as names for the components of the result, truncated to 12 characters if necessary, overriding the naming

conventions just described. If length(CharVec) = 1, say "group", it is used as a "root" for forming names for the components of the form "group1", "group2", ... Otherwise length(CharVec) must match the number of components of the result. It is an error if any element of CharVec contains '$'.

An important usage of split is in boxplot(split(y,groups)), where y is a REAL vector and groups is a factor. This produces parallel boxplots of the of the data in y corresponding to each level of groups. Similarly, when y is a REAL matrix, boxplot(split(y,bycols:T)) or simply boxplot(split(y)), produces parallel box plots of the data in each column of y.

Examples:

```
  Cmd> split(run(4),variety:vector(1,2,1,2))
  component: variety1
  (1)             1             3
  component: variety2
  (1)             2             4
```

An equivalent command would be

```
  Cmd> split(run(4),vector(1,2,1,2),compnames:"variety")

  Cmd> boxplot(split(y),ylab:"Column Number")
```

where y is a matrix, produces parallel boxplots of the columns of y.

See also boxplot(), structures.

2.263 spool

Keywords: output, files

Usage: spool(FileName [,new:T]) spool() toggles spooling on and off.

spool(FileName) begins printing MacAnova input and output into the file with name given in the CHARACTER variable FileName. If the named file already exists, the spooled output will be added at the end of the file, thus allowing a cumulative record of several runs. Comments to annotate what you have done may be added to input lines by preceding them with '#'. See topic comments. In a version with windows (Macintosh, Windows, Motif), when FileName is "", you will be prompted for a name.

spool() suspends spooling if spooling is currently in effect and restarts it on the same file if spooling was previously in effect but is not now.

spool(FileName, new:T) restarts spooling at the beginning of the spool file, destroying any lines previously spooled.

spool(, new:T) restarts or resumes spooling at the beginning of the most recently used spool file.

On a Macintosh, selecting item Spool Output to File on the File menu is equivalent to typing 'spool("")'. If a spool file name has previously been provided, the item is labeled either Suspend Spooling or Resume Spooling and is equivalent to 'spool()'. In both cases it first erases everything after the prompt.

2.264 stemleaf

Keywords: descriptive statistics, plotting
Usage: stemleaf(x [, nstems, outliers:F, depth:F, stats:T, title:"Your title"]), x a REAL vector

stemleaf(var) prints a stem and leaf display of the data in REAL vector var. The number of stems depends on the number of non-missing values (must be at least 2) and the number of lines on the screen , and the maximum number of leaves printed is determined by the screen width (see setoptions()). An asterisk as the last leaf on a stem indicates there has been truncation and some leaves have not been printed.

stemleaf(nvar,nstems), nstems an integer > 1, selects the number of stems to be as large as possible <= nstems.

The data are scaled by a power of 10 and rounded toward zero so as to be integers. Then the final digits are the leaves and the stems are the leftmost digits or 0. If two stems go with a value, they are labeled, for example, as '2*' and '2.'), If 5 stems go on a value, they are labeled, for example, as '2*', '2t', '2f', '2s', and '2.'. A final line specifies the unit of the leaf digit.

By default, outliers more than 1.5 IQR beyond the lower or upper quartiles are listed separately and are not put on a stem, where IQR is the inter-quartile range.

By default, a "depth" column accumulating counts from each end is printed. The depth for the stem that contains the median is the number of leaves on that stem enclosed in parenthese.

With both forms additional keyword arguments are as follows:

outliers:F	This suppresses the special treatment of outliers; all values are put on stems
depth:F	Suppresses printing the "depth" column.
stats:T	Print the sample size, extremes and quartiles.
title:"Your title"	Prints the specified CHARACTER variable before display

See also boxplot().

2.265 strconcat

Keywords: structures, combining variables
Usage: `strconcat(var1 [,var2,...,vark] [, KeyPhrases])`, where var1, var2, ... are arbitrary variables KeyPhrases can be compnames:Charvec, labels:lab, and silent:T, where Charvec and lab are CHARACTER scalars or vectors.

strconcat(a,b,c,...) creates a structure from variables or structures a, b, c, etc. It differs from structure() in that, if any argument is a structure, its top level components become top level components of the result. Thus if a and d are non-structure variables strconcat(a, structure(b,c),d) produces a structure with 4 components, a, b, c, and d while structure(a,structure(b,c),d) produces a structure with 3 components, the second of which is itself a structure with 2 components.

The names of components derived from non-structure arguments are determined similarly to the names of components of structures created by structure().

See structure() for information about keywords 'labels', 'compnames' and 'silent'.

Example: Build structure in a loop

```
Cmd> nterms <- dim(SS)[1];f <- structure(SS[1]/DF[1])
Cmd> for(i,2,nterms-1){f <- strconcat(f,SS[i]/DF[i]);;}
Cmd> f <- strconcat(f/(SS[nterms]/DF[nterms]),\
      compnames:TERMNAMES[-nterms])
```

This produces a structure consisting of F-statistics with components having the names of the terms.

See also structures, structure(), compnames(), changestr(), keywords.

2.266 structure

Keywords: structures, combining variables
Usage: `structure(var1 [,var2,...,vark] [, KeyPhrases])`, where var1, var2, ... are arbitrary variables KeyPhrases can be compnames:Charvec, labels:lab, and silent:T, where Charvec and lab are CHARACTER scalars or vectors.

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

structure() differs from strconcat() in that the any argument that is a structure becomes a single component in the result, whereas strconcat() splits it into its top level components, each of which become top level components of the result.

If varj is a keyword phrase of the form name:value, the component is

named from the keyword. Thus, if a, b, and c are vectors, structure(a,b,c) and structure(a:a,b:b,c:c) are equivalent. If varj is a temporary variable, that is, a variable whose name starts with '@', then the '@' is stripped off to create the name of the component. If varj is an expression or function result, for example "3+4" or "x'", the component name will be NUMBER, LOGICAL, STRING, VECTOR, MATRIX, ARRAY, STRUCTURE, MACRO, or GRAPH depending on the type and shape of the component.

Keywords 'compnames', 'labels', and 'silent'

If a keyword phrase of the form compnames:CharVec is an additional argument, where CharVec is a CHARACTER vector, the names of the components are taken from CharVec, truncated to 12 characters if necessary, taking precedence over other names as described above. If CharVec is a scalar, say, "var", it is used as a root to create names of the form, say, "var1", "var2", ... Otherwise the number of elements in CharVec must match the number of components in the output. This usage is particularly useful in assigning names of 11 or 12 characters such as 'covariances" or 'factorloding'.

It is an error if any element of CharVec contains '$'.

You can add a vector of labels, one for each component by including keyword phrase labels:labs, where labs is a CHARACTER scalar or vector. These are printed instead of the component names on all output. See topic labels.

You can suppress all warning messages by silent:T.

Keywords compnames, labels, and silent must follow all arguments that will be included in the output structure.

Example: You can create a structure with components min and max by

```
  Cmd> extremes <- structure(min:min(x), max:max(x))
```

by

```
  Cmd> @min <- min(x);@max <- max(x); extremes <- structure(@min,@max)
```

or by

```
  Cmd> extremes <- structure(min(x),max(x),\
        compnames:vector("min","max"))
```

In all three cases you can add something like labels:vector("Minimum of x", "Maximum of x") to give a more complete labeling of the components.

2.267 structures

Keywords: structures, syntax, variables

Usage:

```
Create a structure:
   Str <- structure(Name1:x,Name2:y,...[, compnames:Name])
   Str <- strconcat(Name1:x,Name2:y,...[, compnames:Name])
Modify a structure:
   Str <- changestr(Str,comp,x), Str <- changestr(Str,-n)
Extract components:   Str\$name  or  Str[subscript]
Number of components: ncomps(Str)
Component names:      compnames(Str)
```

A structure contains one or more named components, each of which may be a variable or another structure. If structure Stats has 2 components with names 'mean' and 'var', then 'Stats$mean' and 'Stats$var' will access the mean and var components, respectively.

Various functions, for example secoefs() and describe(), return structures as their values. You can use structure() and strconcat() to create a structure with specific components, and changestr() to delete components from, replace components in, or add new components to an existing structure. You can use function split() to create a structure each of whose components are responses at different levels of a factor or consist of a column or row of a matrix. You can take a structure apart with vector() to produce a vector which consists of all the elements in the components of the structure.

The number of components of a structure Str may be computed by ncomps(Str) and their names may be retrieved as a CHARACTER vector by compnames(Str).

Many functions accept structures as arguments, including describe(), sum(), prod(), max(), and min(), and all the transformations. In addition, both binary (for example, +, *, %*%) and unary (-, +, !) operators accept structures as operands. Most of these produce a structure of the same shape as the argument(s) or operands(s).

Each component of the result is obtained by applying the function or operator to the corresponding component of the argument(s) or operand(s). Thus, for example, if a and b are compatible matrices, structure(a) %*% structure(b) is equivalent to structure(a%*%b).

In addition, if one operand of a binary operation is a structure and the other is not, a structure is computed, each of whose components is computed by combining the corresponding component of the argument structure with the other argument. Thus, for example, 3*structure(a,b,c) produces the same structure as structure(a:3*a,b:3*b, c:3*c), and structure(a,b) %C% c, where c is a matrix, produces the same structure as structure(a:a %C% c,b:b %C% c).

Note: You cannot use a structure as an operand to %/% or %\%. See matrices.

You can extract components of structure Str by number rather than name by Str[j] instead of Str$compName, where j is the number of the component wanted. Thus if Stats is as described above, 'Stats[1]' and 'Stats[2]' are equivalent to 'Stats$mean' and 'Stats$var', respectively. If there are more than one component with the same name or if the structure name is not a legal variable name, this is the only way to extract the component. For example Str <- structure(dim(x)[1], dim(x)[2], sqrt(x)) creates a structure with three components, named NUMBER, NUMBER, and MATRIX, but Str$NUMBER retrieves only the first component and Str[2] must be used to retrieve the second. Usually it is preferable to name components using keywords. For example, Str <- structure(m:dim(x)[1],n:dim(x)[2],data:sqrt(x)) creates a structure with components named 'm', 'n', and 'data'.

Str[NULL] is NULL.

More generally, in an expression of the form Str[j], j may be a vector of positive integers, a vector of distinct negative integers, or a LOGICAL vector of length ncomps(Str). Str[j] is a structure whose components are the components of Str selected by j in the same way elements of a vector vec would be extracted by vec[j]. If j selects only one component, Str[j] is that component. If j selects no commponents (all F's or a full set of negative values), STR[j] is NULL. See topic subscripts for how such a vector of subscripts is interpreted.

An alternative way to extract components from a structure is Str[[j]], using double square brackets. This is equivalent to Str[j] when Str is a structure. However, when x is not a structure, the value of x[[1]] is x, not x[1], and x[[j]] is an error if j != 1. This feature can be useful in a macro when an argument can either be a non-structure variable or the first component of a structure.

Examples: If Stats is as above, Stats[vector(2,1)] is equivalent to structure(var:Stats[2],mean:Stats[1]), and Stats[vector(F,T)] is equivalent to Stats[2] or Stats$var.

Assignment to a structure component by name or number (for example Stats$mean <- 3 or Stats[1] <- 3) is illegal. Use changestr() to change a structure.

2.268 subscripts

Keywords: syntax

Usage:
```
Ordinary subscripts:  x[13], y[2,vector(4,5)]
Negative subscripts:  w[-vector(1,3,5)], z[-run(5),-1]
Logical subscripts:  a[vector(T,T,F,F)], b[b < 0]
```

Elements of vectors may be selected in several ways using a list of integer or LOGICAL subscripts inside of square brackets. The list must be of one of three forms. In the examples, z is assumed to be

```
vector(1,7,4,9,6).
```

A list of one or more positive integers.
  Examples: z[4] is 9 and z[vector(1,1,2,2,5,4)] is vector(1,1,7,7,6,9).

A list of negative subscripts. These indicate omission of the absolute values of the subscripts. If all subscripts are omitted the result is NULL.

```
  Examples: z[vector(-1, -3, -5)] is vector(7,9).
            run(3)[-run(3)] is NULL
```

A LOGICAL vector with the same length as the source vector. The value is a vector with elements corresponding to values of True. If all the elements are False, the result is a NULL variable, with no elements.

```
  Examples: z[z>4] is z[vector(F,T,F,T,T)] = vector(7,9,6).
            z[z > 9] is NULL
```

z[vector(1,2,?)] would be an error because MISSING values are not allowed as subscripts.

z[vector(-3,-4,-3)] would be an error because there are duplicate negative subscripts.

z[vector(-1,2)] would be an error because there are both positive and negative integers in the same vector of subscripts.

Elements of matrices may be accessed in an analogous manner to vectors, except that both dimensions must be specified, separated by a comma. An empty row or column specification implies all rows or columns. Suppose

```
                                    1   4
q is the matrix  matrix(run(6),3) = 2   5 .
                                    3   6
```

Then q[3,1] is 3, and both q[vector(1,3),2] and q[-2,2] are the column vector 4,6 with dimensions 2 and 1. q[3,] is the row vector 3,6 and q[q[,1]>=2,] selects those rows of q for which column 1 is not less than 2, that is rows 2 and 3.

Elements of arrays may be accessed as for matrices, except that you must specify all subscripts. Again, an empty subscript specifies all legal values for that subscript.

With vectors, matrices or arrays, if any subscript is NULL or is non-selecting (all False or a complete set of negative subscripts), the result is a NULL variable.

It is not an error to use more subscripts than there are dimensions as long as no extra subscript specifies an element greater than the first for that dimension. Thus run(5)[,1], run(5)[,], and run(5)[,T] are equivalent and result in a 5 by 1 matrix, but run(5)[,2] is an error. This is a useful feature when writing macros that may operate on vectors and matrices, or deal with both ANOVA and MANOVA SS. For example, SS[3,,] is equivalent to SS[3] after anova() and is also meaningful

after manova().

If a is a matrix or array, and i is a vector, then a[i] is equivalent to vector(a)[i]. Thus matrix(vector(1,3,2,4,6,5),2)[vector(1,2,6)] yields vector(1,3,5).

When a is a factor and I is an appropriate vector of subscripts, a[I] is a factor with the same number of levels as a. Thus, for example, anova("{y[-1]} = {a[-1]}") carries out an analysis of variance omitting the first case. See factor(), anova() and models.

Matrix and Array Subscripts

You can also use a matrix as a single subscript in square brackets. If x is a vector, matrix or array with ndim dimensions, and Sub is a nrows by ndim matrix of positive integers, then x[Sub] is a vector of length nrows(Sub), whose i-th element x[Sub][i] is

x[Sub[i,1],Sub[i,2],...,Sub[i,ndim]]

For example, if x is n by n, x[hconcat(run(n),run(n))] is equivalent to diag(x) and x[hconcat(run(n),run(n,1))] extracts the cross diagonal of x. There can be no MISSING values in Sub.

More generally, if Sub is a array of positive integers whose last dimension has length ndim, x[Sub] is an array with ndims(x[Sub]) = ndims(Sub) - 1 with i,j,...,k-th element x[Sub][i,j,...,k] =

x[Sub[i,j,...,k,1],Sub[i,j,...,k,2],...,Sub[i,j,...,k,ndim]].

Assignment to Subscripts

You can assign to subscripts as well as extract from them. For example, if x is a matrix with at least 3 rows x[run(3),] <- 0 sets the first 3 rows of x to 0. x[-vector(run(5),run(16,20)),] <- ? sets rows 6 to 15 and beyond 20 to MISSING. A statement like 'y <- x[2,] <- 3' sets row 2 of x to all 3's and y to a row vector of 3's with dim(y)[2] == dim(x)[2].

You can assign NULL to subscripts provided at least one of the specified subscripts is NULL or is non-selecting (is all False or is a complete set of negative subscripts). The variable assigned to is not changed and the result of the operation is NULL. Thus, even if all the elements of x are positive, y[x < 0] <- w[x < 0] is legal and does not change y.

Similarly it is not an error to assign a scalar to subscripts when there is NULL or non-selecting subscript. The variable assigned to is not changed and the result is NULL. Thus x[vector(x) > 10] <- 10 is legal even when there are no elements of x greater than 10.

It is an error to assign a non-NULL non-scalar variable to subscripts when there is a NULL or non-selecting subscript.

If x is a matrix or array and i is a vector of integers or a LOGICAL vector and y is a scalar or a vector with the same length as i, x[i] <- y is legal and assigns the elements of y to the positions that would be specified by i if x were a vector. The dimensions of x are retained. Thus, for example,

```
  Cmd> x[vector(abs(x)) > 3] <- ?
```
replaces all elements of x that exceed 3 in absolute value by MISSING, without disturbing the dimensioning of x.

When y is a scalar or a vector, matrix or array with length(y) = length(x), x[] <- y is legal. The dimensions of x are preserved while those of a are ignored.

You can specify a single matrix of subscripts in assignment. If x has ndim dimensions, Sub is a k by ndim matrix of integers, and z is a vector of length k, x[Sub] <- z treats the i-th row of Sub as a set of subscripts, and sets the corresponding element of x to z[i]. Thus if x is 4 by 4, x[hconcat(run(4),run(4))] <- run(4,1) sets the diagonal of x to 4, 3, 2, 1. There an be no MISSING values in Sub. Note also that Sub must be a matrix and not an array when assigning values to x[Sub]. This contrasts to when x[Sub] is used to extract values from x, in which case Sub can be an array.

In assigning to a subscript, if the subscripts are such that a given element is referred to more than once, the value assigned to that element is the last of the corresponding values on the right hand side. Thus, for example, a[vector(2,2)] <- b[run(2)], is equivalent to a[2] <- b[2].

When you assign something to a subscripted variable, the assignment itself has a value, so that, for example,
```
  Cmd> y <- x[3,4] <- 17
```
sets both y and x[3,4] to 17. If multiple values are changed by such an assignment, for example,
```
  Cmd> x[1,] <- 17
```
the value has the same size and shape as the part of values in x replaces. Thus, for example,
```
  Cmd> y <- x[3,] <- 17
```
sets y to a 1 by ncols(x) vector, all of whose elements are 17. If Sub is a matrix, the value of x[Sub] <- y is a vector of length nrows(Sub).

See select() for a way to select a the k[i]-th element in the i-th row of a matrix, when k is a vector of positive integers.

A structure may have a single scalar or vector subscript which selects components of the structure in the same way a scalar or vector subscript selects elements of a vector. You cannot assign to a subscripted structure. See structures.

2.269 sum

Keywords: descriptive statistics

Usage: sum(x), x REAL or LOGICAL or a structure with REAL or LOGICAL components. sum(x1,x2,...), x1, x2, ... REAL or LOGICAL vectors, all the same type.

sum(x) computes the sum of the elements of a REAL or LOGICAL vector x. If x is LOGICAL, True is interpreted as 1.0 and False as 0.0 and sum(x) is the number of elements of x that are True.

If x is a m by n matrix, sum(x) computes a row vector (1 by n matrix) consisting of the sum of the elements in each column of x.

If x is an array with dimensions n1, n2, n3, ..., y <- sum(x) computes an array with dimensions 1, n2, n3, ... such that y[1,j,k,...] = sum(x[i,j,k,...], i=1,...,n1). This is consistent with what happens when x is a matrix. Note: MacAnova3.35 and earlier produced a result with dimensions n2, n3, ...

sum(NULL) is NULL.

sum(a,b,c,...) is equivalent to sum(vector(a,b,c,...)) if a, b, c, ... are all vectors. They must all have the same type, REAL or LOGICAL or be NULL. sum(NULL, NULL, ..., NULL) is NULL.

If all the elements of a vector x are MISSING, sum(x) is 0.0.

If x is a structure, sum(x) computes a structure, each of whose components is sum() applied to that component of x. All the components of x be of the same type, REAL or LOGICAL.

Examples:

If x is a n by m matrix, r <- x - sum(x)/n computes the matrix of the residuals of x[i,j] from the column means.

If x is a n by 4 by 5 array, r <- x - sum(x)/n computes an array with r[i,j,k] = the residual of x[i,j,k] from the mean of all x[i,j,k] with the same values for j and k. That is, it treats x analogously to a 4 by 5 array of vectors of length n. See arithmetic.

If x is a REAL matrix, sum(x)/sum(!ismissing(x)) is a row vector consisting of the averages of the non-missing values in each column.

If z is a vector of integers, sum(z == run(min(z),max(z))') computes a row vector giving the frequency distribution of the values in z.

See also prod(), tabs()

2.270 svd

Keywords: matrix algebra

Usage: svd(x [,left:T or F,right:T or F, all:T]), x a REAL matrix

svd(x) computes the vector of singular values in order of decreasing size, of the m by n REAL matrix x, where m >= n.

svd(x,left:T) computes the singular values and the m by n matrix of orthonormal left singular vectors in a structure with components 'values' and 'leftvectors'.

svd(x,right:T) computes the singular values and the orthogonal n by n matrix of right singular vectors in a structure with components 'values' and 'rightvectors'.

svd(x,all:T) and svd(x,left:T,right:T) both compute a structure with components 'values', 'leftvectors', and 'rightvectors'

svd(x,all:T,vals:F) computes a structure with components 'leftvectors' and 'rightvectors' only. Other combinations of all:T and other keywords are possible and do what you would expect.

If l and r are the matrices of left and right singular vectors and s is the vector of singular values then they satisfy (except for rounding error) l %*% dmat(s) %*% r' = x and l %c% l = r %c% r = I-sub-m.

See also eigen(), eigenvals(), releigen(), releigenvals().

2.271 swp

Keywords: matrix algebra, glm

Usage: swp(x, n1 [, n2, ...]), x a REAL matrix, n1, n2, ... positive integers or vectors of positive integers.

swp(x,Rows) uses a form of the Beaton SWP operator on the REAL matrix x to compute a REAL result of the same size. If x is m by n, Rows should be a vector whose elements are positive integers <= min(m,n).

A single SWP of x on row and column k produces a matrix y of the same size as x with

```
y[i,j] = x[i,j] - x[i,k]x[k,j]/x[k,k], for i and j not equal to k
y[i,k] = x[i,k]/x[k,k], for i not equal to k
y[k,j] = -x[k,j]/x[k,k], for j not equal to k
y[k,k] = 1/x[k,k]
```

The element x[k,k] is called a "pivot".

swp(x,Rows) does successive SWPs of x on Rows[1], Rows[2], ... rows and columns. If, on any SWP, the pivot is found to be too small in comparison with the magnitude of the original value in the same position of the diagonal, an informative message is printed and that SWP is skipped.

If x is n by n and non-singular, swp(x,run(n)) computes the inverse of x (it can fail for certain non-positive definite but invertible matrices).

swp(x,Rows1, Rows2, ...), where Rows1, Rows2, ..., are vectors of positive integers is equivalent to swp(x,vector(Rows1, Rows2, ...)).

For example, swp(cp,1,2,3,4) is equivalent to swp(cp,run(4)).

Function swp() can be very useful in regression and analysis of variance.

See also solve(), qr().

2.272 syntax

Keywords: syntax, control, general, character variables, logical variables, variables, missing values, null variables
Usage: Type help(syntax) for a summary of MacAnova syntax

Commands and Statements

A MacAnova command or statement is a sequence of characters typed in at the keyboard followed by ';' or '<cr>' (the RETURN or ENTER key). The first character should not be '!' unless it is a "shell escape" (see shell()).

Typical commands or statements are 'x <- vector(1.2,3.1,5.3,2.4)<cr>' (assign the vector (1.2,3.1,5.3,2.4) to variable x), 'print(x)<cr>' (print the value of variable x), 'regress("y=x1+x2+x3")<cr>' (compute a regression of variable y on variables x1, x2 and x3), or 'y <- 3*x^2<cr>' (assign to variable y the value of 3 times x squared).

Several commands or statements may be put on a single line terminated by <cr> if they are separated by ';'. An example would be

```
Cmd> regress("y=x");plot(x,RESIDUALS)<cr>
```

When you press '<cr>', but not before, all the commands or statements in the line are executed one after the other.

A number or algebraic expression involving numbers or variable names (for example '17.3', 'sqrt(4+cos(-1.32))', '3*log(640320)/sqrt(163)' and '3*y') which is not part of a larger statement or command is considered to be a statement. The only effect of this type of statement is to print out the value. This allows MacAnova to be used as a symbolic calculator.

Below for brevity both commands and statements are usually just called 'commands'.

Variables

Data are stored in permanent or temporary "variables" with names of up to 12 characters. Typical names might be 'x1', 'data', 'weight' or 'time_of_day'. The names of permanent variables start with a letter or '_') and the names of temporary variables start with '@' followed by a letter or '_'). The remainder of a name consists of letter, digits or '_'. Names are case sensitive (for example, 'residuals' is a different name from 'Residuals'. Some variables are "invisible". See topic variables for details.

Typically you will select names that are relevant to the problem such as 'weight', 'residuals', or 'depv'.

Typing the name of a variable that is not "invisible" prints out its value.

Command Line

The 'command line' consists of all the commands that are executed by a single <cr>. The command line follows the standard MacAnova prompt "Cmd> ", and may, in fact be continued on several actual lines -- see next paragraph.

When a command line becomes longer than one physical line, you can just keep typing as the characters wrap around to the next line (they do not wrap in the Windows and Motif versions; see topic wx). Alternatively, you may continue a line by typing '\<cr>'. Continue typing the line after the prompt "More> " (there is no "More> " prompt on in windowed versions -- Macintosh, Windows or Motif). If the break is at the end of a command, you will need to type ';' before '\<cr>'. In the limited memory DOS version you should probably always break lines with '\<cr>' because DOS imposes a maximum length of 128 characters for lines that can be entered at the keyboard.

If you make a mistake, you may backspace to erase the error. On windowed versions (Macintosh, Windows, Motif) you can move the cursor with the mouse at any time to make corrections anywhere in the command line. In most versions, you can also use the arrow keys to back up to edit the command line.

Terminating MacAnova

Type 'quit', 'bye', 'end', 'stop' or 'exit' at the prompt. In the the windowed versions (Macintosh, Windows, Motif) you can select Quit on the File menu and you will be asked if you want to save the command/output window and the workspace. You can stop MacAnova unconditionally by 'quit(F)' or 'bye(F)', for example.

Interrupting MacAnova

To stop the execution of a command after it has been initiated by Return or Enter, press the interrupt key defined as follows:

System	Interrupt Key
MacIntosh	Command+period or Command+I or Interrupt on the File menu
DOS	Ctrl+C
Unix	Ctrl+C (most common)
Windows/Motif	Ctrl+I or Interrupt on the File menu.

Compound Commands

A "compound command" is a sequence of one or more commands inside curly brackets '{' and '}'. For example

```
{i <- i+1; plot(x[,i],y[,i])}
```

is a compound command. Once you have started typing a compound command

by pressing '{' the command line is not executed before you type a matching '}'. Below, '{...}' represents an arbitrary compound command.

In a compound command, you can type <cr> at any place where a semicolon ';' would be appropriate; the compound command will not be executed until you have pressed <cr> after the closing '}'.

A common mistake is to fail to terminate a compound command with '}'. MacAnova appears to be "hung up". Actually it is just waiting for you to finish what you started. Until the compound command is complete, MacAnova has no way to recogonize that you are through typing the command line. You can either type '}'<cr> or press the interrupt key and start over.

Compound commands may be nested ({...{...}...}). Nested compound commands are not executed until the outermost one has been terminated. by '}'.

MacAnova as a Language

The organization of MacAnova commands can be considered as a "functional language," in the sense that the components of commands are functions or operators which take arguments or operands as inputs and may compute values as outputs.

The arguments of (inputs to) a named command or macro are separated by commas and enclosed in '(' and ')', as in 'print(x,y,z)'. Some named commands or macros require no arguments. In this case you just put '()' after the name, as in 'getoptions()'. In MacAnova documentation, including help output, a named command is referred to by its name followed by '()', for example, 'log10()'.

If a named command "returns a value" (has a non-NULL value), it is often referred to as a "function."

All commands have a value. For example, 'sqrt(3)' has the value 1.73205080756888 and '4*atan(1)' has value 3.14159265358979.

Some commands such as print() and regress() have values which are NULL (see below). In addition, an explicitly empty command, ';;' has a NULL value. Only commands with non-NULL values are legal in algebraic expressions or comparisons.

A few functions (for example getseeds()) return an "invisible" value that can be assigned but is not automatically printed when it is not assigned.

The value of a compound command is the value of the last command in the curly brackets. If the last command is empty (';;') a compound statement has a NULL value. When a compound command has a non-NULL value it can be used in an expression. Thus '{print(x); sum(x)}/n' prints x and computes sum(x)/n, because the value of '{...}' is the value of 'sum(x)'.

Output from functions may be arithmetically combined (for example, 'cos(x) + 3*sin(y)') and the value of (output from) one function may be an argument to (input for) another, (for example, 'cos(sqrt(x+y))').

See also arithmetic and transformations.

Side Effects of Commands

Some commands have "side effects" such as printing tables or creating variables containing the results of computation. For example, although anova() returns only a NULL value, it has side effects, namely the printing of an analysis of variance table and creation of several named variables such as RESIDUALS, SS and DF.

Conditional Execution and Looping

There are several syntax elements that you can use to control which commands are executed and in what order. Here is a brief summary

```
 Conditional execution
   if(Logical){...}
   if(Logical){...}else{...}
   if(Logical){...}elseif(Logical){...}else{...}
 Looping
   for(Var,Vector){...}
   for(Var,start,end[,increment]){...}
   while(Logical){...}
 Escaping from a loop
   break, break n, breakall, breakif
```

The first '{' following 'if', 'for', and 'while' must be on the same line. See topics if, for, while, break and breakif for more details.

Types of Data

There are several types of data, including REAL, LOGICAL, CHARACTER, GRAPH, STRUCTURE and NULL. In certain output, LOGICAL, CHARACTER, and STRUCTURE are abbreviated as LOGIC, CHAR, and STRUC, respectively.

REAL data elements are numbers and can be entered from the keyboard, read from a data file, or computed by arithmetic expressions, transformations and other functions. They can be entered as numbers with or without a decimal point (for example, '-3', '1231', or '-345.321'), or in scientific notation (for example, '-3.45321e2' is equivalent to '-345.321'). It is an error to attempt to enter a number that is too large to be represented in the computer, say 1.23e30000. On most machines the largest numbers are about +-1.797693e+308.

A missing value is represented by a special internal code usually referred to as MISSING. When entering data at the keyboard, you can specify a missing value by '?'. On output, the default is for a missing value to be printed as "MISSING" but you can specify a different output string, say "?", by setoptions(missing:"?") (see setoptions()). Virtually all commands and operations pay at least token attention to MISSING, although at present nothing is done that is more complicated than omitting cases with MISSING or setting to MISSING a result item corresponding to a MISSING input item.

You can do arithmetic on REAL data using the arithmetic operators '+', '-', '*', '^' or '**', and '%%' (for example, a * (b + c)). See arithmetic.

You can compare REAL data items using comparison operators '<', '>', '==', '!=', '<=', and '>='. See logic.

LOGICAL data have values limited to True, False and MISSING and are entered and printed as 'T' or 'F'. Comparison expressions (for example 'a < 3') generate LOGICAL data as values.

You can combine LOGICAL variables and expressions using logical operators '&&', '||' and '!'. For example, '(x>3) && (x<=5)' would be True if and only if both (x>3) and (x<=5) are True, that is if 3 < x <= 5. Similarly, '(x > 0) || (abs(x) == 3)' is True if and only if x > 0 or |x| = 3 (or both).

You can use LOGICAL data in arithmetic expressions and comparisons, with True and False being translated as 1 and 0, respectively. Thus F*T is 0, 2*T is 2, and F < T is True . Logical variables can also be used in place of REAL variables as argument to some, but not all functions such as sum(), prod() and max(). See logic.

A CHARACTER data element consists of a sequence of characters, that is letters, numbers, punctuation or anything else that can be typed. It is sometimes called a "string." When entering CHARACTER data you must enclose each string in double quotes as in

```
   greetings <- "Hello!"
```

The opening and closing double quotes are not part of the string. You you include a double quote in a string by prefixing ('escaping') it with '\'. For example,

```
  Cmd> a <- "He said, \"Hello\""
```

assigns the string 'He said, "Hello"' to variable a. You can include characters '\', newline and tab by '\\', '\n' and '\t', respectively, using a convention borrowed from Unix. For example,

```
  Cmd> b <- "1\t2\t3"
```

assigns to b the string consisting of '1', '2', and '3' separated by tab characters.

Once you have started typing a CHARACTER string with '"', MacAnova interprets everything, including <cr>, up to the next (non-escaped) '"' as part of the string. It does not recognize the command line to be complete until you have typed the closing '"'.

A common mistake is the forget to terminate a string with '"'; MacAnova appears to "hang", doing nothing. You need to type a closing '"' and terminate the line or press the interrupt key and start again.

You can include in a string any character, even one you cannot type directly, using the so called escaped octal representation of its internal (ASCII) code. Thus, since 1*8 + 5 = 13, '15' is the octal (base 8) representation of 13 and "\15" or "\015" is the character

(usually CR) with code 13. Similarly, because '123' is the octal representation for 1*8*8 + 2*8 + 3 = 83, the ASCII code for 'S', "\123" is equivalent to "S".

You can compare CHARACTER data items using comparison operators '<', '>', '==', '!=', '<=', and '>='. The ordering of letters is based on their ASCII representation with "A" < "B" < ... < "Z" < "a" < ... < "z". For example, '"A" < "B"', '"a" < "B"' and '"foo" == "bar"' have values True, False and False, respectively. See topic variables for more detail.

See below for information on structures, GRAPH data and NULL data.

Assignment of Values to Variables

You assign values to a variable using the left pointing arrow '<-' made up of the two characters "less than" and "minus". For example, 'foo <- 5' assigns the value 5 to the variable foo. If foo did not previously exist, it is created; otherwise, its previous value is discarded and foo is re-defined. An expression of the form 'x <- 3', say, is always interpreted as 'x <- 3' rather than as 'x < -3'. If you want the latter, be sure to put a space before '-3'.

You can string several assignments together. For example,

```
  Cmd> a <- b <- 1
```

assigns 1 to b and then the new value of b is assigned to a.

See topic 'assignment' for information about the value of an assignment statement and topic arithmetic about arithmetic assignment operators <-+, <--, <-*, <-/, <-%% and <-^.

Organization of Data

REAL, LOGICAL, or CHARACTER data may be organized as scalars, vectors, matrices, or arrays. The transpose of a matrix or array x may be typed as x' or as t(x). Matrix multiplication operators (applicable only to REAL data) are %*%, %c%, and %C%.

See also: vectors, matrices, vector(), matrix(), array(), dim(), ndims(), ismatrix(), nrows(), ncols().

Data may have vectors of labels for each coordinate. In particular, matrices may have row and column labels. Labels propagate through operations and functions in a fairly sensible way. Labels are primarily used in output . See topic 'labels' for information.

Subscripts

You refer to an element or set of elements of a vector, matrix, or array by using subscripts -- numbers or variables enclosed in square brackets '[...]' immediately following the variable name. For example, 'x[3,4]' is the element in row 3 and column 4 of matrix x, 'x[vector(1,3,5),4]' consists of rows 1, 3 and 5 of column 5 of x, 'x[3,]' is row 3 of x and 'x[,4]' is column 4 of x. You can assign values to subscripted elements, as in 'x[3,4] <- 17' or 'x[3,] <- 5'. See matrices, matrix(), array(), transpose, subscripts.

Structures

Structures (data of type STRUCTURE) consist of several named components. You can create a structure using structure(), strconcat() and split(), and modify a structure by changestr(). Some commands such as describe(), eigen(), or tabs() return structures as values.

The components of a structure may be data of any type, say REAL or LOGICAL, or may themselves be structures.

You can extract individual components of a structure by name or by using subscripts. For example

```
Cmd> a <- str$weight; b <- str[2]
```

assigns a component of str named 'weight' to a and the second component of str to b.

You may use structures as arguments to many functions, including transformations, and as operands to arithmetic, comparison or logical operators such as '+', '*', '==', '<=' or '&&'. See arithmetic, logic.

See structures, structure(), strconcat(), changestr(), compnames(), ncomps(), arithmetic.

GRAPH Data

A data element of type GRAPH encapsulate all the information needed to draw a graph or other plot. You can display the plot in GRAPH variable GraphVar by 'showplot(graphVar)'. See topic graphs for more information on GRAPH variables.

NULL Values

A NULL value or variable has no associated data of any kind. It cannot be used in arithmetic, or as an argument to most functions. Exceptions are vector(), hconcat(), vconcat(), strconcat(), structure(), and most of the isxxxx() functions such as isreal() and isnull(). The value returned by commands such as print() or regress() is NULL. A subscript can be NULL with the interpretation that, say, a[NULL] is itself NULL.

You can use the special constant NULL in any context when you might need a NULL value. For example,

```
Cmd> var <- NULL
```

creates variable var as a NULL variable. You cannot assign anything to NULL. Thus 'NULL <- var' is illegal.

NULL is not the same as 0; it cannot be used in arithmetic or compared to other values.

Combining Data Items

There are commands to combine several data items into a larger vector, matrix, or structure. See vector(), hconcat(), vconcat(), structure(), strconcat().

Arithmetic Assignment

You can combine assignment with an arithmetic operator using symbols

'<-+', '<--', '<-*', '<-/', '<-%%', and '<-^' or '<-**'. For example,

```
  Cmd> a <-/ 3
```

and

```
  Cmd> a <- a/3
```

are equivalent. See assignment, arithmetic.

Keyword Phrases

Some command arguments can be "keyword phases" in the form 'keyword:value'. For example, on several commands that write numbers, the argument 'nsig:8' specifies that up to 8 significant digits are to be printed and 'format:"18.13g"' specifies a format with width 18 and 13 significant digits. Particularly common are LOGICAL keyword phrases like 'keep:T' or 'coefs:F'. These enable (T) or suppress (F) alternative actions of a command. See topic keywords for details.

Indirect Reference to Variables and Constants

<<String>>, where String is a quoted string or CHARACTER variable whose value is the name of a variable, refers indirectly to that variable. For example, '<<"cos">>(PI/4)' and '<<"E">>+3' are equivalent to 'cos(PI/4)' and 'E+3', and, after the command 'a <- vector("x1","x2","x3")' creates a CHARACTER vector a, '<<a[2]>> <- 3' is equivalent to 'x2 <- 3'.

The following line creates variables x1, x2, and x3 from the columns of matrix x.

```
  Cmd> for(i,run(ncols(x))){<<paste("x",i,sep:"")>> <- x[,i];;}
```

See topics paste(), for and subscripts.

Indirect reference works even with keywords and structure component names. For example, setoptions(<<"nsig">>:5) is equivalent to setoptions(nsig:5) and, when Str is a structure with a component named 'x', Str$<<"x">> is equivalent to Str$x.

If String is "?", "T", "F", or "NULL", the value of <<String>> is MISSING, True, False, or a NULL variable. If String represents a number, <<String>> is the value of the number. Thus 10*<<"-123.456">> has value -1234.56. If String represents a CHARACTER scalar of the form "\"string without non-escaped quotes\", <<String>> is equivalent to "string without non-escaped quotes". For example, "ABCD" == <<"\"ABCD\"">> is True.

More generally, String can contain one or more MacAnova expressions or commands. In this case the commands are executed and the value of <<String>> is the value of the last command in String. In this case, <<String>> essentially does the same as evaluate(String). See evaluate().

```
  Cmd> <<"sqrt(2*PI")>>
  (1)       2.5066
```

On a Macintosh, you can use Option+\ and Option+| instead of << and >>, respectively.

Re-executing a Command Line

Just before MacAnova accepts a new command line, the immediately preceding command line is saved as macro LASTLINE. This allows you to re-execute the immediately preceding command line by typing LASTLINE() Alternatively, you can used pre-defined macro redo: redo() makes a copy of LASTLINE as macro REDO and then executes REDO. You can subsequently re-execute the same line one or more additional times by typing REDO(). You cannot use redo on two successive lines. See macros, redo.

On machines where macro 'edit' is defined, you can edit the immediately preceding line and re-execute the modified version by

```
Cmd> REDO <- edit(LASTLINE); REDO()
```

See edit.

See also comments.

2.273 t2int

Keywords: probabilities, descriptive statistics, confidence intervals

Usage: t2int(x1,x2,Coverage [, pooled:F]), x1 and x2 REAL vectors, 0 < Coverage < 1

t2int(x1,x2,Coverage) computes a (two sided) t confidence interval with coverage rate Coverage for mu1 - mu2, where mu1 is the population mean of the data in vector x1 and mu2 is the population mean of the data in vector x2. A pooled estimate of the standard error of the difference is used. This assumes equal variances.

Coverage must be between zero and one. The value is a vector of length 2 giving the lower and upper endpoints of the interval.

t2int(x1,x2,Coverage,pooled:F) or simply t2int(x1,x2,Coverage,F) computes a confidence interval for mu1-mu2 based on an unpooled estimate of the standard error and Satterthwaite's approximate degrees of freedom. It does not assume equal variances.

If there are any missing values, they are omitted from the computation and an informative message is printed

See also tint(), tval(), and t2val().

2.274 t2val

Keywords: probabilities, descriptive statistics, comparisons

Usage: t2val(x1,x2 [,df:T, pooled:F])

t2val(x1,x2) computes the two-sample Student's t statistic for testing the null hypothesis that the data in vectors x1 and x2 have the same

population mean. The usual pooled estimate of variance is used in computing the standard error of the difference of means. This assumes that the two variances are equal.

t2val(x1,x2,df:T) computes a structure with two components, 't' containing the t-statistic and 'df' containing its degrees of freedom.

t2val(x1,x2,pooled:F) or simply t2val(x1,x2,F) computes a structure containing the value of t using an unpooled estimate of the standard error of the difference of means and Satterthwaite's approximate degrees of freedom. This unpooled standard error does not assume equal variances

If there are missing values, they are omitted from the computation and an informative message is printed.

If the null hypothesis difference (mu1 - mu2) is something other than zero, say delta, then t2val(x1-delta,x2) will produce the correct t value for that null hypothesis. For example, if delta is -3, use t2val(x1-(-3),x2).

P values may be computed using the function cumstu() or the macro twotailt, for example, by

```
  Cmd> @result <- t2val(x1,x2,df:T);twotailt(@result$t,@result$df)
```

or

```
  Cmd> @result <- t2val(x1,x2,pooled:F);twotailt(@result$t,@result$df)
```

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

2.275 tabs

Keywords: categorical data, descriptive statistics

Usage: tabs(y,a,b,... [, mean:T, var:T, count:T or n:T]), y REAL, a, b, ... factors or vectors of positive integers. tabs(,a,b,...[,count:T or n:T])

tabs(y,a,b,c,...) does cross tabulation of the data in REAL vector or matrix y by the vectors or factors a, b, c, ... All the values in a, b, c, ... must be positive integers. By default, the output is a structure with components 'mean', 'var', and 'count', but this can be modified by keywords. If there are nfact factors, each component of the output is normally an array with nfact dimensions. When y is a matrix with nv columns, each component is an array with nfact + 1 dimensions, the last of which is nv.

tabs(NULL,a,b,c) or tabs(,a,b,...), with no first argument, just computes the counts in each cell, returning a vector, matrix or array.

If any of the keyword phrases 'mean:T', 'var:T', or 'count:T' is an argument after the last factor, then only that component is computed as a REAL matrix or array. If more than one of them appears, the result is

a structure with 2 or 3 components. Thus tabs(y,a,b,mean:T) computes only a matrix of cell means, while tabs(y,a,b,mean:T, var:T, count:T) is equivalent to tabs(y,a,b).

'n:T' can be used instead of 'count:T', but the corresponding component will still be named 'count'.

If y is NULL or missing, only 'count:T' or 'n:T' are permitted.

It is also acceptable for any of the factors to be a LOGICAL vector, in which case False and True correspond to factor levels 1 and 2, respectively. Thus tabs(y, x1 <= 0, x2 <= 0) will cross tabulate y according to the signs of x1 and x2.

If you want to cross tabulate data according to the values of either non-integer REAL or CHARACTER vectors, use pre-defined macro makefactor to create corresponding factors.

2.276 tek

Keywords: plotting
Usage: tek()

tek() (no argument) puts your terminal in Tektronix 4014 emulation mode if you are running MacAnova on Unix through a terminal emulator. In its standard form, tek() transmits the characters corresponding to the ASCII codes 29,27, and 56 (GS, ESC, '8'). If these are not appropriate for your terminal, tek can be redefined to send different characters. For example, to change tek so that it is appropriate for Version 2.32 or later of the public domain Kermit program for IBM-compatibles, use

```
  tek <- macro("putascii(vector(27,91,63,51,56,104))")\
          #ESC,'[','?','3','8','h'
```

For NCSA Telnet 2.6 for a Macintosh, use

```
  tek <- macro("putascii(vector(27,24))")
```

It may be necessary to put 'tek();' on the line before a plotting command to have it actually make a plot.

See tekx if you are running MacAnova in a workstation xterm window.

tek is implemented as a pre-defined macro (Unix versions only)

See also vt, vtx, plot(), chplot(), lineplot(), showplot().

2.277 tekx

Keywords: plotting
Usage: `texk()`

```
tekx() puts a Unix work station xterm terminal emulator in Tektronix
4014 mode from vt100 mode.  You don't normally need tekx since MacAnova
recognizes when it is running in a xterm environment (the value of
environmental variable $HOME is "xterm") and automatically switches to
Tektronix 4014 mode to draw a high resolution graph.

tekx is implemented as a pre-defined macro (Unix versions only)

See also tek, vt, vtx, graphs, unix.
```

2.278 time series

Keywords: time series
Usage: `Type help(time series) for information on the time series analysis capability of MacAnova`

```
Fast Fourier transforms (FFT) for Complex, Hermitian, and Real series
  cft(), hft(), rft()
Functions for working with complex and Hermitian series and Fourier
transforms
  cconj(), creal(), cimag(), cpolar(), crect(), cprdc(), cprdcj(),
  hconj(), hreal(), himag(), hpolar(), hrect(), hprdh(), hprdhj(),
  cmplx(), ctoh(), htoc(), unwind(), reverse(), padto(), rotate()
Autoregressive and moving average operators and their zeros
  autoreg(), movavg(), polyroot()
Other functions
  convolve(), partacf(), yulewalker()

Type, for example, 'usage(cft)' or 'help(cft)', to get a thumbnail
sketch or a complete description of cft().

File tser.mac, distributed with MacAnova, contains the following macros:
  gettsmacros   Eases retrieval of macros from tser.mac
  ffplot        Plot a frequency function against frequency
  tsplot        Plot time series against time
  detrend       Remove a polynomial trend
  autocov       Compute autocovariance function
  spectrum      Compute a smoothed periodogram with no tapering but with
                optional detrending
  crsspectrum   Compute a smoothed cross periodogram with no tapering
                but with optional detrending
  costaper      Compute a cosine taper with a percentage specified
  compza        Compute a cosine tapered Fourier transform, after
                optional detrending
  compfa        Compute smoothed modified periodogram, using cosine
                taper, with optional detrending
```

arspectrum	Compute autoregressive spectrum solving Yule-Walker equations
burg	Compute autoregressive spectrum using Burg's algorithm
dpss	Compute discrete prolate spheroidal sequences
multitaper	Compute multitaper spectrum estimate using dpss
evalpoly	Evaluate a real polynomial of a complex variable

These can be retrieved by, for example, getmacros(multitaper) or multitaper <- macroread("tser.mac","multitaper").

See also macros, macroread(), getmacros, help(), usage(), macrousage().

2.279 tint

Keywords: probabilities, descriptive statistics, confidence intervals

Usage: tint(x,Coverage), x a REAL vector, 0 < Coverage < 1 a REAL scalar

tint(x,Coverage) computes a (two sided) t confidence interval with coverage rate Coverage for the population mean of the data in vector x. If there are missing values, an informative message is printed.

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

Example:

```
Cmd> alpha <- .05; tint(x,1-alpha) # computes 95% conf. interval.
```

See also tval(), t2val(), and t2int().

2.280 toclip

Keywords: character variables, output

Usage: toclip(x [, missing:Code, format:Fmt, sep:C1, linesep:C2]), Code, Fmt, C1, C2 quoted strings or CHARACTER scalars

toclip(x) is equivalent to CLIPBOARD <- x and puts a possibly multi-lined CHARACTER representation of x in special variable CLIPBOARD.

When x is REAL, you can use keywords 'sep', 'missing', 'linesep', and 'format' that are permissible with paste(x,multiline:T,...).

In the Macintosh, Windows and Motif versions, toclip allows easy export of MacAnova data to another application such as a spreadsheet. For example, if x is a 10 by 5 REAL matrix that you want to export to a spreadsheet, after 'toclip(x)', you can select a 10 by 5 rectangle of

```
cells in the spreadsheet and select Paste in the Edit menu.  If the
target application has special requirements for representing MISSING or
field and line separators, you can use keywords to customize what gets
put on the Clipboard.

Examples:
  toclip(x,missing:"NA") and toclip(x,missing:"-99") put x in CLIPBOARD
    with MISSING coded as NA and -99, respectively
  toclip(x,sep:",",format:".10f") puts x in CLIPBOARD with elements
    separated by commas and with 10 decimal places
  toclip(x,linesep:":",sep:",") puts x in CLIPBOARD with elements
    separated by commas and rows separated by colons.

See also clipboard, fromclip, paste().

toclip is implemented as a pre-defined macro.
```

2.281 toeplitz

Keywords: matrix algebra
Usage: `toeplitz(x), x a REAL vector.`

```
toeplitz(x) computes an n by n Toeplitz matrix from a REAL vector x of
length n.  After a <- toeplitz(x), a is constant on the diagonals with
a[i,j] == x[|i-j| + 1].

Function toeplitz() primary use is to create a covariance matrix from an
auto-correlation function.  Thus if gamma0, gamma1, ... are the variance
and first n-1 autocovariances of a stationary time series {x[t]}, then
toeplitz(vector(gamma0, gamma1, ...))  computes the n by n covariance
matrix of the vector vector(x[1], x[2], ..., x[n]), or indeed of
vector(x[1+k], x[2+k], ..., x[n+k]) for any integer k.

See also partacf(), yulewalker().
```

2.282 trace

Keywords: matrix algebra
Usage: `trace(x), x a REAL square matrix`

```
trace(x) computes the so-called trace of REAL square matrix x, that is,
sum(diag(x)).

See also matrices, dmat(), diag(), det().
```

2.283 transformations

Keywords: transformations

Usage: abs(x), acos(x), asin(x), atan(x) or atan(x,y), atanh(x), ceiling(x), cos(x), cosh(x), exp(x), floor(x), lgamma(x), log(x), log10(x), round(x) or round(n,ndec), sin(x), sinh(x), sqrt(x), tan(x), tanh(x), x REAL or structure with REAL components

Here is a list of available transformations

Transformation		Result for REAL scalar x
abs(x)		|x| = absolute value of x
acos(x)	t	arccosine of x
asin(x)	t	arcsine of x
atan(x)	t 2	arctangent of x
atanh(x)		inverse hyperbolic arctangent of x
ceiling(x)		least integer >= x
cos(x)	t	cosine of x
cosh(x)		hyperbolic cosine of x
exp(x)		exp(x) = exponential function of x
floor(x)		greatest integer <= x
lgamma(x)		ln(gamma(x)) = log gamma function of x
log(x)		ln(x) = base-e log(x) = natural logarithm of x
log10(x)		base-10 log(x) = common log of x
round(x)	2	nearest integer to x
sin(x)	t	sine of x
sinh(x)		hyperbolic sine of x
sqrt(x)		square root of x
tan(x)	t	tangent of x
tanh(x)		hyperbolic tangent of x

t: trigonometric function, affected by option 'angles'; see below
2: Has two argument variant, see atan() and round().

Argument x must be REAL or structure of REALS, or can be CHARACTER (see below).

If x is a REAL vector, matrix or array, the result is REAL with the same size and shape, with the elements of the result the transformations of the elements of x. For example, if x is a matrix, exp(x)[i,j] is exp(x[i,j]).

By default, sin(), cos() and tan() interpret their argument as being in radians and the values of asin(), acos() and atan() are in radians. setoptions(angles:"degrees") or setoptions(angles:"cycles") changes this default. See setoptions().

If the argument to a function is illegal (for example, sqrt(-1) or atanh(1.2)), a warning message is printed and the result is set to MISSING.

If the value of a function is too large to be represented in the computer (for example, sinh(-3000)), a message is printed and the result

is set to MISSING.

Because of significant loss of precision in computing trigonmetric functions of a large argument, the result of sin(x), cos(x) or tan(x) is MISSING when |x| >= 5000000*PI radians (= 2500000 cycles = 900000000 degrees) and a warning message is printed.

If the argument x to a transformation is a structure, the result is a structure of the same shape, whose components are the transformed components of the argument.

If x or any components have labels, the result has the same labels.

If the argument x to a transformation is a CHARACTER variable, the result is a CHARACTER variable of the same size and shape with elements involving the transformation name and the elements of x. For example, log(vector("X", "Y")) is vector("log(X)","log(Y)"). This feature can be helpful in creating labels for transformed data.
Example:

```
  Cmd> logx <- matrix(log(x),labels:structure(getlabels(x,1),\
       log(getlabels(x,2))))
```

This uses the row labels of x and transforms the column labels of x.

See also arithmetic, syntax, atan(), hypot(), boxcox, round(), structures, rational().

2.284 transpose

Keywords: matrix algebra, operations
Usage: x' or t(x), where x is a matrix

x' (x followed by a single quote or "prime") computes the transpose of x if x is a matrix, that is the matrix y with y[i,j] = x[j,i].

If x is a vector of length n, x' is a 1 by n matrix, that is, a row vector.

If x is an array with dimensions n1, n2, ..., nk, then y <- x' computes an array y with dimensions nk, ..., n1 such that y[i1,...,ik] is x[ik,...,i1]. If x is a generalized matrix (see matrices), so is x', and matrix(x)' = matrix(x').

t(x) is synonymous with x', but this usage is deprecated. At some point, function t() will probably be removed.

Instead of x' %*% y and x %*% y' you can use x %c% y and x %C% y, respectively which use less internal memory.

If x is a structure, each of whose components is REAL, LOGICAL, or CHARACTER, x' computes a structure whose components are the transposes of the corresponding components of x.

See also array(), matrices, subscripts.

2.285 trideigen

Keywords: matrix algebra, time series
Usage: trideigen(Diag, Subdiag [[, start] , end] , values:F or vectors:F), Diag, Subdiag REAL vectors, start and end positive integers

trideigen(Diag, Subdiag) computes the eigenvalues and eigenvectors of the symmetric tri-diagonal matrix with diagonal Diag and sub- and super-diagonal Subdiag. Diag and Subdiag must be REAL vectors with length(Subdiag) = n - 1 or length(Subdiag) = n, where n = length(Diag). In the latter case, Subdiag[1] is ignored. The result is a structure with components 'values' and 'vectors', similar to eigen. The eigenvalues are returned in decreasing order.

trideigen(Diag, Subdiag, vectors:F) computes only the eigenvalues, returning a vector of length n.

trideigen(Diag, Subdiag, values:F) computes only the eigenvectors, returning a n by n matrix.

trideigen(Diag, Subdiag, start, end) computes the i-th eigenvalues and eigenvectors, for i = start, ..., end. trideigen(Diag, Subdiag) is equivalent to trideigen(Diag, Subdiag, 1, length(Diag)).

trideigen(Diag, Subdiag, end) is eqivalent to trideigen(Diag, Subdiag, 1, end).

Command trideigen() was added specifically to make it straightforward to compute discrete prolate spheroidal sequences (dpss) used in multi-taper spectrum analysis. For these, if W is the desired width, the following computes the first K dpss

```
d <- cos(2*PI*W)*(.5*run(-n+1,n-1,2))^2
e <- .5*run(0,n-1)*run(n,1)
dpssvecs <- trideigen(d,e,K,values:F)
```

The advantages of trideigen() over eigen() are (a) you do not need to create the n by n tridiagonal matrix, and (b) you can obtain a subset of the eigenvalues and eigenvectors.

See also eigen(), releigen().

2.286 trilower

Keywords: matrix algebra, variables, combining variables
Usage: trilower(A), A a matrix

trilower(a) returns a matrix d of the same size and shape as a with d[i,j] = a[i,j] for i >= j (on or below the diagonal) and d[i,j] = 0 for i < j (above the diagonal). Variable a must be a matrix but need not be square.

For example, when a is

1	5	9
2	6	10
3	7	11
4	8	12

, trilower(a) is

1	0	0
2	6	0
3	7	11
4	8	12

.

If a has type CHARACTER, elements above the diagonal are sent to empty strings ("") instead of 0's.

trilower(a,T) or trilower(a,pack:T) returns the lower triangle (elements a[i,j] with i >= j) of a in packed form. Matrix a must be square, that is, nrows(a) = ncols(b). For example, when

a =

1	4	7
2	5	8
3	6	9

,

trilower(a) is vector(1, 2, 5, 3, 6, 9).

Note: keyword 'square' is not valid for trilower().

See also triupper(), triunpack(), qr().

2.287 triunpack

Keywords: matrix algebra, variables, combining variables
Usage: triunpack(vec [, lower:T or upper:T]), vec a vector of length p(p+1)/2

triunpack(v) creates a square matrix from vector v whose length must be of the form p*(p+1)/2, that is, a so-called 'triangular number'. The result is a p by p symmetric matrix whose upper triangle (including the diagonal) has packed form v. Thus triunpack(run(10)) produces the matrix

1	2	4	7
2	3	5	8
4	5	6	9
7	8	9	10

triunpack(v,lower:T) and triunpack(v,upper:T) produce unpacked lower and upper triangular matrices. Thus, for example,

```
  Cmd> triunpack(run(10), upper:T) # or simply triunpack(run(10),T)
```

and

```
  Cmd> triunpack(run(10),lower:T)
```

produce

```
1   2   4   7          1   0   0   0
0   3   5   8          2   3   0   0
0   0   6   9   and    4   5   6   0 , respectively.
0   0   0  10          7   8   9  10
```

When v is a CHARACTER vector, empty strings ("") are used instead of 0's in the other half.

See also triupper(), trilower(), qr().

2.288 triupper

Keywords: matrix algebra, variables, combining variables
Usage: triupper(A [,pack:T]), A a matrix

triupper(a) returns a matrix d of the same size and shape as a with d[i,j] = a[i,j] for i <= j (on or above the diagonal) and d[i,j] = 0 for i > j (below diagonal). Variable a must be a matrix but need not be square.

```
                        1   5   9                    1   5   9
For example, when a is  2   6  10 , triupper(a) is   0   6  10
                        3   7  11                    0   0  11
                        4   8  12                    0   0   0
```

triupper(a,square:T) returns the m by m upper left block of triupper(a), where m = min(nrows(a),ncols(a)).

If a has type CHARACTER, elements below the diagonal are sent to empty strings ("") instead of 0's.

triupper(a,T) or triupper(a,pack:T) returns the upper triangle (elements a[i,j] with i <= j) of a in packed form. Matrix a must be square, that is, nrows(a) = ncols(a). For example, when

```
              1   4   7
        a =   2   5   8
              3   6   9
```

triupper(a,pack:T) is vector(1, 4, 5, 7, 8, 9).

See also trilower(), triunpack(), qr().

2.289 tval

Keywords: probabilities, descriptive statistics, comparisons
Usage: tval(x [,df:T]), x a REAL vector

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

```
tval(x,df:T) produces a structure with two components, 't' containing
the t-statistic and 'df' containing the degrees of freedom.

If x contains MISSING values, an informative message is printed.

If the null hypothesis mean is something other than zero, say delta,
then tval(x-delta) will produce the correct t value for that null
hypothesis.  For example, to test H0: E[x] = 3, use tval(x-3).

P values may be computed using the function cumstu() or the macro
twotailt, for example, by
  Cmd> @result <- tval(x,df:T);twotailt(@result$t,@result$df)

See also t2val(), tint(), t2int(), cumstu(), and twotailt.
```

2.290 twotailt

Keywords: probabilities, descriptive statistics, comparisons
Usage: twotailt(tval, df), tval a REAL scalar, df > 0 REAL.

```
twotailt(tval,df) computes the two tailed P value for a Student's t
value of tval with df degrees of freedom.  It is implemented as a macro.

See also cumstu().
```

2.291 unique

Keywords: ordering, variables
Usage: unique(x [,index:T]), x REAL or CHARACTER

```
unique(x) computes a vector consisting of all the distinct non-missing
values in the REAL or CHARACTER vector, matrix, or array x.  It is an
error if all elements of x are MISSING.

unique(x, index:T) computes a vector J of positive integers such that
x[J] is the same as unique(x).  That is it finds the subscripts of the
unique elements of x.

Example: unique(vector(5,3,1,2,4,2,5,7,2,7)) yields vector(5,3,1,2,4,7).
         unique(vector("B","C","A","B","D","A","A")) yields vector("B",
         "C","A","D")
         a <- factor(match(x,unique(x)))
computes a factor each level of which corresponds to a unique value of
vector x.
         a <- factor(match(x,sort(unique(x))))
computes a factor each level of which corresponds to a unique value of
vector x, with the factor levels in the same numerical or alphabetic
order as the elements of x
```

See also match().

2.292 unix

Keywords: general
Usage: Type help(unix) for a summary of features specific to Unix versions of MacAnova.

Two Unix versions are distributed. One runs from the Unix prompt (usually under xterm on a workstation) with no special windows except those generated by the operating system. The other (Motif) uses the Motif windowing system and operates similarly but not identically to the Macintosh version. It allows multiple command and high resolution graphics windows.

Features common to all Unix versions

Various command line arguments are recognized. These allow automatic restoring of a workspace, suppressing the startup message, etc. See topic launching.

MacAnova uses any default options or file or path names in environmental variable MACANOVA. See topic customize.

The startup file is .macanova.ini in the users home directory unless flag -f has been used on the command line (see launching) or in environmental variable MACANOVA. See topic customize.

The size of variables is limited only by the amount of memory available.

Unless you use command line option -home (see launching) or include -home in environmental variable MACANOVA (see customize), MacAnova pre-defines CHARACTER variable HOME to contain user's home directory. HOME is used by to expand file names of the form "~/path" or "~\\path" by substituting the value of HOME for '~'. This allows you to refer to files such as .macanova.ini as "~/.macanova.ini", even if you have changed directories. If you redefine HOME, it changes the expansion of "~/". See topic files.

File names starting with "~name/" are expanded similarly to csh. That is, name is taken to be the name of a user and "~name" is expanded to the path name of the home directory of that user. Redefining HOME has no effect on this expansion. See topic files.

Pre-defined variables DATAPATH, and DATAPATHS are initialized with path names that are installation dependent. You can override these using options -dpath or -mpath on the command line (see launching) or in environmental variable MACANOVA (see customize).

Non-Motif Versions

High resolution graphics are implemented by emitting codes appropriate

for plotting on a Tektronix 4014 graphics terminal. If this is not appropriate, you are limited to displaying "dumb" plots and should use setoptions(dumbplot:T)

If running in a xterm pseudo VT100 window on a work station, the graphics commands open a pseudo Tektronix 4014 graphics window and draw in it, switching back to the VT100 window when <RETURN> is hit after the plot.

You can execute Unix commands by prefixing the line with '!' in the first position after the prompt or by using command shell(). Keyword phrase keep:T on shell() is recognized. You must use keyword phrase interact:T if the program being invoked expects any input from you. A line of the form '!command ... ' is equivalent to shell("command ...", interact:F). See shell().

A pre-defined macro edit is available allowing editing of macros and data from within MacAnova.

Depending on how it was compiled on a particular system, keyboard line editing and history may be available as implemented using the GNU Readline Library. If so, the default editing mode is based on Emacs. You can customize key bindings by creating a special file named ".inputrc" in your home directory. This will be read each time MacAnova starts up. To enable editing based on Vi commands instead of Emacs, put the following lines in this file:

```
set editing-mode vi
"k": previous-history
"j": next-history
"H": beginning-of-history
"G": end-of-history
```

When keyboard line editing is implemented, a "history" mechanism is also available. A certain number (default is 100) of previous commands are saved. Using the emacs editing mode, you can scroll backward by pressing Ctrl+P (possibly also an up-arrow key) and scroll forward by pressing Ctrl+N (down-arrow). Under the vi editing mode, the corresponding keys (when in vi command mode) are 'k' amd 'j'. See setoptions() for information on how to change the number of lines saved.

As part of this facility you also may have so-called file name completion. If you have partially typed a file name, press the Tab key or press the Esc key twice and an attempt will be made to complete it. Press Tab twice or Esc four times and you will get a list of possible completions. See Readline documentation for information on modifying key bindings using file .inputrc. If you want to use the Tab key as a regular key, but the following line in the .inputrc file: "C-i": self-insert

Motif Version

This version is compiled using the WxWin windowing library so as to use the Motif windowing system. It allows up to eight command/output windows with Copy, Paste and Undo capability and up to eight high

resolution graphics windows.

It has File, Edit and Windows menus which are patterned after the Macintosh version.

Entering and editing of commands is done in a command/output window with the mouse and keyboard.

Each command/output window can be printed, saved to a file or some or all of its contents Copied to the clipboard which is "connected" to special variable CLIPBOARD (see topic clipboard). In a similar manner, special variable SELECTION is "connected" to the X selection. This means you can select data or text in another Window and import it into MacAnova.

Each graphics windows can be printed or Copied to the clipboard.

Currently neither shell() or lines prefixed with '!' are implemented satisfactorially. They do not work under Windows 3.1 or Windows 95 but may work under Windows NT.

See topic wx for details.

2.293 unwind

Keywords: time series, complex arithmetic
Usage: unwind(angleMat [, crit:Val]) where angleMat is REAL matrix and .5 <= Val < 1

unwind(angleMat) attempts to eliminate or reduce discontinuities (jumps between rows) in each column of matrix angleMat, considered as having angles as values. The assumed units of the angles are as specified by option 'angles' ("radians", "degrees", or "cycles"), changeable by setoptions(). For example, if angles are measured in degrees, unwind(vector(350, 352, 359, 2, 1, 355)) is vector(350, 352, 359, 362, 361, 355).

The operation of unwind() is controlled by the value Val of a criterion. Let Jump = |angleMat[i,j] - angleMat[i-1,j]|. Then if Jump > Val, angleMat[i,j] is increased or decreased by a multiple of 2*pi (360 degrees or 1 cycle) so as to bring the change less than Val. The default is Val = 0.75.

unwind(angleMat,crit:Val) uses Val for the unwinding criterion. Val must be between 0.5 and 1 cycles. It is always in units of cycles regardless of the value of option 'angle'.

See also hpolar(), cpolar(), setoptions().

2.294 usage

Keywords: general
Usage: usage([Topic1,Topic2,...] [,file:FileName or orig:T or alt:T])
usage(Pattern) where Pattern has form "part*", "*part", or "*part*".
usage(key:KeyNames), where KeyNames is a CHARACTER vector or "?"

usage(topic) gives short usage information on topic, where topic is a function or command.

Function usage() works identically to help(), except it gives a brief summary of the usage of commands, functions, and macros, instead of all the detail. For some topics, no short information may be available. In that case, usage() gives the same information help() would.

See help() for full details.

2.295 User

Keywords: general, control, files
Usage: User(funName [,resource:resName][,control keyword phrases],arg1 [,...]), funName and resName CHARACTER scalars; control keyword phrases are any of callback:T, symbols:T, pointers:T and quiet:T; arg1, ... arguments to a user function; if argument is keyword phrase other than 'protect:arg', it is returned, possibly modified.

User(FuncName, arg1, arg2, ...) executes a user function, that is, a compiled routine external to MacAnova. Quoted string or CHARACTER scalar FuncName specifies the name of a user function whose code is in a file previously loaded by loadUser(). arg1, arg2, ... are arguments that will be passed to the function. You must have a least one argument in addition to FuncName and no more than 20 (13 in the Macintosh PPC version). Depending on the compiler and system, you may be required to include leading or trailing underscore characters '_' in FuncName, say User("foo_",...) or User("_foo", ...) instead of User("foo", ...).

A 68K version of MacAnova cannot execute a user function compiled for a PPC.

User(FuncName, quiet:T, arg1, ...) does the same except warning messages, if any, are suppressed.

User(FuncName, callback:T, arg1, ...) specifies the function is known to "call back" to MacAnova, that is, to execute functions internal to MacAnova. On a Macintosh, the type of user function (PPC or ordinary 68K) must match the version of MacAnova. See topic callback fun in file

Userfun.hlp (type help(file:"Userfun.hlp", callback fun)).

If the MacAnova version requires a 68881 math coprocessor, there can be problems if a user function that makes call backs does not require a coprocessor.

User(FuncName, symbols:T, arg1, ...) specifies that all the arguments are to be passed as complete MacAnova "symbols", including all dimension information. This should be used only with a user function specifically written to make use of MacAnova symbols.

The PPC Macintosh version cannot pass symbol arguments to a 68K user function.

User(FuncName, pointers:T or F, arg1, ...) changes the default way arguments are passed, either as "pointers" (pointers:T) or as "handles" (pointers:F). On all but Macintosh computers, the default is pointers:T. You are unlikely ever to need to use this keyword.

User(FuncName, resource:ResName, arg1, ...) specifies the name of the PPC Macintosh resource containing the user function. This option is not needed in other versions and needed on a PPC only when the resource name differs from the function name.

You can use more than one of the preceding keywords phrases together (User("goo", resource:"foo",quiet:T,callback:T,x,result:0)).

On most systems, it is possible to include with a user function an "arginfo" function that MacAnova can call to obtain information about the user function. The information includes the number of arguments expected, their types and shapes expected (for example, CHARACTER scalar, REAL matrix), and whether the function makes call backs or expects "symbol" arguments (see above). This allows automatic argument checking. If function is compiled without an arginfo function, using the wrong number or type of arguments will usually result in a crash or other undesirable behavior. In particular, a user function will not be able to handle MacAnova symbol arguments unless it has been specially written to be able to understand their structure.

You normally do not need to use keywords 'callback', 'symbols' and 'pointers' if the user function has an associated arginfo function.

See topics user fun and arginfo fun in help file Userfun.hlp for information about the form of a user function and and an arginfo function (type help(file:"Userfun.hlp", user fun), say).

Interpretation of FuncName

Unix, Motif and Windows:

FuncName should be the name of the function being called, possibly modified by a leading or trailing '_' (leading '_' when compiling for Windows with Borland C 4.5).

Extended memory DOS (DJGPP):

FuncName should be the same as the name of the .dxe file loaded by loadUser() that contains the code except that directory information and the extension ".dxe" may be omitted. Thus after loadUser("../foo.dxe"), you can use any of User("../foo.dxe",...), User("foo.dxe",...) or User("foo",...). If there is more than one file with the same name attached (for example, "/a/foo.dxe" and "/b/foo.dxe", you should use the complete path name.

PPC Macintosh user function
FuncName is the name of the user function. This will usually also be the name of the PPC code resource containing the user function. If it is not, you need to include keyword phrase 'resource:ResName' as an argument, where ResName is a quoted string or CHARACTER scalar specifying the resource name.

68K Macintosh user function:
FuncName should be the name of the 68K code resource containing the user function (only one user function per resource). If 'resource:ResName' is an argument, ResName must be identical with FunName. It is an error to attempt to call a 68K user function that requires a 68881 or 68882 math coprocessor on a Macintosh without one.

User function arguments

All arguments except the function name and 'callback', 'quiet', 'symbols', 'pointers' and 'resource' keyword phrases are passed to the user function as its arguments

Only copies of keyword phrase arguments are passed to a user function. This means that the user function can modify these arguments without danger of changing any MacAnova variable.

Non-keyword phrase arguments to User() (except the user function name) are passed to the user function without being copied. If the argument is a named MacAnova variable and the user function modifies it, the value of the variables itself is changed. If the argument is a literal number (User("foo", 1, 2, 0)) or expression (User("foo", sqrt(2)+3, log(4), 19^2)) the function can safely change the argument without danger to any variable.

Example:
Suppose fooadd expects three arguments, and modifies the third by assigning the sum of the first two. Then

```
Cmd> c <- 0; User("fooadd", 1, 2, c)
```

returns no value (actually a NULL), but c has been changed to 3 (= 1+2). However,

```
Cmd> c <- 0; User("fooadd", 1, 2, protect:c)
```

will not change c itself, but only a copy.

In addition to being copied before use, all keyword phrase user function arguments are returned, possibly modified, as the value of User(). When there are two or more such keyword arguments, a structure is returned with component names taken from the keyword names.

Keyword 'protect' is special, in that its only effect is to cause its value to be copied before being passed to the user function; its value is not returned by User(). Thus the use of 'protect' in the example makes the user function useless: c does not get changed because it is protected by a keyword, but the modified value is not returned either. The following both protects c and causes the modified value of the last argument to be returned as the value of User().

```
Cmd> c <- 0;User("fooadd", 1, 2, result:c)
```

This returns 3 = 1 + 2, but c would be unchanged. In place of variable c for result, you could use any REAL scalar (result:0). This serves to provide space for the answer.

```
Cmd> User("fooadd", left:1, right:2, result:0)
```

returns a structure with components 'left', 'right' and 'result' containing the possibly modified values of the original arguments.

It is essential that the size of any argument that is to be modified matches the size that is expected by the user function. Thus, if foocat is a user function that concatenates its first two arguments into a third argument, the length of the third argument should be the combined length:

```
Cmd> User("foocat", run(4), run(7), combined:rep(0,11)
```

In this example, for the third argument to have fewer than 11 elements would lead to unpredictable results, possibly even a crash of MacAnova.

Except when symbols:T is an argument, all user function arguments are either REAL, LOGICAL, CHARACTER or LONG variables. REAL arguments are passed as double precision data, as are LOGICAL arguments (True = 1.0, False = 0.0). If an argument is a matrix or array, the values are ordered such that the first subscript changes fastest. See user fun in help file Userfun.hlp for more information (type help(file:"Userfun.hlp", user fun)).

LONG is a special MacAnova type whose values are long integers between -2147483647 and 2147483647 = 2^31 - 1. A LONG argument can be created only by function asLong(). A LONG argument that is returned (result:asLong(x)), is turned into an REAL quantity before being returned. See asLong().

```
Cmd> User("goo", run(10), run(10), asLong(10), result:0)
```

invokes user function goo with three REAL arguments and one long argument.

For virtually unlimited flexibility, when keyword phrase 'symbols:T' is an argument to User(), all user function arguments are passed as symbols -- MacAnova objects which encapsulate the data, type, and dimensions of a variable. Thus

```
Cmd> User("foo", symbols:T, x, y, result:z)
```

passes x, y and z to 'foo' as symbols. You cannot have some user function arguments be symbols and some just data; all must be symbols or none.

2.296 user fun

Keywords: general, control, files

Usage: Type help(file:"Userfun.hlp",user fun) for information on the structure of user functions. Type help(file:"Userfun.hlp",callback fun) for information on the structure of user functions making "call backs" to MacAnova. Type help(file:"Userfun.hlp",arginfo fun) for information on how to enable automatic checking of arguments to a user function.

This topic is now in file Userfun.hlp. Type

```
help(file:"Userfun.hlp", user fun)
```

It provides a brief introduction to the form of a user function (routine compiled separately from MacAnova) that can be loaded by loadUser() and executed by User().

Some other useful entries in Userfun.hlp are callback fun and arginfo fun. Type

```
help(file:"Userfun.hlp","**")
```

for a complete list of entries.

2.297 variables

Keywords: syntax, variables, character variables, logical variables

Usage: Type help(variables) for details about (1) names of permanent and temporary variables, (2) types of variables, (3) invisible variables and (4) coordinate labels.

Data are stored in permanent or temporary "variables" with names of up to 12 characters. Variables can be permanent (the variable remains accessible until you delete it) or temporary (the variable is automatically deleted at the next prompt). Temporary variables are particularly useful in macros. See macro() and macros.

Names of permanent variables must begin with a letter (a-z or A-Z) or the character '_', possibly followed by letters, numerals or '_'. For example 'x12a' and 'time_of_day' are names of permanent variables. The use of variable names that start with '_' is discouraged since the variable is then "invisible" (see below).

Names of temporary variables start with '@' followed by a letter or '_', possibly followed by letters, numerals or '_'. Examples are '@x1a' and '@_Result. A variable whose name starts with '@_' is "invisible".

Names are case sensitive (for example, 'residuals' is a different name from 'Residuals'), and names in all capital letters (for example 'RESIDUALS') should be avoided.

Typically you will select names that are relevant to the problem such as 'weight', 'residuals', or 'depv'.

An "invisible" variable differs from a "visible" variable in two ways. (i) Commands list() or listbrief() ignore the variable unless keyword phrase 'invis:T' is an argument (see listbrief(), list()); and (ii) typing the name does not result in the variable's value being printed.

See topic clipboard for information on the special variables CLIPBOARD and SELECTION.

Types of Variables

There are several types of variables, including REAL, LOGICAL, CHARACTER, GRAPH, STRUCTURE, MACRO and NULL. In certain output, LOGICAL, CHARACTER, and STRUCTURE are abbreviated as LOGIC, CHAR, and STRUC, respectively.

Variables of a special type LONG can be created by function asLong() but they exist only transiently, being "coerced" to equivalent REAL variables when assigned. See asLong().

A REAL variable contains numerical data or the special value MISSING. A LOGICAL variable contains data with values limited to True, False or MISSING (see logic). A CHARACTER variable contains data consisting of character information. A LONG variable contains integer values between -2147483647 and +2147483647 = 2^31 - 1.

REAL, LOGICAL, CHARACTER or LONG variables may be scalars (consist of a single data item), vectors (several data items indexed by a single subscript; see vectors), matrices (data items indexed by two subscripts; see matrices), or arrays (data items indexed by more than two subscripts; see array()). See also topic subscripts.

A GRAPH variable encapsulates all the information needed to draw a graph. See topic graphs.

A STRUCTURE variable or simply a structure consists of named components of data which may be of any type, including STRUCTURE. See structures.

A MACRO variable or simply a macro contains one or more MacAnova commands to be executed together. See macros, macro().

A NULL variable contains no data of any sort.

Coordinate Labels

REAL, LOGICAL and CHARACTER variables may have vectors of labels for each coordinate. In particular matrices may have row and column labels. A structure may have labels for each component. Labels propagate through operations and functions in a fairly sensible way. Labels are primarily used in output . See topic 'labels' for details.

2.298 varnames

Keywords: variables, glm, character variables
Usage: varnames(Model) where Model is CHARACTER scalar
varnames() is equivalent to varnames(STRMODEL)

varnames(Model) returns as a CHARACTER vector the names of the reponse variable as well as the factors and variates in Model, a CHARACTER variable or quoted string.

No checking is done to confirm that the model is of the correct form except for checking that no names have length greater than 12 characters. In particular, variables specified by {expr}, where expr is a MacAnova expression, will not be understood. Thus varnames("y a b c") and varnames("y=a+b+a.b.c") both return vector("y","a","b","c"), and varnames("y={sqrt(x)} + a") returns vector("y", "sqrt", "x", "a").

varnames() without a model is equivalent to varnames(STRMODEL), returning a vector of the names of the variables in STRMODEL. This is usually the model used by the most recent GLM (generalized linear or linear model) command such as regress(), anova(), or poisson().

See also xvariables(), modelvars(), models, compnames(), nameof(), glm.

2.299 vboxplot

Keywords: plotting, descriptive statistics
Usage: vboxplot(x1,x2,...,xk [, graphics keyword phrases]), arguments REAL vectors
vboxplot(Struc, [, graphics keyword phrases]), Struc a structure with REAL vector components

vboxplot(var1, var2, ... , vark) produces vertically oriented parallel Tukey boxplots for the vectors var1 through vark. It is identical with boxplot(var1, var2, ..., vark, vertical:T).

vboxplot(Struc) produces parallel box plots for the components of structure Struc, all of which must be vectors. It is identical with boxplot(Struct, vertical:T).

For more information and the use of split() to create a structure argument see boxplot().,

2.300 vconcat

Keywords: combining variables, variables, null variables
Usage: vconcat(a,b,c,...), a, b, c, ... matrices with same number of columns.

vconcat(a,b,c,...) combines matrices a, b, c ... vertically by concatenating their columns.

All arguments must be of the same type, REAL, LOGICAL, or CHARACTER, and have the same number of columns n. The result is a matrix of that type with n columns and ma+mb+mc+... rows, where ma, mb, mc, ... are the number of rows of a, b, c, ...

Any argument that is a vector of length m is considered to be a m by 1 matrix. In particular, if a is a vector of length m, vconcat(a) is a m by 1 matrix.

An argument that is an array with only two dimensions not equal to 1 is considered to be a matrix (see matrices). Thus

```
vconcat(array(run(6),1,3,2),array(run(7,14),4,1,2))
```

is equivalent to

```
vconcat(matrix(run(6),3),matrix(run(7,14),4))
```

Any argument of type NULL is ignored. If all arguments are NULL, so is the result.

See also hconcat(), matrices, vectors.

2.301 vecread

Keywords: input, files

Usage:
```
vecread(FileName [,stop:stopChar, skip:skipChar,
silent:T])
vecread(FileName, character:T [,stop:stopChar,
skip:skipChar, silent:T])
vecread(FileName, bylines:T [,stop:stopChar,
skip:skipChar, silent:T])
Other keyword phrases: quiet:F and echo:F or echo:T.
FileName can also be of the form string:charVal where
charVal is a CHARACTER scalar or vector.
```

vecread(FileName) reads numerical data from the file with name FileName and returns a REAL vector containing the data. Argument FileName must be a quoted string or a CHARACTER variable. On a Macintosh, if FileName is "", you will be prompted to enter the file name using the usual scrolling dialog box.

For this usage of vecread(), the file should contain a sequence of numbers, separated by tabs, spaces or single commas. "?", "??", "???", ... are interpreted as representing a MISSING value. In addition isolated periods "." are interpreted as MISSING. The values are read sequentially, row by row. Unreadable items are skipped and an informative message is printed once. A badly formed number like "1.2a5" may be read as more than one number, 1.2 and 5 in this case, since the non-numerical part is treated as an unreadable item. Any number that is too large to be represented in the computer is set to MISSING. Single commas between items are ignored; a sequence of several commas is treated as an unreadable item.

vecread(FileName, badvalue:RealVal), where RealVal is a REAL scalar or MISSING (?), returns a REAL vector with RealVal substituted for every non-numerical item in the file. For example if the file looks like the following:

```
   Henry   Male   67.3,10.5
   Susan   Female 59.2,15.1
```

vecread(FileName, badvalue:?) returns vector(?,?,67.3,10.5,?,?,59.2, 15.1). If there were two or more commas between 59.2 and 15.1 they would be treated as a bad value.

vecread(FileName, character:T) reads CHARACTER data from file FileName. Keyword phrase character:T can be abbreviated to char:T.

For this usage, the file is considered as a sequence of "words" consisting of printable non-blank characters, separated by commas, or spaces, tabs or other "invisible" characters. Each element of the result is one such word. Two successive commas, with no intervening visible characters are assumed to delineate a null string (""). Quotation marks (") are not special and are treated as any other visible character that is not a comma.

If the file is above, vecread(FileName, character:T) returns

vector("Henry","Male","67.3","10.5","Susan","Female","59.2","15.1"). If there were two commas between 59.2 and 15.1, there would be a null string "" between "59.2" and "15.1" in the result. If there were 3 commas, there would be two null strings.

By default, whether reading numerical or CHARACTER data, an exclamation point ('!') is the stopping character. That is, a '!' at any point in the file terminates scanning. This may be changed by keyword 'stop'; see below.

If the file contains n lines of k items each, then it can be read into a n by k matrix by x <- matrix(vecread(FileName),k)' or x <- matrix(vecread(FileName, character:T),k)'. Note the transpose.

vecread(fileName, bylines:T) creates a CHARACTER vector each element of which is an entire line read from file fileName. This is the only usage of vecread() which can read CHARACTER data containing commas, spaces or tabs or other "invisible" characters. These can be read using matread(), provided the data is preceded by a suitable header.

With bylines:T, the stopping character ("!") must be the first character in the line.

For the data file above, vecread(FileName, bylines:T) returns
vector("Henry Male 67.3,10.5","Susan Female 59.2,15.1").

vecread(string:CharVar [, keywords]) where CharVec is a CHARACTER scalar or vector, does not read from a file. Instead, it "reads" CharVar as if each element were a line read from a file (or several lines if there are embedded end-of-line characters). In particular, x <- vecread(string:CLIPBOARD) and x <- vecread(string:CLIPBOARD, character:T) would read data from the special variable CLIPBOARD. In the Macintosh, Windows and Motif versions this would be taken from the Clipboard. Pre-defined macro fromclip() makes use of this feature to read data on the Clipboard. In the Motif version, you can use special variable SELECTION in a similar way to read the current X selection. See topic clipboard.

vecread(CONSOLE [,keywords]) or vecread("CONSOLE" [,keywords]) results in reading directly what you type rather than a file. On a Macintosh a dialog box is displayed in which you enter data; on other platforms, you are prompted to type in the data. Data should be typed in the format just described. The terminating '!' followed by RETURN is required (on a Macintosh you can click on the "Done" button instead). Thus x <- vecread(CONSOLE [, char:T]) allows you to type in REAL or CHARACTER data, without the necessity of separating values by commas as would be required by x <- vector(3, 4, 5, ...) or x <- vector("weight", "height","age"). This feature is also useful in a batch file since it allows the lines of data immediately to follow the line containing the vecread() command.

Several keyword phrases may be used as arguments to vecread().

```
stop:"$"                Set stopping character to '$'
skip:"#"                Set skipping character to '#'; lines
                        starting with skipping character are
                        ignored
quiet:F                 Prints skipped lines, if any.
echo:T                  Echo all lines scanned, including skipped
                        lines
echo:F                  Do not echo lines scanned (except skipped
                        lines with quiet:F)
silent:T                Suppress all warning messages; incompatible
                        with echo:T or quiet:F.
```

For numerical data, instead of "$" or "#" as stopping or skipping characters you may use any punctuation character except '+', '-', ',', '?' or '.', that is, any of !"#$%&'()*/:;<=>@[\]^_`{|}~.

With character:T or bylines:T, any visible printing character may be a skipping character and any punctuation character except ',', including '+', '-', '?' or '.', may be a stopping character. With bylines:T the stopping character is recognized only at the start of a line.

If the skipping character is the same as the stopping character (for example, stop:"#",skip:"#"), scanning will be stopped only if the stopping character occurs after the first character in a non-skipped line (for example, by a line starting " #..." but not by "#...").

If FileName is CONSOLE and the vecread() command is in a batch file, data that is read is echoed to output unless echo:F is an argument. This also happens on a Macintosh even when not reading from a batch file.

Here is a macro that tests whether each "field" of a character scalar represents a valid number:

```
  isnumber <- macro("@tmp <-  paste(vecread(string:$1,char:T))
      !ismissing(vecread(string:@tmp,badval:?))")
```

isnumber("3.45") returns True, isnumber("3b45") returns vector(T,F,T) and isnumber("3.4 4.5 A") returns vector(T,T,F).

See also readcols(), matread(), macroread(), batch(), data files, files, console, vectors.

2.302 vector

Keywords: variables, combining variables, character variables, null variables

Usage: vector(x1,x2,...,xk [,KeyPhrases]) where x1, x2, ... all have the same type, REAL, LOGICAL, or CHARACTER, or are structures with components all of the same type KeyPhrases can be labels:lab and/or silent:T, where lab is a CHARACTER scalar or vector.

vector(x1, x2, ..., xk) combines scalars x1, x2, ... xk into a vector of length k. Thus you can enter a small set of data by, for example,

```
Cmd> x <- vector(3.5, 9.6, 2.5, 2.3, 7.7, 2.6, 6.3, 6.5, 6.6, 4.1)
```

The arguments x1, x2, ..., xk can be REAL, LOGICAL or CHARACTER and must all have the same type. Thus, for example, you can create a CHARACTER vector of labels by

```
Cmd> labels <- vector("Length","Width", "Weight")
```

vector() is identical to cat(). However, cat() is a deprecated function, that is, it will remain available for the immediate future, but at some time it may be disabled. Use vector() instead.

The arguments may also be vectors. In that case vector(x1, x2, ..., xk) combines all the arguments into a single vector. Thus, for example,

```
Cmd> vector(run(3), run(3,1))
```

is equivalent to

```
Cmd> vector(1,2,3,3,2,1).
```

More generally, any or all of the arguments may be matrices, arrays, as long as they all have the same type. In that case, vector(x1, x2, ..., xk) has the same effect as vector(vector(x1), vector(x2), ..., vector(xk)), combining all the elements of its arguments into one long vector. See the next paragraph for what vector(x) does when x is a matrix or array.

vector(x) creates a vector from a matrix or array x by "unravelling" it, with the first subscript changing fastest, the second changing next fastest, etc. Specifically, if the dimensions of x are n1, n2, ..., nk, vector(x) is a vector with length n1*n2*...*nk, with elements x[1,1,...,1], x[2,1,...,1], ..., x[n1,1,...,1], x[1,2,...,1], x[2,2,...,1], ..., x[n1,2,...,1], ..., x[1,3,...,1], ..., x[n1,n2,...,nk]. x may be REAL, LOGICAL, or CHARACTER.

If Str is a structure with n components, vector(Str) is a vector equivalent to vector(vector(Str[1]),...,vector(Str[n])), defined recursively if any component is a structure. All the data components must be of the same type, REAL, LOGICAL or CHARACTER. This should not be confused with strconcat() which combines structures into a larger structure.

Any argument of type NULL is ignored. Thus, vector(NULL, a) or vector(a, NULL) are equivalent to vector(a). If all arguments to vector() have type NULL, so does the result.

See also vectors, structures.

2.303 vectors

Keywords: variables, syntax
Usage: `Create a vector: x <- vector(x1,x2, x3, ...) Extract element(s) x[i], i an integer scalar or vector or LOGICAL vector`

A vector is an array with only one dimension, its length. The most usual way to create a vector is to use vector():

```
Cmd> x <- vector(1,3,2.5,6); y <- vector("Hi","Lois")

Cmd> z <- vector(T,T,F,F,T)

Cmd> list(x,y,z) # all are vectors, having only 1 dimension
x               REAL   4
y               CHAR   2
z               LOGIC  5
```

Practically always, a n by 1 matrix (column vector) or a n by 1 by 1 ... by 1 array is also treated as a vector. For example, if y is a n by p matrix, then factor(y[,3]) is legal, even though factor() expects a vector as argument, because y[,3] has only one column. Moreover, isvector(x) is True when x is a vector, a n by 1 matrix, or a n by 1 by 1 ... by 1 array.

See also vector(), isvector(), factor(), matrices.

2.304 vt

Keywords: plotting
Usage: `vt()`

vt() (no argument) puts your terminal in vt100 emulation mode if you are running MacAnova on Unix through a terminal emulator. In its standard form, vt() transmits the characters corresponding to the ASCII codes 27 and 50 (ESC, '2'). If these are not appropriate for your terminal, vt can be redefined to send different characters. For example, to change vt so that it is appropriate for Version 2.32 or later of the public domain Kermit program for IBM-compatibles, use

```
Cmd> vt <- macro("putascii(vector(27,91,63,51,56,108))")\
      #ESC,'[','?','3','8','l'
```

For version 2.6 of NSCA Telnet on a Macintosh, use

```
Cmd> vt <- macro("putascii(vector(27,12))")
```

See vtx if you are running MacAnova in an xterm window on a workstation.

If you need vt at all, it is often convenient to put ';vt();' on the

line after a plotting command, so that the next prompt will not overwrite your graph.

vt is implemented as a pre-defined macro (Unix versions only).

See also tek(), plot(), chplot(), lineplot(), boxplot(), showplot()

2.305 vtx

Keywords: plotting
Usage: vtx()

vtx() switches a Unix workstation xterm terminal emulator to vt100 mode from Tektronix 4014 mode. You don't normally need vtx since MacAnova recognizes when it is running in a xterm environment (the value of environmental variable $HOME is "xterm") and automatically switches back to vt100 mode after drawing a high resolution graph.

vtx is implemented as a pre-defined macro (Unix versions only).

See also tek, tekx, vt, graphs, unix.

2.306 while

Keywords: syntax, control
Usage: while(Logical)statement1;statement2;...;

while(Logical){statement1;statement2;....;} repeatedly executes the statements enclosed in '{' and '}' as long as Logical has value True. Logical should be a LOGICAL variable or expression, but not a constant expression. Unless the last statement in {...} is empty (';;' just before '}'), its value may be printed on every repetition.

The opening '{' must be on the same line as 'while' unless that line terminates with '\'.

To avoid "infinite" loops, a 'while' loop will automatically terminate after 1000 repetitions. This limit can be changed by option 'maxwhile'. After

```
  Cmd> setoptions(maxwhile:10000)
```

a 'while' loop will terminate after 10000 repetitions.

It is essential that one of the statements modify the variable(s) used in the LOGICAL expression, or that a break or breakall statement is used. Otherwise the loop will repeated until 1000 (or value reset by option 'maxwhile') repetitions are completed.

A 'while' statement does not have a value. Hence such constructs as yyy

```
<- while(n > 0){...} or zzz + while(n > 0){...} are illegal.

Examples:
  Cmd> @s <- 0;@n <- length(x);while(@n>0){\
        @s <- @s+x[@n];@n <- @n-1;;};@s
```

This would print the sum of all the elements in x.

```
  Cmd> ex <- macro("@x <- $1;@dims <- dim(@x);@x <- vector(@x)
                  @neg <- 1-2*(@x<0);@x <- abs(@x);@eps <- 1e-12
                  @s <- @t <- @k <- 1
                  while(max(@t) > @eps*max(@s)){
                    @t <- (@x/@k)*@t
                    @s <- @s+@t;@k <- @k+1
                  }
                  array(@s^@neg,@dims)")
```

will create a macro ex such that ex(x) computes exp(x) using a power series, when x is a vector, matrix, or array.

See also if, break, breakall.

2.307 write

Keywords: output, files

Usage: write(a, b, ...[,format:Fmt or nsig:m, header:F, labels:F, width:w, height:h, missing:missStr, name:setName] [, file:fileName [,new:T]]]), Fmt, missStr, fileName, setName CHARACTER scalars, m > 0, w >= 30, h >= 12 integers

write(a, b, ...) prints objects (variables, expressions, macros) a, b, ..., ordinarily retaining more decimal places than print(). By default, write() formats REAL items using the format identified by 'wformat' on getoptions() output. This normally specifies 9 significant digits in floating point form but may be changed by setoptions().

Except for using a different default format, write() is identical with print(), and recognizes the same keywords. It is provided as an easy way to print results with more significant digits than does print(), without having explicitly to specify a format. See print() for a full description of the various keywords and output.

See also setoptions(), fwrite(), fprint(), matprint(), matwrite(), macrowrite().

2.308 wtanova

Keywords: glm, anova
Usage: wtanova([Model] ,Wts [print:F or silent:T, coefs:F, pvals:T, fstats:T]), Model a CHARACTER scalar, Wts a REAL vector.

wtanova(Model,Wts) is equivalent to anova(Model,weights:Wts). See anova() for details.

2.309 wtmanova

Keywords: glm, anova
Usage: wtmanova([Model] ,Wts [print:F or silent:T, coefs:F, pvals:T, fstats:T , sssp:F or T]), Model a CHARACTER scalar, Wts a REAL vector.

wtmanova(Model,Wts) is equivalent to manova(Model, weights:Wts). See manova() for details.

2.310 wtregress

Keywords: glm, regression
Usage: wtregress([Model], Wts [, print:F or silent:T, pvals:T]) Model a CHARACTER scalar, Wts a REAL vector.

wtregress(Model,Wts) is equivalent to regress(Model,weights:Wts). See regress() for details.

2.311 wx

Keywords: general
Usage: Type help(wx) for information on the Motif and Windows versions of MacAnova.

Both the Windows and Unix/Motif versions of MacAnova make use of the wxWindows version 1.68 windowing interface (see topic copyright). There are minor differences in the look and feel of MacAnova on different platforms. Although the interface was patterned on the Macintosh version, the matching is incomplete.

You can have up to eight command/output windows and up to eight high resolution graphics windows. All windows have four menus -- File, Edit, Windows, and Help (no Edit menu on Motif graphics windows).

Commands are typed or pasted into the front-most command/output window

immediately after the prompt.

Editing the contents of the command/output window, including any command being entered after the prompt, is done using the mouse and the keyboard.

History of Previous Commands

Typed commands are automatically saved in an internal "history" list. You can move through this list, inserting previous commands after the prompt, using items Up History and Down History on the Edit menu. Pressing F7 or F8 is equivalent to selecting menu item Up History or Down History, respectively. Instead of F7 and F8 you can press the Ctrl and the up or down arrow key on the key pad at the right side of the keyboard. In Windows, but not Motif, you can use Ctrl and the ordinary up and down arrow keys.

By default, MacAnova saves the most recent 100 lines. To change this, say, to 50, type 'setoptions(history:50)'. See setoptions().

Moving Around the Window

Arrow keys move the cursor as you would expect.

Ctrl+A (Go To Prompt on the Windows menu) moves the cursor to the start of the current command line, just after the current prompt.

Ctrl+E (Go To End on the Windows menu) moves the cursor to the end of the current command line at the very end of window.

Ctrl+T (Scroll To Top on the Windows menu) scrolls to the Top of the current command/output window without moving the cursor.

Specifying File Names

When you use "" as the file name in any command requiring one (for example vecread("")), a dialog box lets you select the file. You can also use an explicit file or "path" name. In the latter case, if the name does not contain ':' or '/' (which would identify it as a "path" name), MacAnova will look for it first in the default Folder (see topic Files) and then in the Folders specified in pre- defined CHARACTER vector DATAPATHS. See adddatapath, customize.

The Windows version can read and write files with long names when running Windows 95 or Windows NT.

Help

Selecting item Help on the Help menu displays a dialog box containing basic information on using command help().

Interrupting

Ctrl+I (Interrupt on the File menu) may be used to interrupt an operation or output. Depending on the operation it may not be recognized immediately. The Tab key does not trigger an interrupt.

File Menu

Open (Ctrl+O) allows you to read a file into a new command/output window. This would usually be the contents of a command/output window saved on an earlier run, but can be any text file. You select the file using a dialog box. Note: This is not how you read data from a file; for that you use vecread(), matread() or readcols.

Save Window (Ctrl+S) saves the contents of the front-most command/ output window in a file. If the window hasn't previously been saved, you are asked to provide a name for the file in a dialog box.

Save Window As saves the window, always asking for a name in a dialog box.

Page Setup allows you some control over how a command/output window is printed.

Print Window (Ctrl+P) prints the contents of the window. There is currently no way to print part of the window as on a Macintosh.

Interrupt (Ctrl+I) interrupts any command that is running or stops output. See above.

Restore Workspace (Ctrl+R) is equivalent to restore(""), allowing you to restore a previously saved workspace.

Save Workspace (Ctrl+K) is equivalent to save("") if there has been no previous save or asciisave, or to save() otherwise. See save().

Save Workspace As is equivalent to save(""). See save().

Quit (Ctrl+Q) is equivalent to typing quit. You will be asked if you want to save the workspace and any open command output windows. If you change your mind about quitting, you can click on Cancel in the dialog box. If you want to quit unconditionally, without the chance to save stuff, type quit(F) at the prompt. See quitting.

When a graphics window is in front, the File menu contains only three items, Interrupt, Go On and Print Graph. Go On is used to restart execution after pause:T is used on a plotting command (see graphs) and Print Graph allows you to print a copy of the graphics window.

Edit Menu

The first four items are Undo, Cut, Copy and Paste.

Undo (Ctrl+Z) gives you a limited capability to undo changes to the window. After you have undone something, selecting Undo redoes what you just undid. Undo affects only the window and cannot change anything that has been computed.

Cut (Ctrl+X) and Copy (Ctrl+C) are operative only when you have selected some text with the mouse. Both copy the text to an internal "clipboard" from which it can later be retrieved. Cut also removes the selected text from the window. The text on the clipboard can be pasted into any

window with a Paste item on its Edit menu, if there is one. In particular, under Windows, the text on the clipboard can be pasted into a Microsoft Word or other word processor document.

Paste (Ctrl+V) inserts at the cursor any text that is on the clipboard, perhaps because you just Copied or Cut it from elsewhere in the window. The text may have been put on the clipboard in another program, allowing you to transfer text to MacAnova.

The clipbord is "connected" to special variable CLIPBOARD. See topic clipboard.

Copy to End (F5) copies to the end of the command line any text that has been selected with the mouse. This is an easy way to reuse previously typed commands. In Motif version, pressing Ctrl+/ also selects this item.

Execute (F6) is similar to Copy to End except it adds a Return after the copied text. This will usually cause the line to be executed. If nothing is selected, Execute simply adds a Return to the command line, usually executing what has been typed. This is helpful when you have edited something in the middle of the line and then want to execute it. Pressing Shift+Enter or Ctrl+Enter also selects this item; in the Motif version, so do Ctrl+/ or Enter on the "keypad".

Up History (F7) inserts the previously typed command after the prompt. Repeated selection of Up History successively inserts older and older commands. Ctrl+Keypad Up Arrow also selects this item; on Windows, so does Ctrl+Up Arrow.

Down History (F8) moves forward through previously saved commands, inserting them after the prompt. Ctrl+Keypad Down Arrow also selects this item; on Windows, so does Ctrl+Down Arrow.

By default, the Wx MacAnova versions save the most recent 100 lines. To change this, say, to 50, type 'setoptions(history:50)'.

When a graph window is in front, under Motif there is no Edit menu at all. Under Windows the only item is Copy. This allows you to copy the graph to the clipboard to paste into a Windows word processor or drawing program.

Windows Menu

Hide hides the front window without destroying its contents.

Close (Contol-W) closes the front window. If it is a command/output window you are asked if you want to save it.

Graphs pulls down a submenu allowing you to select any graph window in use, including any that are hidden. You can also use key combinations Ctrl+F1, Ctrl+F2, ..., Ctrl+F8 to select a graph window. In Motif, you can use Ctrl+1, ..., Ctrl+8 instead.

Windows pulls down a submenu allowing you to select any command/output window in use, including any that are hidden.

When a command/output window is in front, the following additional menu items are available.

New (Ctrl+N) creates a new command/output window and makes it the front window. If there are already 8 command/output windows New does nothing.

Scroll to Top (Ctrl+T) scrolls the command/output window to the beginning but does not move the cursor.

Go to End (Ctrl+E) moves the cursor after the last text in the command/output window.

Go to Prompt (Ctrl+A) moves the cursor immediately after the prompt.

In the Windows version, but not Motif, there is also a Fonts item, allowing you to change the font used in the command/output window. The default font is Courier New size 12.

In addition, there is a Help menu which doesn't do much except give a little information on how to use help().

Files produced by save() by the Windows version are restorable by both DOS versions and Linux. Files produced by the Motif version are restorable by other Unix versions on the same computer.

See topic unix for general information on Unix versions and topic dos-windows for general information about versions on DOS/Windows systems.

2.312 xrows

Keywords: glm
Usage: xrows(variates [,factors]), variates and factors REAL matrices or vectors or NULL.

xrows(Variates, Factors) computes rows of an X-variable (design) matrix corresponding to variate values in Variates and factor levels in Factors, using the information saved by the preceding GLM command.

Variates should either be a REAL vector with length(Variates) = nv, where nv = number of variates in the model, or a REAL matrix with ncols(Variates) = nv. If nv = 0, Variates should be NULL.

Factors should either be a REAL vector with length(Factors) = nf, where nf = number of factors in the model, or a REAL matrix with ncols(Factors) = nf. All the elements of Factors should be positive integersnot exceeding the maximim level for each factor. If nf = 0, Factors should be NULL or should be omitted entirely.

Let nrowv be 1 if Variates is a vector and nrows(Variates) otherwise, and let nrowf be 1 if Factors is a vector and nrows(Factors) otherwise. Then if nrowv != nrowf, you must have either nrowv = 1 or nrowf = 1 and the value of Variates or Factors is used for every row of the output.

The result is a REAL matrix with max(nrowv, nrowf) rows. Each row consists of the values of the X-variables (design matrix) corresponding to that row of Variates and Factors.

Examples:
After anova("y=x1+x2+a*b"), xrows(hconcat(x1,x2), hconcat(a,b)) is equivalent to xvariables().

After regress("y=x1+x2"); xrows(x0) %*% COEF is equivalent to regpred(x0,seest:F,sepred:F) where x0 is a matrix of values for x1 and x2.

See also xvariables(), modelinfo(), regpred(), glmpred(), glm

2.313 xvariables

Keywords: glm
Usage: xvariables(Model [, missing:val]), Model a CHARACTER scalar

xvariables(Model) returns the full design matrix (matrix of X-variables) associated with Model, including a column for the intercept (constant term), if any. Model must be a scalar CHARACTER variable or quoted string.

See topic 'models' for information on specifying Model.

xvariables() (without a model) does the same except it uses the model specified in STRMODEL which is usually the model used by the most recent GLM (generalized linear or linear model) command such as regress(), anova(), or poisson().

xvariables(Model, missing:val) and xvariables(missing:val) where val is a REAL scalar provides a value for cases with missing values.

Each variate in the model will appear as a column of the output. Factors and interactions are translated to one or more "dummy" variables with values 1, 0, or -1, or to products of dummy variables or of dummy variables and variates. Any row in which the depending variable or any factor or variable or the dependent variable is MISSING is set entirely to 0 or, if 'missing:val' is an argument, to val. Thus, for example, xvariables(Model, missing:?) or xvariables(missing:?) results in rows with missing data being set to MISSING rather than to 0.

Note: If no model is specified and the previous command was regress(),

xvariables() does not compute dummy variables associated with any factors in the model, but treats them as if they were variates. Except in this special case, the behavior of xvariables() differs sharply from the behavior of modelvars() which retrieves factors and variates unchanged.

When there are no factors in the model, a regression of the dependent variable on the columns of xvariables()[,-1], that is, the result of xvariables() excluding the constant column, should yield the same coefficients as does coefs() after regress(Model) or anova(Model).

When there are factors in the model, the regression coefficients in such a regression will differ from those produced by coefs() after anova(Model), although the fitted values and residuals will be the same. For example, following anova("y=a.x"), where a is a factor and x is a variate, the coefficients from coefs("a.x") will be the slopes in a model fitting separate lines for each level of a, but with a common intercept. This is not the parametrization implicit in a regression of y on the columns of xvariables[,-1].

See also varnames(), modelvars(), modelinfo(), models.

2.314 yates

Keywords: glm, anova
Usage: yates(x), x a REAL vector

yates(x) performs Yates' algorithm for the effects in a 2-series factorial experiment. The argument x should be a REAL vector (univariate case) or matrix (multivariate case) containing the 2^k observations in standard order, that is, the levels of the first factor changing most rapidly.

If x is a vector, the value is a vector of the (2^k)-1 effects in standard order and divisor 2^(k-1). For example, for a 2^3 experiment with data vector(x111,x211,x121,x221,x112,x212,x122,x222) the result is vector(A,B,AB,C,AC, BC,ABC), where A = (-x111+x211-x121+x221-x112+ x212-x122+x222)/4, B = (-x111-x211+x121+x221-x112-x212+x122+x222)/4, and so on. The mean, (x111+x211+x121+x221+x112+x212+x122+x222)/8, is not included.

If x is a matrix with m columns, the value is a (2^k) - 1 by m matrix, each column of which is the result of applying Yates's algorithm to the corresponding column of x.

2.315 yhat

Keywords: glm, regression, anova
Usage: yhat() or yhat(Model [,T])

yhat(), with no argument, computes a REAL matrix of various quantities useful in making predictions from a regression or analysis of variance model. It uses side effect variables RESIDUALS, HII, etc. produced by the most recent GLM (generalized linear or linear model) command such as regress() or anova(). If weights were supplied, it also uses the result of modelinfo(weights:T).

NOTE: If the most recent GLM was not linear, that is, not regress(), anova(), manova() or their weighted variants, only the first two columns are computed by yhat are meaningful.

It is an error if any of the needed side effect variables do not exist.

yhat(Model) first executes manova(Model, silent:T) to compute the side effect variables and then computes its usual output. Model should be a CHARACTER variable or string specifying a linear model. If there are factors in the model, they will be so interpreted. If you want them treated as variates, use yhat(Model,T).

Each row of the result corresponds to a case. If the dependent variable Y is univariate (has one column), there are 5 columns in the result, as follows:

Col. 1	Y = observed response
Col. 2	Yhat = predicted or fitted value computed using all data
Col. 3	Predictive residuals = Y - (Yhat computed excluding the case)
Col. 4	SE[Yhat] = estimated standard error of Yhat as estimate of E[Y|x]
Col. 5	SE[pred] = estimated s.e. of prediction error

If the all the independent variables in the model "y=x1+x2+x3+...+xk" are variates and not factors, columns 2, 4, and 5 of the output correspond to components 'estimate', 'SEest', and 'SEpred" in the output of regpred(hconcat(x1,x2,...)) following regress("y=x1+x2+...+xk"). See regpred().

If Y is multivariate of dimension p, there are 5*p columns in groups of p -- the p columns of Y, the p columns of YHat, and so on.

If a case has missing values, most entries will be MISSING and there are no useful numbers.

Example:

```
Cmd> yhat("y=x1+x2+x3+x4")
```

The output of yhat is modelled on output printed by the 'yhat' command in program Multreg.

yhat is implemented as a pre-defined macro.

See also regpred(), glm, resid.

2.316 yulewalker

Keywords: time series
Usage: yulewalker(vec [, inverse:T]), vec a REAL vector.

yulewalker(Rho) computes a REAL vector of autoregressive (AR) coefficients phi[1], phi[2], ... phi[p] of a p-th order AR time series whose autocorrelations r[1], r[2], ... r[p], are in REAL vector Rho, where p = nrows(Rho). The values of phi satisfy the Yule-Walker equations rho[j] = sum(phi[k]*rho[j-k],j=1,...,p), k = 1,...,p, with rho[0] = 1, rho[-j] = rho[j].

If Rho is a matrix, yulewalker(Rho) computes AR coefficients from each column separately, that is, yulewalker(Rho) = hconcat(yulewalker(Rho[,1]), yulewalker(Rho[,2]), ...). If Rho is a generalized matrix (at most two dimensions >= 1), yulewalker(Rho) = yulewalker(matrix(Rho)) (see matrices, matrix()).

If any column of Rho is not a valid autocorrelation function, that is, if the implied Toeplitz correlation matrix is not positive definite, yulewalker() prints a warning message, sets the element in the result where the violation occurred to the most extreme value possible and any subsequent elements to zero. For instance, yulewalker(vector(-.3, -.9,.5)) returns the result vector(-.6, -1, 0).

A typical usage is yulewalker(rhohat) where rhohat is the vector of the first p estimated autocorrelations from a time series. The result is a vector containing the coefficients of the autoregressive series whose first p autocorrelations are the same as rhohat. Thus it provides a method-of-moments estimating method. If the time series is in fact an normal stationary AR series of order length(rhohat), these estimates are asymptotically equivalent to maximum likelihood estimates.

yulewalker(Phi,inverse:T) computes auto correlations corresponding to autoregressive coefficients in the columns of REAL vector or matrix Phi.

Effectively yulewalker(Phi,inverse:T) is the inverse function to yulewalker() in that yulewalker(yulewalker(Rho),inverse:T) should the same as Rho, except for rounding error. One important usage is

```
yulewalker(padto(Phi,n), inverse:T)
```

where n >= nrows(phi). This computes the first n autocorrelations of the autoregressive series with autoregression coefficients in Phi.

See also padto(), partacf(), toeplitz().

Chapter 3

Design Macros Help File

This Chapter contains help for the set of design and analysis of experiments macros that are distributed with MacAnova in the file design.mac. The material here is a reformatting of file design.hlp.

3.1 aliases2

Keywords: Aliasing, design, factorial
Usage: `aliases2(basis[,effect:vec][,length:j]), REAL matrix basis of alias generators, REAL vector vec of 0's and 1's, positive integer j`

```
aliases2(basis,effect:vec,length:j) finds aliases in a 2^(k-p)
fractional factorial and returns a CHARACTER vector of these aliases as
its value.  k must be no larger than 25, and factors are labeled A, B,
... Z (skipping I).

The p x k matrix basis contains the generators for the aliasing, one row
for each generator and one column for each factor in the design.  The
elements in basis are 0, -1, or 1.  A nonzero entry indicates that a
factor is present in the generator for that row. The sign of a generator
is the product of the signs of the nonzero elements of the generator.
For example, 1 0 1 0 0 -1 means -ACF is a generator (alias of I).

By default, all aliases of I are returned.  If the keyword phrase
effect:vec is used, then the aliases of vec are found, where vec is a
vector of length k of 0s and 1s indicating an effect.  (The default
value for vec is all 0s, ie I.)  If length:j is used, only aliases of
length j are found.

Examples:
   Cmd> print(format:"2.0f",b) # Matrix b is 2x5, so 2^(5-2) design
   b:
   (1,1)  1  1  1  0  0       [ABC is a generator]
   (2,1)  0  0  1  1 -1       [-CDE is a generator]

   Cmd> aliases2(b)   # aliases of I
```

```
(1) "I"
(2) "ABC"
(3) "-CDE"
(4) "-ABDE"

Cmd> aliases2(b,length:3)  # length 3 aliases of I
(1) "I"
(2) "ABC"
(3) "-CDE"

Cmd> aliases2(b,effect:vector(1,1,0,0,0)) # aliases of AB
(1) "AB"
(2) "C"
(3) "-ABCDE"
(4) "-DE"
```

3.2 aliases3

Keywords: Aliasing, design, factorial

Usage: aliases3(basis[,effect:vec][,length:j]), REAL matrix basis of alias generators, REAL vector vec of 0's and 1's, positive integer j

aliases3(basis,effect:vec,length:j) finds aliases in a 3^(k-p) fractional factorial and returns a CHARACTER vector of these aliases as its value. k must be no larger than 25, and factors are labeled A, B, ... Z (skipping I).

The p x k matrix basis contains the generators for the aliasing, one row for each generator and one column for each factor in the design. The elements in basis are 0, 1, or 2. 1 0 2 0 0 1 means AC^2F is a generator (alias of I).

By default, all aliases of I are returned. If the keyword phrase effect:vec is used, then the aliases of vec are found, where vec is a vector of length k of 0s, 1s, and 2s indicating an effect. (The default value for vec is all 0s, ie I.) If length:j is used, only aliases of length j are found.

Examples:

```
Cmd> print(c,format:"2.0f") # Matrix c is 2x4, so 3^(4-2)
c:
(1,1)  1  2  0  2        [A B^2 D^2 is a generator]
(2,1)  0  1  2  2        [B C^2 D^2 is a generator]

Cmd> aliases3(c)  # all aliases of I
(1) "I"
(2) "A^1 B^2 D^2 "
(3) "A^1 B^2 D^2 "
(4) "B^1 C^2 D^2 "
(5) "A^1 C^2 D^1 "
```

```
(6) "A^1 B^1 C^1 "
(7) "B^1 C^2 D^2 "
(8) "A^1 B^1 C^1 "
(9) "A^1 C^2 D^1 "

Cmd> aliases3(c,effect:vector(1,1,0,0)) # aliases of A^1B^1
(1) "A^1 B^1 "
(2) "A^1 D^1 "
(3) "B^1 D^2 "
(4) "A^1 B^2 C^2 D^2 "
(5) "A^1 B^2 C^1 D^2 "
(6) "C^1 "
(7) "A^1 C^1 D^1 "
(8) "A^1 B^1 C^2 "
(9) "B^1 C^1 D^2 "
```

3.3 allaliases2

Keywords: Aliasing, design, factorial
Usage: `allaliases2(basis), REAL matrix basis of alias generators`

allaliases2(basis) finds the full set of aliases in a 2^(k-p) fractional factorial and returns a CHARACTER vector of these aliases as its value. k must be no larger than 25, and factors are labeled A, B, ... Z (skipping I).

The p x k matrix basis contains the generators for the aliasing, one row for each generator and one column for each factor in the design. The elements in basis are 0, -1, or 1. A nonzero entry indicates that a factor is present in the generator for that row. The sign of a generator is the product of the signs of the nonzero elements of the generator. For example, 1 0 1 0 0 -1 means -ACF is a generator (alias of I).

Examples:

```
Cmd> print(b,format:"2.0f")  # Matrix b is 2x5, so 2^(5-2) design
b:
(1,1)  1  1  1  0  0     [ABC is a generator]
(2,1)  0  0  1  1 -1     [-CDE is a generator]

Cmd> allaliases2(b)  # alias table
(1) "I = ABC = -CDE = -ABDE"
(2) "A = BC = -ACDE = -BDE"
(3) "B = AC = -BCDE = -ADE"
(4) "AB = C = -ABCDE = -DE"
(5) "D = ABCD = -CE = -ABE"
(6) "AD = BCD = -ACE = -BE"
(7) "BD = ACD = -BCE = -AE"
(8) "ABD = CD = -ABCE = -E"
```

3.4 boxcoxvec

Keywords: ANOVA, analysis
Usage: boxcoxvec(rhsmodel,y[,powers:pow]), CHARACTER scalar rhsmodel, REAL vectors y and pow.

boxcoxvec(rhsmodel,y,powers:pow) computes the error SS for y transformed to the boxcox powers given in pow and modeled using the explanatory variables in rhsmodel. The returned value is a structure with named components power and SS giving the boxcox powers and the error SS.

rhsmodel is the right hand side of the anova model (the part following '=') as a character scalar, for example "x + a + b". If powers:pow is omitted, the default powers are run(-1,2,.25)

Examples:

```
Cmd> y <- rnorm(20);ey <- exp(y)
NOTE: random number seeds set to 59622139 and 172924584

Cmd> a <- factor(rep(run(5),4))

Cmd> boxcoxvec("a",ey)  # default powers
component: power
 (1)          -1        -0.75         -0.5        -0.25            0
 (6)        0.25          0.5         0.75           1         1.25
(11)         1.5         1.75            2
component: SS
 (1)       179.93       72.889       34.971       21.487       17.979
 (6)       20.624       31.191       58.929       131.81       332.94
(11)       915.71         2673       8140.8

Cmd> boxcoxvec("a",ey,powers:run(-.25,.25,1/16))#explore powers ~= 0
component: power
(1)         -0.25      -0.1875       -0.125      -0.0625            0
(6)       0.0625        0.125       0.1875         0.25
component: SS
(1)       21.487       19.938       18.875       18.236       17.979
(6)       18.083       18.544        19.38       20.624
```

3.5 choosegen2

Keywords: Aliasing, design, factorial
Usage: choosegen2(k,p, all:T or tries:m [,res:r]), positive integers k, p, m, and r.

choosegen2(k,p,all:T,tries:m,res:r) finds a set of generators for a 2^(k-p) fractional factorial design. Either all:T or tries:m must be specified. If all:T is specified, all potential generators are searched. If tries:m is specified, m random sets of generators are searched. If res:r is not specified, a set of generators giving the

greatest resolution among those searched is returned. If res:r is specified, the first set of generators found giving resolution r is returned. If no such set is found, the generators giving the best resolution found are printed.

The return value is a structure with named components resolution and generators.

Examples:

```
Cmd> choosegen2(5,2,all:T)  # find best resolution
component: resolution
(1)            3
component: generators
(1) "ABCD"
(2) "BCE"

Cmd> choosegen2(5,2,all:T,res:4)  # try to find 4 (we won't)
component: resolution
(1)            3
component: generators
(1) "ABCD"
(2) "BCE"

Cmd> # look for 2^(10-4) resol. 4, just make 1000 tries, since\
there are lots of combinations to explore

Cmd> choosegen2(10,4,tries:1000,res:4)# look for 2^(10-4) resol. 4
component: resolution
(1)            4
component: generators
(1) "ABCFG"
(2) "ABCDEH"
(3) "ACEJ"
(4) "ACDK"
```

3.6 confound2

Keywords: Confounding, design, factorial
Usage: `confound2(basis), REAL matrix basis containing confounding generators`

confound2(basis) confounds a two series factorial into blocks based on the generators given in the matrix basis. Results are returned in a structure with component names block1, block2, etc. Each component has a character vector of factor/level combinations for that block.

The p x k matrix basis contains the generators for the confounding, one row for each generator and one column for each factor in the design. The elements in basis are 0 or 1. A nonzero entry indicates that a factor is present in the generator for that row.

```
Examples:
   Cmd> # Matrix d is 2x4 (2 generators so 4 blocks of size 4)

   Cmd> print(d,format:"2.0f")
   d:
   (1,1)  1  1  0  0     [AB is a generator]
   (2,1)  0  1  1  1     [BCD is a generator]

   Cmd> confound2(d)
   component: block1
   (1) "(1)"
   (2) "abc"
   (3) "abd"
   (4) "cd"
   component: block2
   (1) "a"
   (2) "bc"
   (3) "bd"
   (4) "acd"
   component: block3
   (1) "ab"
   (2) "c"
   (3) "d"
   (4) "abcd"
   component: block4
   (1) "b"
   (2) "ac"
   (3) "ad"
   (4) "bcd"
```

3.7 confound3

Keywords: Confounding, design, factorial
Usage: `confound3(basis), REAL matrix basis containing confounding generators`

`confound3(basis)` confounds a three series factorial into blocks based on the generators given in the matrix basis. Results are returned in a structure with component names block1, block2, etc. Each component has a character vector of factor/level combinations for that block.

The p x k matrix basis contains the generators for the confounding, one row for each generator and one column for each factor in the design. The elements in basis are 0, 1, or 2, indicating the exponent of each factor in the generator.

```
Example:
   Cmd> print(e,format:"2.0f")# Matrix e is 1x2 (one generator in a 3^2)
   e:
   (1,1)  1  2        [A^1 B^2 is a generator]
```

```
Cmd> confound3(e)
component: block1
(1) "00"
(2) "11"
(3) "22"
component: block2
(1) "10"
(2) "21"
(3) "02"
component: block3
(1) "20"
(2) "01"
(3) "12"
```

3.8 ems

Keywords: ANOVA, analysis, factorial
Usage: `ems(Model,randomvars [,marg:T] [,restrict:F] [,nonhier:T] [,keep:T] [,print:T])`

ems(Model,Randomvars) computes the expected mean squares for the terms in the ANOVA for the model given in CHARACTER scalar Model. Randomvars is a CHARACTER vector specifying the names of factors in the model which are random. Randomvars can also be REAL with integer elements specifying the index of a factor in the model. If there are no random factors, Randomvars should be NULL.

In this default use, ems() computes sequential (Type I) sums of squares for the the restricted (mixed effects add to zero across fixed factors) model, prints these expected means squares for each term, and returns no value. Contributions from random terms are shown as multiples of the variance component (for example, 16V(a.b)); contributions from fixed terms are shown as a multiple of a quadratic function for the term, for example, 32Q(c). In a balanced design, the Q() function is the sum of the squared coefficients divided by degrees of freedom, for example, sum(c^2)/(K-1). In an unbalanced situation, Q(c) is a more complicated quantity defined using matrix algebra.

ems() works only for factors -- no variates are allowed in the model.

ems() works for both balanced and unbalanced data.

ems() assumes that if a factor first appears in an interaction, then that factor is nested in the other terms of the interaction. For example, if the first appearance of factor c is in the term a.b.c, then c is assumed nested in the a.b combinations. This nesting is assumed in the remainder of the model. That is, continuing the example, if there is a later term c.d, it will be interpreted as a.b.c.d even though a.b.c.d is not specifically in the model.

When a term contains the first appearance in the model of more than one

factor, ems assumes that the new factors are merged to make a single factor, whose number of levels is the product of the numbers of levels in the factors being merged. For example, if the first appearance of factors b and c with 5 and 3 levels, respectively is in the term a.b.c, then b and c together are considered a single factor with 15 levels. This grouping is assumed in the remainder of the model. That is, continuing the example, if there is a later term c.d, it will be interpreted as b.c.d even though b.c.d is not specifically in the model. This grouped factor is interpreted as random if any of the factors in the group is random.

ems() uses the "synthesis" method of Hartley, as explained in 10.5.2 of R. R. Hocking (1985), The Analysis of Linear Models, Brooks/Cole, Belmont, CA.

The examples below are based on balanced two factor and three factor models with a total of 64 responses. All factors have 2 levels so two factor and three factor models have 16 and 8 replications, respectively. In some examples, one of the responses is set to MISSING to destroy balance.

A fully nested model, with c fixed and both d and e random.

```
Cmd> ems("y=c/d/e",vector("d","e"))
EMS(CONSTANT) = V(ERROR1) + 8V(c.d.e) + 16V(c.d) + 64Q(CONSTANT)
EMS(c) = V(ERROR1) + 8V(c.d.e) + 16V(c.d) + 32Q(c)
EMS(c.d) = V(ERROR1) + 8V(c.d.e) + 16V(c.d)
EMS(c.d.e) = V(ERROR1) + 8V(c.d.e)
EMS(ERROR1) = V(ERROR1)
```

A 3 factor crossed model, with c and d fixed, e random.

```
Cmd> ems("y=c*d*e",3) # e is factor 3
EMS(CONSTANT) = V(ERROR1) + 32V(e) + 64Q(CONSTANT)
EMS(c) = V(ERROR1) + 16V(c.e) + 32Q(c)
EMS(d) = V(ERROR1) + 16V(d.e) + 32Q(d)
EMS(c.d) = V(ERROR1) + 8V(c.d.e) + 16Q(c.d)
EMS(e) = V(ERROR1) + 32V(e)
EMS(c.e) = V(ERROR1) + 16V(c.e)
EMS(d.e) = V(ERROR1) + 16V(d.e)
EMS(c.d.e) = V(ERROR1) + 8V(c.d.e)
EMS(ERROR1) = V(ERROR1)
```

A 2 factor crossed model with unbalanced data, c fixed and d random

```
Cmd> y1 <- y[1]; y[1] <- ? # make data unbalanced

Cmd> ems("y=c*d",2) # d is factor 2
EMS(CONSTANT) = V(ERROR1) + 0.0080645V(c.d) + 31.508V(d) +
  0.0079365Q(c) + 63Q(CONSTANT)
EMS(c) = V(ERROR1) + 15.746V(c.d) + 0.0081925V(d) + 31.492Q(c)
EMS(d) = V(ERROR1) + 0.0042316V(c.d) + 31.484V(d)
EMS(c.d) = V(ERROR1) + 15.742V(c.d)
EMS(ERROR1) = V(ERROR1)
```

Use of Keywords to change the action of ems().

ems(Model,Randomvars,keep:T) suppresses printed output but returns a

structure (described below) containing the results. If you want the printed output too, use keep:T,print:T.

ems(Model,Randomvars,marg:T) computes expected mean squares based on adjusted (Type III) sums of squares.

ems(Model,Randomvars,restrict:F) computes expected mean squares assuming no marginal restrictions on the random effects in the model.

ems(Model,Randomvars,nonhier:T) computes expected mean squares for an analysis of variance that does not enforce the usual MacAnova hierarchy assumptions. That is, for example, model "y=a+b+c+a.b.c" does not imply that the two-way interaction degrees of freedom are part of the "a.b.c" term. You cannot use anova() to compute such an analysis although it can be done (if you know how) using swp().

These keywords can be used together. For example, ems(Model,Randomvars, marg:T,restrict:F) provides answers equivalent to the EMS in SAS PROC GLM.

More examples, still with y[1] MISSING:

```
  Cmd> ems("y=c*d",2,marg:T) # crossed with d random
  EMS(CONSTANT) = V(ERROR1) + 31.475V(d) + 62.951Q(CONSTANT)
  EMS(c) = V(ERROR1) + 15.742V(c.d) + 31.475Q(c)
  EMS(d) = V(ERROR1) + 31.475V(d)
  EMS(c.d) = V(ERROR1) + 15.742V(c.d)
  EMS(ERROR1) = V(ERROR1)

  Cmd> ems("y=c*d",2,restrict:F) # crossed with d random
  EMS(CONSTANT) = V(ERROR1) + 15.762V(c.d) + 31.508V(d) + 0.0079365Q(c)
     + 63Q(CONSTANT)
  EMS(c) = V(ERROR1) + 15.754V(c.d) + 0.0081925V(d) + 31.492Q(c)
  EMS(d) = V(ERROR1) + 15.746V(c.d) + 31.484V(d)
  EMS(c.d) = V(ERROR1) + 15.738V(c.d)
  EMS(ERROR1) = V(ERROR1)

  Cmd> ems("y=c*d",2,marg:T,restrict:F) # same as SAS PROC GLM
  EMS(CONSTANT) = V(ERROR1) + 15.738V(c.d) + 31.475V(d) + 62.951Q(CONSTANT)
  EMS(c) = V(ERROR1) + 15.738V(c.d) + 31.475Q(c)
  EMS(d) = V(ERROR1) + 15.738V(c.d) + 31.475V(d)
  EMS(c.d) = V(ERROR1) + 15.738V(c.d)
  EMS(ERROR1) = V(ERROR1)

  Cmd> y[1] <- y1 # restore value for y[1] to regain balance

  Cmd> ems("y=c*d+e+c.d.e",3)
  EMS(CONSTANT) = V(ERROR1) + 32V(e) + 64Q(CONSTANT)
  EMS(c) = V(ERROR1) + 8V(c.d.e) + 32Q(c)
  EMS(d) = V(ERROR1) + 8V(c.d.e) + 32Q(d)
  EMS(c.d) = V(ERROR1) + 8V(c.d.e) + 16Q(c.d)
  EMS(e) = V(ERROR1) + 32V(e)
  EMS(c.d.e) = V(ERROR1) + 8V(c.d.e)
  EMS(ERROR1) = V(ERROR1)
```

```
Cmd> ems("y=c*d+e+c.d.e",3,nonhier:T)
EMS(CONSTANT) = V(ERROR1) + 32V(e) + 64Q(CONSTANT)
EMS(c) = V(ERROR1) + 32Q(c)
EMS(d) = V(ERROR1) + 32Q(d)
EMS(c.d) = V(ERROR1) + 8V(c.d.e) + 16Q(c.d)
EMS(e) = V(ERROR1) + 32V(e)
EMS(c.d.e) = V(ERROR1) + 8V(c.d.e)
EMS(ERROR1) = V(ERROR1)
```

Note that 'nonhier:T' makes the c.d.e term disappear from the EMS for the fixed terms; compare with the default hierarchical model that precedes it.

Structure returned with keep:T

When 'keep:T' is an argument, the structure returned has components 'df', 'ss', 'termnames', 'coefs', and 'rterms'.

Component	Description
df	REAL Vector of degrees of freedom for all terms in model
ss	REAL Vector of sums of squares for all terms in model
termnames	CHARACTER vector of labels for each term
coefs	REAL matrix with coefs[i,j] the coefficient for term j in the EMS of term i
rterms	LOGICAL vector with T indicating that a term is random.

Components ss and df are just those computed from a MacAnova anova() command (possibly with marg:T as needed), and may not be in conformance with the model as used by ems for the following reasons:

1. anova() computes only hierarchical models, while you may specify nonhierarchical models in ems by using nonhier:T.
2. ems enforces nesting and grouping. If b first appears in a.b then b is nested in a and any appearance of b in a later term implies the presence of a. anova() does no such enforcing. For example, in "y=a+a.b+c+b.c", b.c would be interpreted by ems as a.b.c while anova() would not include a.b.c in the model. If b and c first appear together, then "y=b.c+d+c.d" is interpreted in ems as "y=b.c+d+b.c.d".

More examples, with 2^2 design in 16 replicates:

```
Cmd> ems("y=c*d",2)
EMS(CONSTANT) = V(ERROR1) + 32V(d) + 64Q(CONSTANT)
EMS(c) = V(ERROR1) + 16V(c.d) + 32Q(c)
EMS(d) = V(ERROR1) + 32V(d)
EMS(c.d) = V(ERROR1) + 16V(c.d)
EMS(ERROR1) = V(ERROR1)

Cmd> ems("y=c*d",2,keep:T)
component: df
(1)          1          1          1          1         60
component: ss
(1)    0.76155   0.036871    0.31116      1.623     56.318
component: termnames
(1) "CONSTANT"
(2) "c"
```

```
(3) "d"
(4) "c.d"
(5) "ERROR1"
component: coefs
(1,1)        64          0         32          0          1
(2,1)         0         32          0         16          1
(3,1)         0          0         32          0          1
(4,1)         0          0          0         16          1
(5,1)         0          0          0          0          1
component: rterms
(1) F       F       T       T
```

3.9 ffdesign2

Keywords: Design, Aliasing, factorial
Usage: `ffdesign2(basis), REAL matrix basis containing confounding generators`

ffdesign2(basis) finds the set of factor/level combinations used in the 2^(k-p) fractional factorial corresponding to the given generators. The result is a CHARACTER vector giving the factor/level combinations.

The p x k matrix basis contains the generators for the aliasing, one row for each generator and one column for each factor in the design. The elements in basis are 0, -1, or 1. A nonzero entry indicates that a factor is present in the generator for that row. The sign of a generator is the product of the signs of the nonzero elements of the generator. For example, 1 0 1 0 0 -1 means -ACF is a generator (alias of I).

Examples:

```
Cmd> print(b, format:"2.0f") # Matrix b is 2x5, so 2^(5-2) design
b:
(1,1)  1  1  1  0  0      [ABC is a generator]
(2,1)  0  0  1  1 -1      [-CDE is a generator]

Cmd> ffdesign2(b)
(1) "ce"
(2) "a"
(3) "b"
(4) "abce"
(5) "dc"
(6) "ade"
(7) "bde"
(8) "abdc"
```

3.10 mixed

Keywords: anova, analysis, random effects, factorial

Usage: mixed(Model,randomvars[,marg:T][,restrict:F][,nonhier:T] [,useneg:T][,keepmixed:T]) mixed(emsOutput[,useneg:T][,keepmixed:T])

mixed computes and prints an "ANOVA" table appropriate for the model and random factors given in Model and randomvars. These arguments (as well as marg, restrict, and nonhier) are exactly the same as for ems. Alternatively, mixed will accept the output of ems as argument.

The anova table has one row for each term in the model and seven columns. The columns give the term label, degrees of freedom and mean square for an appropriate numerator for testing the term, the degrees of freedom and mean square for an appropriate denominator for the term, and the F ratio and p-value. Numerators and denominators are linear combinations of mean squares the expectations of which differ only by the parameter of interest. If a numerator or denominator is not a simple mean square, approximate degrees of freedom are found using the Satterthwaite approximation.

By default, only linear combinations of mean squares with positive coefficients are used. This means that the numerator for a term may be the sum of the mean square for the term and one or more mean squares from other terms. If the keyword useneg:T is used, then the numerator for a term will be the mean square for that term, and denominators may contain differences as well as sums of mean squares.

By default, mixed prints the table and returns no value. If keepmixed:T is used, the table will be returned as a labeled matrix and not printed.

Example. Three populations, 4 males nested in each population, 2 females nested in each male, six offspring from each mating, with these randomly assigned to three environments. Male and female are random. First the simple anova.

```
Cmd> anova("y=(pop+m.pop+f.pop+m.f.pop)*env")
Model used is y=(pop+m.pop+f.pop+m.f.pop)*env
                DF           SS           MS
CONSTANT         1       5.4299       5.4299
pop              2       2091.4       1045.7
pop.m            9        112.5         12.5
pop.f            9       370.02       41.113
pop.m.f         27       56.774       2.1027
env              2       206.15       103.08
pop.env          4      0.16527     0.041316
pop.m.env       18       3.4185      0.18992
pop.f.env       18       8.2354      0.45752
pop.m.f.env     54       17.117      0.31698
ERROR1         144       30.448      0.21144
```

Now compute the expected mean squares, and keep the ems output.

```
Cmd> emsstuff<-ems("y=(pop+m.pop+f.pop+m.f.pop)*env",vector("m","f"),
   keep:T,print:T)
EMS(CONSTANT) = V(ERROR1) + 6V(pop.m.f) + 24V(pop.f) + 24V(pop.m) +
   288Q(CONSTANT)
EMS(pop) = V(ERROR1) + 6V(pop.m.f) + 24V(pop.f) + 24V(pop.m) + 96Q(pop)
EMS(pop.m) = V(ERROR1) + 6V(pop.m.f) + 24V(pop.m)
EMS(pop.f) = V(ERROR1) + 6V(pop.m.f) + 24V(pop.f)
EMS(pop.m.f) = V(ERROR1) + 6V(pop.m.f)
EMS(env) = V(ERROR1) + 2V(pop.m.f.env) + 8V(pop.f.env) +
   8V(pop.m.env) + 96Q(env)
EMS(pop.env) = V(ERROR1) + 2V(pop.m.f.env) + 8V(pop.f.env) +
   8V(pop.m.env) + 32Q(pop.env)
EMS(pop.m.env) = V(ERROR1) + 2V(pop.m.f.env) + 8V(pop.m.env)
EMS(pop.f.env) = V(ERROR1) + 2V(pop.m.f.env) + 8V(pop.f.env)
EMS(pop.m.f.env) = V(ERROR1) + 2V(pop.m.f.env)
EMS(ERROR1) = V(ERROR1)
```

Now use mixed.

```
Cmd> mixed(emsstuff)
                DF        MS  Error DF  Error MS         F    P value
CONSTANT     1.914     7.533     14.01     53.61    0.1405     0.8617
pop          2.008      1048     14.01     53.61     19.54  8.745e-05
pop.m            9      12.5        27     2.103     5.945  0.0001412
pop.f            9     41.11        27     2.103     19.55  1.242e-09
pop.m.f         27     2.103       144    0.2114     9.945          0
env          2.012     103.4     30.75    0.6474     159.7          0
pop.env      56.12    0.3583     30.75    0.6474    0.5534     0.9729
pop.m.env       18    0.1899        54     0.317    0.5991     0.8844
pop.f.env       18    0.4575        54     0.317     1.443     0.1496
pop.m.f.env     54     0.317       144    0.2114     1.499    0.03044
ERROR1         144    0.2114         0         0   MISSING    MISSING
```

The test for environment should be
(MS(env)+MS(m.f.env))/(MS(m.env)+MS(f.env)) = (103.08+.32)/(.190+.458) =
(103.4)/(.6474) = 159.7 as reported in the table.

3.11 pairedcomp

Keywords: ANOVA, analysis

Usage: `pairedcomp(factorname,lev [,method:T] [,error:term]),
factorname CHARACTER, positive REAL scalar lev <
1, positive integer or term name term, method one
of 'lsd', 'bsd', 'snk', 'hsd', 'regwb', or 'regwr'
pairedcomp(factorname,critval:val), positive REAL
scalar val`

pairedcomp(factorname,lev [,method:T]) does all paired comparisons between the levels of the factor given in factorname at the level of significance lev. If no method keyword is used, then the comparisons are done using the Bonferroni method. method keywords may be used to specify alternate methods. The available methods are: lsd (least significant difference), bsd (Bonferroni), snk (Student-Newman-Keuls), hsd (Tukey's honest significant difference, also called the studentized range procedure), regwb (regw with Bonferroni test), and regwr (regw with studentized) range. Thus, for example, pairedcomp("trt",.01,hsd:T) does paired comparisons between the levels of trt at signficance .01 using the hsd method. pairedcomp("trt",.01) and pairedcomp("trt",.01,bsd:T) are equivalent.

There must be a current ANOVA model and the factor name in the CHARACTER variable factorname must be in the current ANOVA model. factorname must correspond to a single factor, not an interaction. The level lev must be between 0 and 1.

pairedcomp prints a summary of the results, but returns no value. The printed output is one row for each level of the term, sorted from smallest to largest effect, giving the "underlines", level number, and effect.

By default, the error in these tests is taken from the last error term of the current model (the last line of the ANOVA table). You may specify a different error term with a keyword error:term. term may be a number, indicating a line in the ANOVA table, or the name of a term in the ANOVA table, for example, error:4 or error:"ERROR1".

pairedcomp(factorname,critval:val) does all paired comparisons but uses val as the critical value for a t-test between the levels of factorname rather than a computed cutoff.

The comparisons are made via the contrast() command. In particular this implies that the comparisons are adjusted for any other terms in the model and that there should be no missing degrees of freedom in the factor.

Examples:

```
   Cmd> anova("y=a")
   Model used is y=a
                    DF           SS           MS
   CONSTANT          1       15.082       15.082
   a                 4       67.535       16.884
   ERROR1           15       20.132       1.3421

   Cmd> pairedcomp("a",.05,hsd:T) #hsd method
     |         5    -1.86
     | |       2    -1.18
     | |       4    -1.15
       | |     3     1.13
         |     1     3.07
```

```
Cmd> pairedcomp("a",.05,lsd:T) #lsd w/ alpha=.05
   |    5    -1.86
   |    2    -1.18
   |    4    -1.15
        3     1.13
        1     3.07

Cmd> pairedcomp("a",critval:2.13) #lsd w/ alpha=.05 a different way
   |    5    -1.86
   |    2    -1.18
   |    4    -1.15
        3     1.13
        1     3.07
```

3.12 quadmax

Keywords: analysis

Usage: quadmax(A,b[,eq:eqmat][,gte:gtemat][,ckbounds:F]),
A square REAL matrix, b REAL vector with nrows(b)
= nrows(A), eqmat and gtemat REAL matrices with
ncols(eqmat) = ncols(gtemat) = nrows(A) + 1,

quadmax finds the x that maximizes x'Ax + b'x; if the problem is unbounded then quadmax returns an error. A is a p by p real matrix and b is a p by 1 vector.

You may specify linear equality constraints on the solution by using the eq:eqmat keyword phrase. eqmat is an q by p+1 matrix partitioned [Q:y] where Q is q by p and y is q by 1. When this keyword phrase is used, the solution is constrained to satisfy Qx = y. If the constrained problem is unbounded, then quadmax returns an error. If the constraints cannot be met, then quadmax returns NULL.

You may specify linear inequality constraints on the solution by using the gte:gtemat keyword phrase. gtemat is an g by p+1 matrix partitioned [G:z] where G is g by p and z is g by 1. When this keyword phrase is used, the solution is constrained to satisfy Gx >= z elementwise. If the constrained problem is unbounded, then quadmax returns an error. If constraints cannot be met, then quadmax returns NULL.

For example, in a three variable mixture problem you might have the equality constraint that the sum of the x's is 1 and each element of x is at least .05. Then use eq:eqmat,gte:gtemat where eqmat is

```
                        [ 1 0 0 .05 ]
[1 1 1 1] and gtemat is [ 0 1 0 .05 ]
                        [ 0 0 1 .05 ]
```

quadmax tries to determine when the problem is unbounded. This can increase the computational time substantially, so you may tell quadmax not to check for unboundedness by using the ckbounds:F keyword phrase.

This only seems reasonable when the problem is known to be bounded, for example, in a problem where the range of the x's is totally bounded by inequality constraints, or in a problem where x'Ax is known to have a unique maximum (A has all negative eigenvalues).

The algorithm used by quadmax can be described as intelligent brute force and will probably be overwhelmed by too many constraints. When you are sure the problem has a bounded solution, be sure to use keyword phrase ckbounds:F.

3.13 randsign

Keywords: permutation test, analysis
Usage: randsign(diffs [,trials:n]), REAL vector diffs, positive integer n

randsign(diffs) computes sum(s_i * diffs_i) for all 2^length(diffs) possible combinations of signs s_i and returns these as a REAL vector.

The form randsign(diffs,trials:n) samples from the distribution of sum(s_i * diffs_i) based on n sets of random signs. This is appropriate when diffs is long, as 2^length(diffs) grows quickly!!

You can use the results of randsign to compute p-values for the randomization equivalent of the paired t-test by finding the fraction of the total differences based on random signs as extreme or more extreme than the observed total difference for the two groups.

Examples:

```
Cmd> x1 <- vector(1,4,2,4,6,3,7) # data set 1

Cmd> x2 <- vector(3,5,3,6,2,9,8) # data set 2

Cmd> diffs <- x2-x1

Cmd> diffs # the differences
(1)            2            1            1            2           -4
(6)            6            1

Cmd> sum(diffs)  # observed total difference
(1)            9

Cmd> out <- randsign(diffs)  # all differences with random signs

Cmd> stemleaf(out)
    1   -1s|7
    4   -1f|555
    9   -1t|33333
   16   -1*|1111111
   24   -0.|99999999
   32   -0s|77777777
```

```
 41   -0f|555555555
 52   -0t|33333333333
 64   -0*|111111111111
 64   +0*|111111111111
 52   +0t|33333333333
 41   +0f|555555555
 32   +0s|77777777
 24   +0.|99999999
 16    1*|1111111
  9    1t|33333
  4    1f|555
  1    1s|7

       1*|1  represents 11  Leaf digit unit = 1

Cmd> sum(out >= 9)/128  # one sided randomization p-value
(1)       0.1875
```

3.14 randt

Keywords: permutation test, analysis
Usage: `randt(dvec, m [,trials:n]), dvec REAL vector, positive integer n`

randt(dvec,m) computes xbar_1 - xbar_2 for all combinations with m data values from dvec in group 1 and the remainder in group 2. The returned value is a vector containing all the differences.

The form randt(dvec,m,trials:n) samples from the distribution of xbar_1 - xbar_2 based on n sets of random assignments to the groups.

You can use the results of randt to compute p-values for the randomization equivalent of the two sample t-test by finding the fraction of the differences as extreme or more extreme than the observed difference for the two groups.

Examples:

```
Cmd> x1 <- vector(3,4,6,9,10,13) # Group 1 data

Cmd> x2 <- vector(2,3,4,4,5) # Group 2 data

Cmd> sum(x1)/6 - sum(x2)/5  # observed difference
(1)          3.9

Cmd> all <- vector(x1, x2) # combined data

Cmd> out <- randt(all,6)  # get all differences

Cmd> length(out)  # how many are there?
(1)          462
```

```
Cmd> stemleaf(out)  # how do they look
    1     -5|2
   15     -4|99955555111111
   39     -3|888888844444440000000000
   75     -2|77777777777777733333333333333333333
  153     -1|999999999999999999999999966666666666666666666666622222222222*
  227     -0|888888888888888888888888555555555555555555555551111111111111*
 ( 92)    +0|22222222222222222222222222226666666666666666666666666666699*
  143      1|33333333333333333333333333333777777777777777777777777777
   87      2|00000000000000000004444444444444444488888888888
   41      3|11111111111155555555555999999999
    8      4|22222666

            1|1  represents 1.1  Leaf digit unit = 0.1

Cmd> sum(out >= 3.9)/462  # one sided p-value
(1)      0.038961
```

3.15 rscanon

Keywords: analysis

Usage: `rscanon(y,x1,x2,...,xk[,block:var1,block:var2,...])`,
REAL vectors y, x1, ...,xk, var1, var2, ... factors;
all should have the same number of rows.

rscanon(y,x1,x2,...,xk) performs the canonical analysis for the quadratic response surface model with response y and predictors x1, ... , xk. The output is a structure with components b0, b, B, x0, y0, H, and lambda, giving the intercept, linear coefficients, the quadratic/cross product coefficient matrix, the stationary point, the predicted response at the stationary point, the matrix of canonical directions, and the eigenvalues respectively.

If the design was blocked, you may include blocking variables using the block: keyword. More than one block keyword may be used, for example rscanon(yield,time,temperature,block:date,block:analyst) The blocking variables should be factors. Output is the same as before, but block adjusted.

Examples:

```
Cmd> #data from example 16-2 of Montgomery

Cmd> x1 <- vector(-1,-1,1,1,0,0,0,0,0,1.414,-1.414,0,0)

Cmd> x2 <- vector(-1,1,-1,1,0,0,0,0,0,0,0,1.414,-1.414)

Cmd> y <- vector(76.5,77.0,78.0,79.5,79.9,80.3,80.0,79.7,\
79.8,78.4,75.6,78.5,77.0)

Cmd> rscanon(y,x1,x2)
component: b0
```

```
(1)        79.94
component: b
(1)      0.99505       0.5152
component: B
(1,1)      -1.3764        0.125
(2,1)        0.125      -1.0013
component: x0
(1)      0.38923      0.30585
component: y0
(1)       80.212
component: H
(1,1)      0.28972      0.95711
(2,1)      0.95711     -0.28972
component: lambda
(1)      -0.9635      -1.4143
```

3.16 typeIIIss

Keywords:
Usage: `typeIIIss(model), obsolete, use anova(model,marginal:T)`

```
typeIIIss(model) is obsolete, use anova(model,marginal:T) instead.
```

3.17 varcomp

Keywords: anova, analysis, random effects, factorial
Usage: `varcomp(model,randomvars[,marg:T][,restrict:F][,nonhier:T])`
`varcomp(emsOutput)`

```
varcomp computes the "ANOVA" estimates of random effects in
mixed effects analysis of variance.  These estimates are linear
combinations of mean squares for random effects and may be
negative.  The arguments to varcomp can be of two types.  First,
varcomp can take a model, list of random factors, and optional
list of keyword modifiers exactly as ems.  Second, varcomp can
take a structure that is the output of an ems with the keep:T
keyword.

varcomp computes unbiased estimates of variance components by
taking linear combinations of mean squares for random terms.
varcomp assumes that the EMS for random terms have no contributions
from fixed factors.  This is true for balanced data and may be
guaranteed in general by using marg:T.  varcomp also computes
a standard error for each estimate.

The value of varcomp is a matrix with one row for each random
term and two columns giving the estimated variance component
and its standard error.
```

Example. Three populations, 4 males nested in each population, 2 females nested in each male, six offspring from each mating, with these randomly assigned to three environments. Male and female are random. First the simple anova.

```
Cmd> anova("y=(pop+m.pop+f.pop+m.f.pop)*env")
Model used is y=(pop+m.pop+f.pop+m.f.pop)*env
                DF          SS          MS
CONSTANT         1      5.4299      5.4299
pop              2      2091.4      1045.7
pop.m            9       112.5        12.5
pop.f            9      370.02      41.113
pop.m.f         27      56.774      2.1027
env              2      206.15      103.08
pop.env          4     0.16527    0.041316
pop.m.env       18      3.4185     0.18992
pop.f.env       18      8.2354     0.45752
pop.m.f.env     54      17.117     0.31698
ERROR1         144      30.448     0.21144
```

Now compute the expected mean squares, and keep the ems output.

```
Cmd> emsstuff<-ems("y=(pop+m.pop+f.pop+m.f.pop)*env",vector("m","f"),
   keep:T,print:T)
EMS(CONSTANT) = V(ERROR1) + 6V(pop.m.f) + 24V(pop.f) + 24V(pop.m) +
   288Q(CONSTANT)
EMS(pop) = V(ERROR1) + 6V(pop.m.f) + 24V(pop.f) + 24V(pop.m) + 96Q(pop)
EMS(pop.m) = V(ERROR1) + 6V(pop.m.f) + 24V(pop.m)
EMS(pop.f) = V(ERROR1) + 6V(pop.m.f) + 24V(pop.f)
EMS(pop.m.f) = V(ERROR1) + 6V(pop.m.f)
EMS(env) = V(ERROR1) + 2V(pop.m.f.env) + 8V(pop.f.env) +
   8V(pop.m.env) + 96Q(env)
EMS(pop.env) = V(ERROR1) + 2V(pop.m.f.env) + 8V(pop.f.env) +
   8V(pop.m.env) + 32Q(pop.env)
EMS(pop.m.env) = V(ERROR1) + 2V(pop.m.f.env) + 8V(pop.m.env)
EMS(pop.f.env) = V(ERROR1) + 2V(pop.m.f.env) + 8V(pop.f.env)
EMS(pop.m.f.env) = V(ERROR1) + 2V(pop.m.f.env)
EMS(ERROR1) = V(ERROR1)
```

From the EMS, we see than (MS(pop.m.f.env)-MS(ERROR1))/2 is an unbiased estimate of V(m.f.env); here, we have (.31698 - .21144)/2 = .05277. Similarly, (MS(pop.f.env)-MS(pop.m.f.env))/8 is an unbiased estimate of V(f.env); here, we have (.45752 - .31698)/8 = .01757. varcomp automates these calculations, as well as providing the standard error.

```
Cmd> varcomp(emsstuff)
              Estimate          SE
pop.m          0.43323     0.24668
pop.f           1.6254     0.80788
pop.m.f        0.31521    0.095472
```

```
pop.m.env      -0.015883      0.010989
pop.f.env       0.017568      0.020532
pop.m.f.env     0.052766      0.032948
ERROR1           0.21144      0.024919
```

Note that variance component estimates can be negative; varcomp does not truncate the estimates at 0. We would get the same output from the following command.

```
Cmd> varcomp("y=(pop+m.pop+f.pop+m.f.pop)*env",vector("m","f"))
```

Chapter 4

User Function Help File

This Chapter contains help for users interested in producing user functions (compiled C or FORTRAN code that is loaded into MacAnova and executed). The material here is a reformatting of file userfun.hlp. A suggested order for reading this material is

1. loadUser
2. User
3. user fun
4. arginfo fun
5. callback fun
6. c macros
7. compile dos compile mac compile unix compile win
8. type codes

4.1 arginfo fun

Keywords: user functions, coding, sample source

Usage: `Type help(user fun) for information on the structure of user functions. Type help(callback fun) for information on the structure of user functions making "call backs" to MacAnova. Type help(arginfo fun) for information on how to enable automatic checking of arguments to a user function.`

`This topic presumes familiarity with topic User() and user fun. It provides a brief introduction to the form of an arginfo function, that is an externally compiled function that can be called by MacAnova to obtain information about the arguments expected by a user function. If`

available, an arginfo function operates transparently to the user of MacAnova. Because of the inherent dependence on the computer and operating system, there are many details that are not covered here. Additional details may be found in topics compile dos, compile mac, compile_unx and compile win.

This topic presumes familiarity with topics user fun and callback fun.

Arginfo functions are not currently possible when compiling for the protected mode DOS version (DJGPP).

Since we have no experience with writing arginfo functions in Fortran, no Fortran related information is provided here.

In the following, 'handle' is used in the Macintosh OS sense, as a pointer to a pointer.

To compile an arginfo function associated with user function foo, say, you need to include a function named 'arginfo_foo' in the source for foo. arginfo_foo should have no arguments and should return a pointer to a vector of long integers, that is it should be declared as

```
long * arginfo_foo(void)
```

When compiling for Windows using Borland C/C++ 4.5, the declaration should be

```
long * _export arginfo_foo(void)
```

The ending of the name of the arginfo function (here 'foo') must match the name of the user function.

arginfo_foo should return a pointer to a vector arginfo of Nargs + 2 long integers, where Nargs is the number of arguments expected by foo, excluding the list, if any, of call back functions (see callback fun).

The first element of vector arginfo (arginfo[0]) must be Nargs >= 1.

The second element of vector arginfo (arginfo[1]) is composed of bit constants that specify various properties of the function (whether it makes call backs, whether it expects pointers or handles, whether its arguments should be data or symbols, and whether a required 68881 co-processor is absent (Macintosh only). Symbolic names for these are defined in header file dynload.h which is automatically included by header file Userfun.h.

Name of bit	Meaning
DOESCALLBACK	Call backs to MacAnova functions will be made
NOCALLBACK	No call backs to MacAnova functions will be made
USESPOINTERS	Arguments should be pointers
USESHANDLES	Arguments should be handles
POINTERUSE	Same as USESHANDLES on Macintosh and same as USESPOINTERS on other systems
SYMBOLARGS	All arguments (except call back function list) are pointers or handles to Symbols
NOSYMBOLARGS	All arguments (except call back function list) are pointers or handles to data

```
    COPROCESSOROK     Co-processor not needed or, if needed, is available
    COPROCESSORERROR A co-processor is needed but not available
```

For example, for a function with default pointer/handle usage that makes call backs and does not expect Symbol arguments, arginfo[1] should be DOESCALLBACK | POINTERUSE | NOSYMBOLARGS. When compiling for a Macintosh, this is equivalent to DOESCALLBACK | USERSHANDLES | NOSYMBOLARGS; when compiling for other computers it is equivalent to DOESCALLBACK | USESPOINTERS | NOSYMBOLARGS

Strictly speaking NOCALLBACK and NOSYMBOLARGS are not needed since they evaluate to 0, but their use can make for clearer code.

The remaining Nargs elements (arginfo[2], arginfo[3], ..., arginfo[Nargs+1]) of the vector are integers that specify the MacAnova types of the user function arguments, using symbolic constants defined in Userfun.h. Typical constants are REALMATRIX, CHARSCALAR, LOGICSCALAR, INTVECTOR, POSITIVEREALVECTOR, NONNEGATIVEINT, LONGVECTOR and SYMHVALUE. The qualifier INT means REAL with integer values; the qualifer LONG means actual long integers as produced by asLong(). See topic type codes for a complete list of permissible constants.

In writing a function for a 68K Macintosh when compiling using Metrowerks CodeWarrior, to ensure correct compilation, all declarations of call back and arginfo functions must be bracketed by

```
#pragma mpwc on
...
#pragma mpwc off
```

Here is an example of a function to provide argument information for fooeval() listed under topic callback fun and executed from MacAnova by, say,

```
Cmd> User("fooeval", "sqrt(PI/2)")
```

Non-Macintosh version:

```
#include "Userfun.h"

static long Fooevalarginfo[] =
      {1, DOESCALLBACK | POINTERUSE | NOSYMBOLARGS, CHARSCALAR};

long * arginfo_fooeval(void)
{
    return(Fooevalarginfo);
}
```

Macintosh version:

```
#include "Userfun.h"

#define info_main main

static long Fooevalarginfo[] =
```

```
        {1, DOESCALLBACK | POINTERUSE | NOSYMBOLARGS, CHARSCALAR};

  #ifndef powerc
  #pragma mpwc on
  #endif
  long * info_main(void)
  {
      long          *arginfo;

      EnterCode();

      arginfo = Fooevalarginfo;
      /*add COPROCESSORERROR to arginfo[1] if appropriate*/
      CHECK68881(arginfo);

      ExitCode();
      return(arginfo);
  }
  #ifndef powerc
  #pragma mpwc off
  #endif

  #ifdef powerc
  RoutineDescriptor arginfo_fooeval =
     BUILD_ROUTINE_DESCRIPTOR(uppArgInfoEntryProcInfo, info_main);
  #endif /*powerc*/
```

When compiled for a 68K Macintosh, this must be compiled separately from fooeval. If the source is in the same file as source for fooeval, some form of conditional compilation should be used so that both arginfo_fooeval and fooeval don't both get compiled at once. The code resource produced should have name arginfo_fooeval and be included in the same resource file as fooeval.

When compiled for a Power PC Macintosh, arginfo_fooeval would normally be in the same source file as fooeval (C function main) and info_main would not be defined to be main. A single compilation would produce a resource file containing resource fooeval with entries fooeval and arginfo_fooeval. The actual entry points would be specified by RoutineDescriptors fooeval and fooeval_arginfo.

See topics compile dos, compile mac, compile unix and compile dos for information on compiling a user function on different types of computers.

See loadUser() and User() for information on how to load and execute a user function.

See topic user fun for information on the structure of a user function not making call backs to MacAnova.

See topic callback fun for information on the structure of a user function making call backs to Macanova.

See topic c macros and header file Userfun.h distributed with MacAnova for C macros that are helpful in writing arginfo functions.

4.2 c macros

Keywords: user functions, coding, sample source
Usage: type help(c macros) for information on available C macros for compiling user functions that may be compiled for more than one type of computer.

This topic presumes familiarity with topics User(), user fun, arginfo fun and callback fun. It describes the use of the C macros in header file Userfun.h in writing user functions in such a way that their code may be compiled on a variety of computers with little or no change. Some of the macros in Userfun.h are helpful even when writing a user function to run on a single type of computer. Among other things, use of these macros ensures that all non-symbol arguments end up as pointers and symbol arguments end up as handles. They can also make it easier to call back to MacAnova. See callback fun.

To make these macros available, the following should appear in your source file

```
#include "Userfun.h"
```

and both files Userfun.h and dynload.h (included by Userfun.h) should be in the same directory as the file being compiled.

Userfun.h also includes macros for working directly with Macanova symbols. These are not discussed here. However, example source fooeval.c includes some example of their use. See Userfun.h for more information.

Userfun.h also defines constants for describing the type and shape of user function arguments. See topic type codes for a complete list.

Here is a brief summary of the most important macros in Userfun.h.

Prefix each user and arginfo function name with EXPORTED:

```
void EXPORTED foo(...)
```

or

```
long * EXPORTED arginfo_fooeval(void)
```

If MACINTOSH is defined, the macros referencing arguments (THEARG, THECOMMAND, THESYMBOL, CALLBACKFUN) normally assume the arguments are handles; if MACINTOSH is not defined, they normally assume arguments are pointers.

If, for some reason, you want to deviate from this convention, you can override it by using one of the following

```
#define POINTERARGS 1  /*arguments assumed to be pointers*/
```

or

```
#define POINTERARGS 0  /*arguments assumed to be handles*/
```

If POINTERARGS is not defined, Userfun.h defines it to be 0 on a Macintosh or 1 otherwise.

Use DOUBLEARG(argx), CHARARG(argx), LONGARG(argx) and SYMBOLARG(argx) to declare arguments other than a list of call back functions. They expand to handles (double ** argx, char ** argx, long ** argx, Symbol ** argx) when POINTERARGS is 0 (Macintosh) and to pointers (double * argx, char * argx, long * argx, Symbol * argx) when POINTERARGS is 1 (everywhere else). If you do deviate and do not provide an arginfo function (see arginfo fun), you will have to include either pointers:T (Macintosh) or pointers:F (otherwise) as an argument to User().

Use CALLBACKLIST(funlist) to declare the call back function list structure. It expands to MacAnovaCBSH funlist (a handle) when POINTERARGS is 0 and to MacAnovaCBSPtr funlist (a pointer) otherwise.

Use arg = THEARG(argx) to obtain a pointer to non-symbol argument. This expands to arg = *argx (dereferencing a handle) when POINTERARGS is 0 and to arg = argx otherwise. On a Macintosh, you should dereference any handle argument again after calling back to a function internal to MacAnova.

Use commandH = THECOMMAND(argx) to obtain a handle (char **) to a CHARACTER argument that is to be an argument to the mvEval() call back function.

Use symhArg = THESYMBOL(argx) to obtain a Symbolhandle (Symbol **) for a symbol type argument.

Use CALLBACKFUN(funlist, funName) to obtain a pointer to function funName in the list of call back functions.

In any user function making callbacks define C macro MVCALLBACKS before include Userfun.h. This results in the declaration of MvCallbackFuns, a global handle or pointer (depending on the value of POINTERARGS) to a call back function list. It this case, one of the first executable lines in the user function should be setMvFuns(funlist) to initialize MvCallbackFuns. Subsequently you can call the standard call back functions by mvPrint(msg), mvAlert(msg), mvEval(cmd), mvIsmissing(&x), mvSeterror(errorNumber), and mvFindfun(funName), where msg, cmd and funName have type char *, x has type double and errorNumber has type long. For example, mvPrint("Hello!") expands to CALLBACKFUN(MvCallbackFuns, print)("Hello").

When compiling for a 68K Macintosh, Userfun.h defines PRAGMAMPWC. This is to be used as follows:

```
#ifdef PRAGMAMPWC
#pragma mpwc on
#endif
```

```
  Declaration of arginfo or call back function(s)

  #ifdef PRAGMAMPWC
  #pragma mpwc off
  #endif
```

This ensures the use of function calling conventions that are compatible with the 68K version of MacAnova which is compiled using MPW C. See the sample files goo.c and fooeval.c below for examples of the use of PRAGMAMPWC.

If the user function makes call backs and has more than one source file, define USERSUBFUNCTION in all but one source file. This assures that MvCallbackFuns will not be multiply defined.

In topic user fun is C code not using these macros for user function goo and its arginfo function arginfo_goo. Separate versions are given there for Macintosh and non-Macintosh use. Here is C code for these functions that uses the macros. It should compile correctly on all platforms. When compiled for a 68K Macintosh, two compilation runs will be required, changing the value of WHICHFUN (1 for goo, 2 for arginfo_goo).

Use of these macros requires that MACINTOSH be defined when compiling for a Macintosh (with powerc also defined for PPC and MW_CW defined if using Metrowerks CodeWarrior compiler), DJGPP is defined when compiling for use with the protected mode DOS version, and WIN32 is defined when compiling for use with the Windows version.

The use of macros such as USERFUN and ARGINFO to define function names is needed to meet the requirements for Macintosh compilation for which an entry point must be named 'main'. The conditional compilation of the user function and arginfo function (depending on whether MAINFUN and/or INFOFUN is defined) is in response to limitations on compilations for a 68K Macintosh for which there can be only one entry point per resource.

We suggest the examples below, source for which is distributed with MacAnova) be used as templates, changing most of the executable code and the augument lists to meet your particular needs.

File goo.c:

```
  #include "Userfun.h"

  #ifndef MACINTOSH
  #define MAINFUN    /*main entry will be compiled*/
  #ifndef DJGPP
  #define INFOFUN    /*arginfo entry will be compiled*/
  #endif /*DJGPP*/
  #define USERFUN goo
  #define ARGINFO arginfo_goo
  #else /*!MACINTOSH*/

  /*
      For PPC MAC, one function must be called main
```

```
    For 68K MAC, only one function is reachable per compilation
    project and it must be called main
*/
#ifdef powerc
#define MAINFUN     /*main entry will be compiled*/
#define INFOFUN     /*arginfo entry will be compiled*/
#define main_goo main
#else /*powerc*/

/* define which of the 2 functions will be compiled*/
#define WHICHFUN 1  /*must be 1 or 2 */
#if WHICHFUN == 1
#define MAINFUN
#else
#define INFOFUN
#endif

#if defined(MAINFUN)
#define main_goo    main
#else
#define info_goo    main
#endif

#endif /*powerc*/

#define USERFUN       main_goo
#define USERFUNENTRY goo
#define ARGINFO       info_goo
#define ARGINFOENTRY arginfo_goo

#endif /*!MACINTOSH*/

#ifdef MAINFUN
/*
  EXPORTED is _export for Windows compiled by Borland C/C++ 4.5
  DOUBLEARG(argx) expands as double * argx or double ** argx, and
  similarly for LONGARG(argn)
*/
void EXPORTED USERFUN(DOUBLEARG(argx), DOUBLEARG(argy), LONGARG(argn),
    DOUBLEARG(argresult))
{
    /* THEARG(argx) expands as argx or *argx */
    double      *x = THEARG(argx);
    double      *y = THEARG(argy);
    long        *n = THEARG(argn);
    double      *result = THEARG(argresult);
    int          i;

    EnterCode();
    *result = 0.0;
    for (i = 0; i < *n; i++)
    {
        *result += x[i]*y[i];
```

```
    }
    ExitCode();
}
#endif /*MAINFUN*/

#ifdef INFOFUN
static long Gooarginfo[] =
{
    4, NOCALLBACK | POINTERUSE | NOSYMBOLARGS,
    NONMISSINGREALVECTOR, NONMISSINGREALVECTOR, LONGSCALAR, REALSCALAR
};

#ifdef PRAGMAMPWC /*PRAGMAMPWC defined in Userfun.h only for 68K Mac*/
#pragma mpwc on
#endif /*PRAGMAMPWC*/
long * EXPORTED ARGINFO(void)
{
    long        *arginfo;

    EnterCode();

    arginfo = Gooarginfo;
    CHECK68881(arginfo);

    ExitCode();

    return(arginfo);
}
#ifdef PRAGMAMPWC
#pragma mpwc off
#endif /*PRAGMAMPWC*/
#endif /*INFOFUN*/

#ifdef powerc /*powerc defined means compiling for Power PC*/
RoutineDescriptor USERFUNENTRY =
    BUILD_ROUTINE_DESCRIPTOR(uppMainEntryProcInfo04, USERFUN);
RoutineDescriptor ARGINFOENTRY =
    BUILD_ROUTINE_DESCRIPTOR(uppArgInfoEntryProcInfo, ARGINFO);
#endif /*powerc*/
```

In topic arginfo fun is C code not using the macros in Userfun.h for user function fooeval and its arginfo function arginfo_foo. Here is C code using the macros that should compile correctly on all platforms using the C macros defined in Userfun.h. Since fooeval calls back to MacAnova, it must define MVCALLBACKS before including Userfun.h. This allows use of macros such as mvPrint and mvEval to call back to MacAnova. See topic callback fun or header file Userfun.h for information on type sprintftype.

File fooeval.c:

```
#define MVCALLBACKS   /*enables call backs; required before include*/
#include "Userfun.h"
```

```
#ifndef MACINTOSH
#define MAINFUN  /*if defined, compile fooeval*/
#ifndef DJGPP
#define INFOFUN  /*if defined, compile arginfo function for fooeval*/
#endif
#define USERFUN fooeval
#define ARGINFO arginfo_fooeval
#else /*MACINTOSH*/

/*
    For PPC MAC, one function must be called main
    For 68K MAC, only one function is reachable per compilation
    project and it must be called main
*/
#ifdef powerc
#define MAINFUN
#define INFOFUN
#define main_fooeval main
#else /*powerc*/
/* define which of the 2 functions this project is for*/
#define WHICHFUN 1

#if WHICHFUN == 1
#define MAINFUN
#else
#define INFOFUN
#endif

#if defined(MAINFUN)
#define main_fooeval main
#else
#define main_info    main
#endif

#endif /*powerc*/
#define USERFUN       main_fooeval
#define USERFUNENTRY  fooeval
#define ARGINFO       main_info
#define ARGINFOENTRY  arginfo_fooeval

#endif /*MACINTOSH*/

#ifdef MAINFUN
void EXPORTED USERFUN(CHARARG(commandarg), CALLBACKLIST(funlist))
{
    char              **commandH = THECOMMAND(commandarg);
    Symbolhandle        result;
    char                line[200];
    sprintftype         sprintf; /*type defined in Userfun.h*/

    EnterCode();

    setMvFuns(funlist); /*initializes global duplicate of funlist*/
```

```
    sprintf = (sprintftype) mvFindfun("sprintf");

    result = mvEval(commandH);

    if (result != (Symbolhandle) 0)
    {
        switch (TYPE(result))
        {
          case CHAR:
            sprintf(line, "STRINGPTR(result) = '%s'", STRINGPTR(result));
            break;

          case REAL:
            if (!mvIsmissing(&DATAVALUE(result,0)))
            {
                sprintf(line, "DATAVALUE(result,0) = %.17g",
                      DATAVALUE(result,0));
            }
            else
            {
                sprintf(line, "DATAVALUE(result,0) = MISSING");
            }

            break;

          case LOGIC:
            sprintf(line, "DATAVALUE(result,0) = %c",
                    (DATAVALUE(result,0)) ? 'T' : 'F');
            break;

          case NULLSYM:
            sprintf(line, "Result is NULL");
            break;

          default:
            sprintf(line,
              "Type %ld of result not CHARACTER, REAL, LOGICAL, or NULL",
              TYPE(result))
        }
        mvPrint(line);
    }
    else
    {
        mvAlert("ERROR: Command produced error");
      /*tell User() error occurred but no message should be printed*/
        mvSeterror(silentCallbackError);
    }
    ExitCode();
} /*fooeval()*/
#endif /*MAINFUN*/

#ifdef INFOFUN
```

```
static long Fooevalarginfo[] =
      {1, DOESCALLBACK | POINTERUSE | NOSYMBOLARGS, CHARSCALAR};

#ifdef PRAGMAMPWC
#pragma mpwc on
#endif /*PRAGMAMPWC*/

long * EXPORTED ARGINFO(void)
{
    long         *arginfo;

    EnterCode();

    arginfo = Fooevalarginfo;
    CHECK68881(arginfo);
    ExitCode();
    return(arginfo);
}
#ifdef PRAGMAMPW
#pragma mpwc off
#endif /*PRAGMAMPW*/

#endif /*INFOFUN*/

#ifdef powerc
RoutineDescriptor USERFUNENTRY =
   BUILD_ROUTINE_DESCRIPTOR(uppMainEntryProcInfo02, USERFUN);
RoutineDescriptor ARGINFOENTRY =
   BUILD_ROUTINE_DESCRIPTOR(uppArgInfoEntryProcInfo, ARGINFO);
#endif /*powerc*/
```

4.3 callback fun

Keywords: user functions, coding, sample source

Usage: Type help(user fun) for information on the structure of user functions. Type help(callback fun) for information on the structure of user functions making "call backs" to MacAnova. Type help(arginfo fun) for information on how to enable automatic checking of arguments to a user function.

This topic provides a brief introduction to the form of a user function that makes "call backs" (executes functions internal to MacAnova). Because of the inherent dependence on the computer and operating system, there are many details that are not covered here. Additional details may be found in topics compile dos, compile mac, compile_unx and compile win.

It presumes familiarity with topic user fun which describes the structure of user functions not making call backs.

See headerfile Userfun.h distributed with MacAnova for C macros that are helpful in writing user functions.

See loadUser() and User() for information on how to load and execute a user function.

See topic arginfo fun for information on how to make it possible for MacAnova to obtain information about a user function for automatic argument checking.

Since we have no experience in making call backs from a Fortran routine, no Fortran tips are given.

In the following, 'handle' is used in the Macintosh OS sense, as a pointer to a pointer.

Structure of user functions calling back to MacAnova

In addition to regular arguments (pointers or handles to data or symbols; see topic user fun), a user function that calls back to MacAnova must have an extra argument providing a list of functions that can be called. This is either a pointer (non-Macintosh) or handle (Macintosh) to a MacAnovaCBS structure (defined in header file dynload.h, included by header file Userfun.h).

Example of non-Macintosh declaration

```
  void fooclbck(char * m, MacAnovaCBS * funlist)
or
  void fooclbck(char * m, MacAnovaCBSPtr funlist)
```

Example of Macintosh declaration

```
  void fooclbck(char ** m, MacAnovaCBS ** funlist)
or
  void fooclbck(char ** m, MacAnovaCBSH funlist)
```

Here is the current definition of a MacAnovaCBS structure taken from header file dynload.h:

```
  typedef struct MacAnovaCBS
  {
      void          (*print)(char *);
      void          (*alert)(char *);
      Symbol **     (*eval)(char **);
      long          (*ismissing)(double *);
      void          (*seterror)(long);
      void *        (*findfun)(char *);
  } MacAnovaCBS, *MacAnovaCBSPtr, **MacAnovaCBSH;
```

All the components are pointers to single argument functions internal to MacAnova.

'print' points to mvPrint which expects a pointer to null terminated character vector (a "string") as argument. It inserts the string in the

MacAnova output stream, usually the screen or command/output window. Virtually all MacAnova output is printed with mvPrint. If output is being spooled to a file (see spool()), mvPrint correctly handles it.

'alert' points to mvAlert() which expects a pointer to a string as argument. In a windowed version (Macintosh, Windows, Motif), this displays the string in a dialog box. In other versions, 'alert' is equivalent to 'print'.

'eval' points to mvEval which expects a handle to a character string (type char **) as argument. mvEval evaluates this string as if it were input to MacAnova, almost as if it were the text of a macro, and returns a handle of type Symbolhandle as value. Just as in a macro, this is the value of the last expression evaluated. C macros in dynload.h allow access to the type (REAL, CHARACTER, ...), dimension and value of the value returned by mvEval. The argument to mvEval must be a handle (char **) even in a non-Macintosh user function.

'ismissing' points to mvIsmissing which expects a pointer to double (double *) as argument. If x is a pointer to a double vector, mvIsmissing(&x[i]) (or mvIsmissing(x+i)) returns 1 if x[i] is MISSING and 0 otherwise.

'seterror' points to mvSeterror which expects a long integer as argument. mvSeterror(code) sets a variable that will be checked by User() on return. If the value is non-zero User() treats it as an error. Unless code = silentCallbackError (defined whenever MacAnovaCBS is defined), User will print the value.

'findfun' points to mvFindfun which expects a pointer to a string containing the name of an internal MacAnova function as argument. mvFindfun returns a pointer to void (C type void *) which must be cast to a function pointer of the appropriate type. A NULL return value indicates the function could not be found.

On some systems, you may be able to access any function known to MacAnova; on others, the available functions are limited to those in a short list. The functions available always include the following functions that can be used to allocate and de-allocate memory and to create and decode character strings. Additional functions may be available on other systems.

`char ** mygethandle(long n)`	Allocate n bytes of memory and return handle to the space allocated
`void mydisphandle(char ** x)`	De-allocate memory referenced by handle x
`char ** mygrowhandle(char **x, long n)`	Allocate n bytes, copy at most n bytes of x to it and then de-allocate x.
`int sprintf(char * bf, char * fmt, ...)`	Formatted "print" to buffer bf
`int sscanf(char * bf, char * fmt, ...)`	Formatted "scan" of buffer bf

C types for these functions are defined in header file Userfun.h so that you can use the following to declare local pointers to them:

```
mygethandletype      mygethandle;
mydisphandletype     mydisphandle;
mygrowhandletype     mygrowhandle;
sprintftype          sprintf;
sscanftype           sscanf.
```

See below for an example.

Memory management functions mygethandle, mydisphandle and mygrowhandle work with handles (pointers to pointers) in all versions.

The char ** arguments to mydisphandle and mygrowhandle must have been allocated by mygethandle.

On a Macintosh, although memory is allocated using Macintosh OS function NewHandle, the values returned by mygethandle and mygrowhandle cannot be used as handle arguments to Macintosh OS functions such as DisposHandle.

It appears that calling back to these sprintf and sscanf is the only way to use them in the protected mode DOS version; in other versions, you can probably use them directly.

In writing a 68K Macintosh user function, to ensure correct compilation, all declarations of call back and arginfo functions must be bracketed by

```
#pragma mpwc on
...
#pragma mpwc off
```

This is because the released 68K versions of MacAnova are compiled using MPW C.

Here is an example of a function that calls back to MacAnova. It uses mvEval to evaluate its first argument as a command and then uses call back functions to print a message describing the result of the evaluation. A typical use might be User("fooeval","sqrt(2*PI)").

Non-Macintosh version:

```
#include "Userfun.h"

void fooeval(char * commandarg, MacAnovaCBSPtr funlist)
{
    char              **commandH = &commandarg;
    Symbolhandle        result;
    char                line[200];
    void              (*mvPrint)(char *) = funlist->print;
    void              (*mvAlert)(char *) = funlist->alert;
    long              (*mvIsmissing)(double *) = funlist->ismissing;
    Symbolhandle      (*mvEval)(char **) = funlist->eval;
```

```
void              (*mvSeterror)(long) = funlist->seterror;
void             *(*mvFindfun)(char *) = funlist->findfun;
sprintftype         sprintf;

/* the code from here to END is the same for any version */
sprintf = (sprintftype) mvFindfun("sprintf");

result = mvEval(commandH); /* have MacAnova evaluate the command*/

if (result != (Symbolhandle) 0)
{
    /*
      C macros TYPE, STRINGPTR, DATAVALUE and constants
      CHAR, REAL, LOGIC, and NULLSYM are defined in Userfun.h
      along with other macros for working with Symbols
      */
    switch (TYPE(result))
    {
      case CHAR:
        sprintf(line,
                "STRINGPTR(result) = '%s'", STRINGPTR(result));
        break;

      case REAL:
        if (!mvIsmissing(&DATAVALUE(result,0)))
        {
            sprintf(line,
                    "DATAVALUE(result,0) = %.17g", DATAVALUE(result,0));
        }
        else
        {
            sprintf(line, "DATAVALUE(result,0) = MISSING");
        }

        break;

      case LOGIC:
        sprintf(line, "DATAVALUE(result,0) = %c",
                (DATAVALUE(result,0)) ? 'T' : 'F');
        break;

      case NULLSYM:
        sprintf(line, "Result is NULL");
        break;

      default:
        sprintf(line,
         "Type of result not CHARACTER, REAL, LOGICAL, or NULL");
    }
    mvPrint(line);
}
else
{
```

```
        mvAlert("ERROR: Command produced error");
  /*Tell User an error has occurred but code should not be printed*/
        mvSeterror(silentCallbackError);
    }
    /* END */
  }
```

Macintosh version:

```
  #include "Userfun.h"
  /*
    On a Macintosh, the main entry must be called main;  the function
    is found by the name of its code resource
  */
  void main(char ** commandarg, MacAnovaCBSH funlist)
  {
      char              **commandH = commandarg;
      Symbolhandle        result;
      char                line[200];
  #ifndef powerc /*if compiled for 68K Macintosh*/
  #pragma mpwc on
  #endif
      void              (*mvPrint)(char *) = funlist->print;
      void              (*mvAlert)(char *) = funlist->alert;
      long              (*mvIsmissing)(double *) = funlist->ismissing;
      Symbolhandle      (*mvEval)(char **) = funlist->eval;
      void              (*mvSeterror)(long) = funlist->seterror;
      void             *(*mvFindfun)(char *) = funlist->findfun;
  #ifndef powerc
  #pragma mpwc off
  #endif
      EnterCode();
  /* the code from here to END is the same for any version */
      . . . . . see above . . . . .
  /* END */
      Exitcode();
  }

  #ifdef powerc /*powerc defined means compiling for Power PC*/
   RoutineDescriptor fooeval =
      BUILD_ROUTINE_DESCRIPTOR(uppMainEntryProcInfo02,main);
  #endif /*powerc*/
```

4.4 compile dos

Keywords: user functions, compiling

Usage: Type help(compile dos) for information on how to compile a user function for use with the protected mode DOS version of MacAnova.

This topic provides some details about compiling a user function for use

with the protected mode DOS version of MacAnova. User functions are not implemented in the real mode version (BCPP). Compilation uses version 2.0 of the DJGPP compiler.

DJGPP uses the dxe format for loading. This is a simple but restricted method of loading. In particular, you can access only one entry point (function) in a file. You should compile the file with gcc as usual as in

```
  gcc -c goo.c subs.c
```

and then run dxegen (supplied with DJGPP 2.0) as in

```
  dxegen goo.dxe _goo goo.o subs.o -lm -lc
```

The first two arguments to dxegen are the output file and the entry point to be made visible for loading (note the prepended underscore). These are followed by the compiled (*.o) files and arguments specifying libraries to be searched.

There are some restrictions on your source file. Not all library functions may be used. Excluded functions include input/output functions and their relatives such as sprintf and sscanf. In addition there are some naming restrictions. For example, you can't have functions foo and foo2 and try to load entry point _foo, but you could have foo and dofoo (it seems that the leading string must be unique).

Because sprintf and sscanf are frequently needed, (sprintf is often used to build output lines or error messages), they are included in the list of functions known to mvFindfun.

Because you can have only one entry point using dxe files, you cannot also provide arginfo_foo() to check the number and types of arguments. Moreover, you must use 'callback:T' and/or 'symbols:T' on User() when executing a user function that makes call backs and/or expects Macanova symbols as arguments.

4.5 compile mac

Keywords: user functions, compiling

Usage: Type help(compile mac) for information on how to compile a user function for use with Macintosh versions of MacAnova.

This topic provides some details about compiling a user function for use with the Macintosh versions of MacAnova. It assumes the Metrowerks CodeWarrior compiler is used.

The Macintosh is a bit more complicated than other platforms, since there are two kinds of processors (PPC and 68K) to support. The 68K case is further complicated by the fact that code may or may not compiled to use a 68881 math coprocessor.

In coding a Macintosh user function, if you make a pointer by

dereferencing a handle argument, you should dereference it again after calling back to a function internal to MacAnova since its location in memory may have been changed by the call back.

User functions and arginfo functions are compiled into code resources in files of type 'rsrc'. PPC resources must have resource type 'MVPP'; 68K resources not requiring a 68881 coprocessor must have resource type 'MV6n'; and 68K resources requiring a coprocessor must have resource type 'MV6c'.

User() accesses the resources themselves by name. The resource for a function should have the name, say 'foo', you will give in your User() call or 'arginfo_foo. All resources of the same type should in a file should have distinct resource numbers, say 4000, 4001, ... A PPC user function may be in a resource with a different name, in which case you have to provide the name of the resource using User(funName, resource:resName, ...).

Source files for both PPC and 68K user functions must have a function named 'main', plus possibly other functions.

PPC code resources, but not 68K ones, can have additional entry points, usually an arginfo function, but occasionally other user functions.

Macintosh 68K user functions

For 68K user functions, it is necessary to set 68000 register A4 so that global variables will be found. Using CodeWarrior, this is accomplished by including

```
EnterCodeResource();
```

immediately after declaring local variables and before any reference to global variables, and including

```
ExitCodeResource();
```

immediately before returning. When C macro MACINTOSH is defined but powerc is not, macros EnterCode() and ExitCode() defined in header file Userfun.h expand to EnterCodeResource() and ExitCodeResource(). Otherwise they expand to nothing.

68K code resources have just one entry point which must be named 'main', so a single resource cannot include both a user function and its arginfo function. However, the Codewarrior compiler has a "Merge to file" option that allows you to add to an existing resource file the resource created when compiling a function.

PPC User Functions

PPC code resources may have multiple entry points which are taken from the names of global RoutineDescriptor variables in the source. Their names should similar to 'goo' and 'arginfo_goo' or 'fooeval' and 'arginfo_fooeval'. The name given to the function that actually codes the user function should be 'main' and the name given to the function with the arginfo function code should be something like 'main_info' different from the name given to the arginfo. Here are typical RoutineDescriptor declarations.

```
  RoutineDescriptor goo =
      BUILD_ROUTINE_DESCRIPTOR(uppMainEntryProcInfo04, main);
  RoutineDescriptor arginfo_goo =
     BUILD_ROUTINE_DESCRIPTOR(uppArgInfoEntryProcInfo, info_main);
```

Argument uppMainEntryProcInfo04 is appropriate for a user function expecting 4 arguments, including the call back function list. For a function expecting 5 arguments, use uppMainEntryProcInfo05, and so on.

Since you will normally use the funName given in the User("funName",...) call for both the name of the resource and the name of the RoutineDescriptor entry point, you will ordinarily include only one user function (and its arginfo function) in a resource. If you include more than one user function in a resource, you must use keyword phrase 'resource:Resname' to specify the resource name. An arginfo function must be in the same resource as its user function.

Setting up Macintosh 68K projects

Here is how to set things up to create a 68K code file with resources 'fooeval' and 'arginfo_fooeval' based on file fooeval.c listed in topic c macros. This has been written in such a way that only one of fooeval or arginfo_fooeval will be compiled, depending on the value of C macro WHICHFUN.

(1) Create two MacOS 68K CodeWarrior project files, one for fooeval and the other for arginfo_fooeval. They should both have source files fooeval.c, Userfun.h and dynload.h. See below for library files needed.

(1.a) Specify Code Resource for the project type for both projects.

(1.a) For both projects specify resource type 'rsrc' and the same resource file, say, Fooeval.rez as Project options. The fooeval project should specify 'fooeval' and 4000 as the resource name and number. The arginfo_fooeval project should specify 'arginfo_fooeval' and 4001 as the resource name and number, and should have Merge to File checked. The resource numbers are arbitrary but should be different. The resource type should be 'MV6c' or 'MV6n', depending on whether you are compiling to use a 68881 math coprocessor.

(1.b) Both projects should specify processor options 68020 Codegen, 4 byte ints, 8 byte doubles and far data. If you are compiling to use a math coprocess, also specify 68881 Codegen,

(1.c) Set linker options Link Single Segment.

(1.d) For a user function making call backs (as does fooeval), C/C++ language option MPW Newlines should be checked.

(1.e) If you reference any C library functions such as strcpy (fooeval does not) you will need library file 'ANSIFa(N/4i/8d)C.A4.68K.Lib' and possibly 'MathLib68K Fa(4i/8d).A4.Lib' ('ANSIFa(N/4i/F/8d)C.A4.68K.Lib' and 'MathLib68K Fa(4i/f/8d).A4.Lib' if compiling to use a 68881 math comprocessor). You may also need MacOS.lib. For example, all three

libraries are needed if you use the library version of sprintf. None is required for fooeval.c as written.

(1.f) Both projects should specify a prefix file, say Userfun.pch, containing at least the following:
```
#define MACINTOSH
#define MW_CW
```

(2) Edit fooeval.c to define C macro WHICHFUN as 1 and compile the fooeval project.

(3) Re-edit fooeval.c to define WHICHFUN as 2 and compile the arginfo_fooeval project.

You end up with one resource file 'Fooeval.rez' containing resources 'fooeval' and 'arginfo_fooeval'.

Setting up Macintosh PPC projects

Here is how to set up a CodeWarrior project to compile a PPC version of fooeval using source file fooeval.c listed in topic c macros.

(1) Create a MacOS PPC project with source files fooeval.c, Userfun.h and dynload.h and library files MPCRuntime.Lib and InterfaceLib. See below for other library files.

(1.a) Specify Code Resource for the project type

(1.b) Specify resource file 'fooevalppc.rez' of type 'rsrc'. The resource type must be 'MVPP'. The resource name should be 'fooeval'. The resource number should be different from any other PPC user functions you might be using simultaneously.

(1.c) Specify Main entry 'main' for the PPC linker.

(1.d) For a user function making call backs (as does fooeval), C/C++ language option MPW Newlines should be checked.

(1.e) If you reference any C library functions such as strcmp you should add libraries 'ANSI C.PPC.Lib' and 'MathLib' and file 'console.stubs.c'.

(2) Compile the fooeval PPC project.

You end up with a resource file fooevalppc.rez containing resource 'foo'. You load it into MacAnova by loadUser("fooevalppc.rez") and execute it by, say, User("fooeval","exp(-x^2/2)/sqrt(2*PI)").

4.6 compile unix

Keywords: user functions, compiling
Usage: Type help(compile unix) for information on how to compile a user function for use with a Unix version of Macanova, including Motif.

This topic provides some details about compiling a user function to be used with a Unix version of MacAnova (including Unix Motif).

Hewlett-Packard UX (HPUX)

The file to load must be a shared library. This can be constructed, for example, by

```
cc -c +z -Aa fooeval.c
ld -b fooeval.o -o fooeval.sl
```

It should be loaded by loadUser("fooeval.sl").

At present (version 4.05 release 1), there may be unsatisfied external problems when linking with system libraries.

Other Unix Versions

Compilation and linking commands may be different. MacAnova will have to have been compiled and linked with functions for using a shared library. The actual loading of shared libraries and execution of routines in them is done in file dynload.c. Currently (July 1997) this has been coded only for HPUX (using shl_load() and shl_findsym()). Most Unix versions will have similar functions. For example, on IRIX, the corresponding functions are dlopen() and dlsym().

4.7 compile win

Keywords: user functions, compiling
Usage: Type help(compile win) for information on how to compile a user function for use with the Windows version of MacAnova.

This topic provides some details about compiling a user function using Borland C/C++ for use with the Windows version of MacAnova.

Set up the project to construct a 32 bit DLL.

Change the default Project Options as follows:

- Add WIN32 to the list of defines
- Set 32-bit Compiler Processor Data Alignment to Quad word (8 bytes).
- Set Resources Target Windows Version to Win32

In the source, add the modifier "_export" to the functions in the project, as in

```
void _export goo(double *x, double *y, long *n, double *result)
long * _export goo_arginfo(void)
```

This will be accomplished automatically if you include header file Userfun.h, and preface routine names with EXPORTED instead of _export.

Entry names will be prefixed by the compiler with '_'. For this reason, User("_foo", ...) and User("foo", ...) are equivalent.

Files goo.c and fooeval.c listed in topic c macros are examples of user functions that compile for Windows.

4.8 loadUser

Keywords: user functions, loading
Usage: loadUser(fileName [,reload:T or clear:T]), CHARACTER scalar fileName.

loadUser(FileName) loads a user function (separately compiled routine) into MacAnova. FileName should be a quoted string or CHARACTER scalar giving the name of the file containing the user function to be loaded. Once loaded, a user function can be executed by function User(). As usual, in windowed versions (Macintosh, Windows, Motif), FileName can be "". If the file has been previously loaded, it is not reloaded, but it may be put at the start of the entry search list for the next use of User().

loadUser(FileName, reload:T) does the same, except that the file will be reloaded, even if it has been previously loaded into MacAnova.

loadUser(FileName, clear:T) does the same, except all previously loaded files will be forgotten.

On some systems, the user function can be written in Fortran, although some features such as call back functions and argument checking may not be available.

Functions loadUser() and User() are inherently specific to a particular computer system although it is possible to write user functions that can be compiled on multiple systems without change.

Unix:
  FileName must be the name of a shared library.
Windows:
  FileName must be the name of a DLL.
Protected mode DOS (DJGPP):
  FileName must be the name of a dxe file.
Macintosh:
  FileName must be the name of a file containing one or more code resources. The PPC version of MacAnova can call both 68K and PPC code resources, but a 68K version of MacAnova can call only 68K code resources. Resource types must be one of 'MVPP' (PPC), 'MV6n' (68K without coprocessor) or 'MV6c' (68K with coprocessor).

4.9 type codes

Keywords: user functions, coding
Usage: Type help(type codes) for a complete list of argument type and shape codes to be returned by an arginfo function.

This topic lists the type and shape codes that may be used by an arginfo function to provide information about the arguments expected by a user function (see topic arginfo fun). They are all integer constants defined in header file Userfun.h.

Scalar argument (all dimensions 1)
CHARSCALAR, LOGICSCALAR, REALSCALAR, NONMISSINGREAL, POSITIVEREAL, NONNEGATIVEREAL, INTSCALAR, POSITIVEINT, NONNEGATIVEINT, LONGSCALAR, POSITIVELONG, NONNEGATIVELONG

Vector argument (all dimensions beyond first, if any, are 1)
CHARVECTOR, LOGICVECTOR, REALVECTOR, NONMISSINGREALVECTOR, POSITIVEVECTOR, NONNEGATIVEVECTOR, INTVECTOR, POSITIVEINTVECTOR, NONNEGATIVEINTVECTOR, LONGVECTOR, POSITIVELONGVECTOR, NONNEGATIVELONGVECTOR

Matrix argument (no more than 2 dimensions >= 1)
CHARMATRIX, LOGICMATRIX, REALMATRIX, NONMISSINGREALMATRIX, POSITIVEMATRIX, NONNEGATIVEMATRIX, INTMATRIX, POSITIVEINTMATRIX, NONNEGATIVEINTMATRIX, LONGMATRIX, POSITIVELONGMATRIX, NONNEGATIVELONGMATRIX

Square matrix argument
REALSQUAREMATRIX

Array argument
CHARARRAY, LOGICARRAY, REALARRAY, NONMISSINGREALARRAY, POSITIVEARRAY, NONNEGATIVEARRAY, INTARRAY, POSITIVEINTARRAY, NONNEGATIVEINTARRAY, LONGARRAY, POSITIVELONGARRAY, NONNEGATIVELONGARRAY

Arbitrary Symbol argument
SYMHVALUE

The qualifiers INT, POSITIVE and NONNEGATIVE imply all elements must be non-MISSING.

These codes are applicable both for user functions whose arguments are pointers or handles to data, and for user functions whose arguments are pointers or handles to MacAnova symbols. SYMHVALUE should only be used when symbol arguments are expected and then only when the argument is not restricted to one type and shape or may has type different from REAL, LOGICAL, CHARACTER or LONG.

4.10 User

Keywords: user functions, executing

Usage: User(funName [,resource:resName][,control keyword phrases],arg1 [,...]), funName and resName CHARACTER scalars; control keyword phrases are any of callback:T, symbols:T, pointers:T and quiet:T; arg1, ... arguments to a user function; if argument is keyword phrase other than 'protect:arg', it is returned, possibly modified.

User(FuncName, arg1, arg2, ...) executes a user function, that is, a compiled routine external to MacAnova. Quoted string or CHARACTER scalar FuncName specifies the name of a user function whose code is in a file previously loaded by loadUser(). arg1, arg2, ... are arguments that will be passed to the function. You must have a least one argument in addition to FuncName and no more than 20 (13 in the Macintosh PPC version). Depending on the compiler and system, you may be required to include leading or trailing underscore characters '_' in FuncName, say User("foo_",...) or User("_foo", ...) instead of User("foo", ...).

A 68K version of MacAnova cannot execute a user function compiled for a PPC.

User(FuncName, quiet:T, arg1, ...) does the same except warning messages, if any, are suppressed.

User(FuncName, callback:T, arg1, ...) specifies the function is known to "call back" to MacAnova, that is, to execute functions internal to MacAnova. On a Macintosh, the type of user function (PPC or ordinary 68K) must match the version of MacAnova. See topic callback fun.

If the MacAnova version requires a 68881 math coprocessor, there can be problems if a user function that makes call backs does not require a coprocessor.

User(FuncName, symbols:T, arg1, ...) specifies that all the arguments are to be passed as complete MacAnova "symbols", including all dimension information. This should be used only with a user function specifically written to make use of MacAnova symbols.

The PPC Macintosh version cannot pass symbol arguments to a 68K user function.

User(FuncName, pointers:T or F, arg1, ...) changes the default way arguments are passed, either as "pointers" (pointers:T) or as "handles" (pointers:F). On all but Macintosh computers, the default is pointers:T. You are unlikely ever to need to use this keyword.

User(FuncName, resource:ResName, arg1, ...) specifies the name of the PPC Macintosh resource containing the user function. This option is not needed in other versions and needed on a PPC only when the resource name differs from the function name.

You can use more than one of the preceding keywords phrases together (User("goo", resource:"foo",quiet:T,callback:T,x,result:0)).

On most systems, it is possible to include with a user function an "arginfo" function that MacAnova can call to obtain information about the user function. The information includes the number of arguments expected, their types and shapes expected (for example, CHARACTER scalar, REAL matrix), and whether the function makes call backs or expects "symbol" arguments (see above). This allows automatic argument checking. If function is compiled without an arginfo function, using the wrong number or type of arguments will usually result in a crash or other undesirable behavior. In particular, a user function will not be able to handle MacAnova symbol arguments unless it has been specially written to be able to understand their structure.

You normally do not need to use keywords 'callback', 'symbols' and 'pointers' if the user function has an associated arginfo function which is possible on all systems except protected mode DOS.

See topics user fun and arginfo fun for information about the form of user and arginfo functions.

Interpretation of FuncName

Unix, Motif and Windows:
FuncName should be the name of the function being called, possibly modified by a leading or trailing '_' (leading '_' when compiling for Windows with Borland C 4.5). In Windows and Unix versions where it is known entry names start with '_', when the function is not found using the name as provided, a second search is made after prepending '_' to the name. Thus if User("_foo", ...) would be successful, so will be User("foo", ...).

Protected mode DOS (DJGPP):
FuncName should be the same as the name of the .dxe file loaded by loadUser() that contains the code except that directory information and the extension ".dxe" may be omitted. Thus after loadUser("../foo.dxe"), you can use any of User("../foo.dxe",...), User("foo.dxe",...) or User("foo",...). If there is more than one file with the same name attached (for example, "/a/foo.dxe" and "/b/foo.dxe", you should use the complete path name.

PPC Macintosh user function
FuncName is the name of the user function. This will usually also be the name of the PPC code resource containing the user function. If it is not, you need to include keyword phrase 'resource:ResName' as an argument, where ResName is a quoted string or CHARACTER scalar specifying the resource name.

68K Macintosh user function:
FuncName should be the name of the 68K code resource containing the user function (only one user function per resource). If 'resource:ResName' is an argument, ResName must be identical with FunName. It is an error to attempt to call a 68K user function that

requires a 68881 or 68882 math co-processor on a Macintosh without one.

User function arguments

All arguments except the function name and 'callback', 'quiet', 'symbols', 'pointers' and 'resource' keyword phrases are passed to the user function as its arguments

Only copies of keyword phrase arguments are passed to a user function. This means that the user function can modify these arguments without danger of changing any MacAnova variable.

Non-keyword phrase arguments to User() (except the user function name) are passed to the user function without being copied. If the argument is a named MacAnova variable and the user function modifies it, the value of the variables itself is changed. If the argument is a literal number (User("foo", 1, 2, 0)) or expression (User("foo", sqrt(2)+3, log(4), 19^2)) the function can safely change the argument without danger to any variable.

Example:
Suppose fooadd expects three arguments, and modifies the third by assigning the sum of the first two. Then

```
Cmd> c <- 0; User("fooadd", 1, 2, c)
```

returns no value (actually a NULL), but c has been changed to 3 (= 1+2). However,

```
Cmd> c <- 0; User("fooadd", 1, 2, protect:c)
```

will not change c itself, but only a copy.

In addition to being copied before use, all keyword phrase user function arguments are returned, possibly modified, as the value of User(). When there are two or more such keyword arguments, a structure is returned with component names taken from the keyword names.

Keyword 'protect' is special, in that its only effect is to cause its value to be copied before being passed to the user function; its value is not returned by User(). Thus the use of 'protect' in the example makes the user function useless: c does not get changed because it is protected by a keyword, but the modified value is not returned either. The following both protects c and causes the modified value of the last argument to be returned as the value of User().

```
Cmd> c <- 0;User("fooadd", 1, 2, result:c)
```

This returns 3 = 1 + 2, but c would be unchanged. In place of variable c for result, you could use any REAL scalar (result:0). This serves to provide space for the answer.

```
Cmd> User("fooadd", left:1, right:2, result:0)
```

returns a structure with components 'left', 'right' and 'result' containing the possibly modified values of the original arguments.

It is essential that the size of any argument that is to be modified matches the size that is expected by the user function. Thus, if foocat is a user function that concatenates its first two arguments into a third argument, the length of the third argument should be the combined length:

```
Cmd> User("foocat", run(4), run(7), combined:rep(0,11)
```

In this example, for the third argument to have fewer than 11 elements would lead to unpredictable results, possibly even a crash of MacAnova.

Except when symbols:T is an argument, all user function arguments are either REAL, LOGICAL, CHARACTER or LONG variables. REAL arguments are passed as double precision data, as are LOGICAL arguments (True = 1.0, False = 0.0). If an argument is a matrix or array, the values are ordered such that the first subscript changes fastest. See user fun for more information.

LONG is a special MacAnova type whose values are long integers between -2147483647 and 2147483647 = 2^31 - 1. A LONG argument can be created only by function asLong(). A LONG argument that is returned (result:asLong(x)), is turned into a REAL quantity before being returned. See asLong().

```
Cmd> User("goo", run(10), run(10), asLong(10), result:0)
```

invokes user function goo with three REAL arguments and one long argument.

For virtually unlimited flexibility, when keyword phrase 'symbols:T' is an argument to User(), all user function arguments are passed as symbols -- MacAnova objects which encapsulate the data, type, and dimensions of a variable. Thus

```
Cmd> User("foo", symbols:T, x, y, result:z)
```

passes x, y and z to 'foo' as symbols. You cannot have some user function arguments be symbols and some just data; all must be symbols or none.

4.11 user fun

Keywords: user functions, coding, sample source

Usage: Type help(user fun) for information on the structure of user functions. Type help(callback fun) for information on the structure of user functions making "call backs" to MacAnova. Type help(arginfo fun) for information on how to enable automatic checking of arguments to a user function.

This topic provides a brief introduction to the form of a user function (routine compiled separately from MacAnova) that can be loaded by loadUser() and executed by User(). Because of the inherent dependence on the computer and operating system, there are many details that are not covered here. Additional details may be found in topics compile dos, compile mac, compile_unx and compile win.

See headerfile Userfun.h distributed with MacAnova for C macros that are helpful in writing user functions.

See loadUser() and User() for information on how to load and execute a user function.

See topic callback fun for information on the structure of a user function that makes call backs to Macanova. It presumes familiarity with this topic (user fun).

See topic arginfo fun for information on how to make it possible for MacAnova to obtain information about a user function for automatic argument checking.

On some systems, if you are willing to forego automatic argument checking, creating a user function file may be as simple as recompiling and linking existing code with certain options set. On others you may need to modify the code to include C header file Userfun.h which defines various constants and C macros. You will almost certainly need to use Userfun.h if you write a user function that makes call backs (executes routines internal to MacAnova) or provides argument checking capability.

Structure of a user function

Most of this discussion assumes the user function is written in C although a few tips are given for user functions written in Fortran. If you write a user function in Fortran, you need to be aware that all its arguments are pointers, that is the function receives the location in computer memory of each argument, not its value.

Header file Userfun.h should normally be included in the C source, especially if you expect that the user function will be compiled for more than one computer type. Userfun.h not only contains information that may be essential for compilation (including type declaration for symbols; see above), but also contains many C macros that make coding easier. In particular, it contains macros allowing you to write a user

function that may be compiled with little or no change on Unix, Macintosh and Windows. However, to make clear the principles, the structure of a user function is illustrated without using these macros.

Note: Header file Userfun.h itself includes header dynload.h which is also distributed with MacAnova. Both need to be available when compiling a user function.

Value returned:
The user function should not return a value (C type void, Fortran subroutine).

Non-Macintosh argument types
Each user function argument must be declared as a pointer. The legal types are double * (REAL or LOGICAL data), char * (CHARACTER data), long * (LONG data), or Symbol * (symbol argument). For Fortran, these are double precision, character and integer*4 (symbol not possible).

Example of non-Macintosh declaration:
C:
```
void goo(double * x, double * y, long * n, double * result)
```

Fortran:
```
subroutine goo(x, y, n, result)
integer*4  n
double precision  x(n), y(n), result
```

Macintosh argument types
Each argument must be declared as a pointer to a pointer, known to Macintosh programmers as a "handle". Thus the legal types are double ** (REAL or LOGICAL data), char ** (CHARACTER data), long ** (LONG data), or Symbol ** (symbol argument. Type Symbolhandle declared in Userfun.h is equivalent to Symbol **.

It is probably not possible to do this directly in Fortran. A C interface to the Fortran subroutine will be required.

Example of Macintosh declaration:
```
void goo(double ** x, double ** y, long ** n, double ** result)
```

When "handle" is used below, it always means a pointer to a pointer, not any of the various handles used when programming for Windows.

The reason for the use of handles as arguments to functions is that MacAnova functions may allocate memory. On the Macintosh, this can move the contents of previously allocated memory, so that a pointer would no longer be valid. However, the handle remains valid even if the pointer it points to changes.

If you make a pointer by dereferencing a handle argument, you should dereference it again after calling back to a function internal to MacAnova since its location in memory may have changed.

Executable statements
There should be no direct input or output statements (you can do output using a callback function) or any direct memory allocation (also possible using a callback function). On a Macintosh, code must take into account the extra level of indirection of the handle arguments.

Here is C code for an example user function that computes the inner product of two real vectors.

Non-Macintosh version:
C:

```
#include "Userfun.h" /*not needed here as coded*/

void goo(double * x, double * y, long * n, double * result)
{
    int         i;

    *result = 0.0;
    for (i = 0; i < *n; i++)
    {
        *result += x[i]*y[i];
    }
}
```

Fortran:

```
      subroutine goo(x, y, n, result)
      integer*4        n
      double precision x(n), y(n), result
      integer*4        i

      result = 0.0d0
      do 2 i = 1, n
        result = result + x(i)*y(i)
2     continue
      return
      end
```

In Windows, when compiling using Borland C/C++ 4.5, you need to replace "void goo" by "void _export goo".

Macintosh (both PPC and 68K) version:

```
#include "Userfun.h" /* required */
#define main_goo main /*entry must have internal name 'main'*/

void main_goo(double ** argx, double ** argy, long ** argn,
         double ** argresult)
{
    double     *x = *argx, *y = *argy, *z = *argz;
    long       *n = *argn;
    int         i;

    EnterCode();
```

```
    *result = 0.0;
    for (i = 0; i < *n; i++)
    {
        *result += x[i]*y[i];
    }
    ExitCode();
}

#ifdef powerc /*powerc defined means compiling for Power PC*/
 RoutineDescriptor goo =
    BUILD_ROUTINE_DESCRIPTOR(uppMainEntryProcInfo04,main_goo);
#endif /*powerc*/
```

The first executable statement in a 68K Macintosh user function must be EnterCodeResource(), and the last before the return must be ExitCodeResource(). EnterCode() and ExitCode() are C macros defined in Userfun.h that expand to these statements in a 68K Macintosh compilation and to nothing in a PPC, DOS, Windows, Motif, Unix or other compilation.

When coding for a PPC Macintosh, an additional statement declaring and initializing a RoutineDescriptor is required for each function. Constant uppMainEntryProcInfo04 is defined in dynload.h (automatically included by Userfun.h) and is appropriate for a function with 4 arguments.

Chapter 5

Search Key Tables

The tables in this Chapter relate the search keys used in each help file to the help topics that reference them.

Table 5.1: Search Keys and References from MacAnova.hlp

Search Key	References
ANOVA	anova, anovapred, cellstats, coefs, contrast, design, factor, fastanova, glm, glmfit, glmpred, glmtable, makefactor, manova, models, power, power2, predtable, regcoefs, resid, resvsindex, resvsrankits, resvsyhat, robust, samplesize, secoefs, wtanova, wtmanova, yates, yhat
Categorical Data	bin, glm, glmfit, glmpred, ipf, logistic, poisson, probit, tabs
CHARACTER Variables	array, cat, clipboard, compnames, delete, fromclip, inforead, ischar, makefactor, match, nameof, syntax, toclip, variables, varnames, vector
Combining Variables	array, cat, hconcat, makecols, makestr, matrix, rep, rotate, run, select, split, strconcat, structure, trilower, triunpack, triupper, vconcat, vector
Comparisons	contrast, cumdunnett, cumstudrng, invdunnett, invstu, invstudrng, t2val, tval, twotailt
Complex Arithmetic	cconj, cft, cimag, cmplx, complex, cpolar, cprdc, cprdcj, creal, crect, ctoh, hconj, hft, himag, hpolar, hprdh, hprdhj, hreal, hrect, htoc, polyroot, rft, unwind
Confidence Intervals	invchi, invF, invnor, invstu, invstudrng, regcoefs, secoefs, t2int, tint

Continued from previous page

Search Key	References
Control	arginfo fun, batch, break, breakall, breakif, callback fun, customize, else, elseif, evaluate, for, getoptions, if, loadUser, macro, macro syntax, macros, redo, REDO, setoptions, shell, syntax, User, user fun, while
Descriptive Statistics	boxplot, cellstats, cor, describe, halfnorm, hist, max, min, rankits, stemleaf, sum, t2int, t2val, tabs, tint, tval, twotailt, vboxplot
Files	adddatapath, addmacrofile, asciisave, batch, console, data files, fboxplot, fchplot, fcolplot, files, flineplot, fplot, fprint, frowplot, fwrite, getdata, getmacros, graph files, inforead, launching, loadUser, macro files, macroread, macrowrite, matprint, matread, matwrite, read, readcols, restore, save, spool, User, user fun, vecread, write
GLM	anova, anovapred, bcprd, bit operations, coefs, contrast, design, factor, fastanova, glm, glmfit, glmpred, glmtable, ipf, isfactor, logistic, makefactor, manova, modelinfo, models, modelvars, nbits, poisson, power, power2, predtable, probit, regcoefs, regpred, regress, regs, resid, resvsindex, resvsrankits, resvsyhat, robust, samplesize, screen, secoefs, swp, varnames, wtanova, wtmanova, wtregress, xrows, xvariables, yates, yhat
General	arginfo fun, asciisave, callback fun, copyright, customize, dos-windows, edit, evaluate, gethistory, getlabels, gettime, haslabels, help, isarray, ischar, isdefined, isfactor, isgraph, islogic, ismacro, ismatrix, ismissing, isnull, isreal, isscalar, isstruc, isvector, labels, launching, list, listbrief, loadUser, macintosh, macrousage, more, quitting, rename, restore, save, sethistory, shell, syntax, unix, usage, User, user fun, wx
Input	adddatapath, clipboard, console, data files, enter, enterchars, files, fromclip, getdata, getmacros, inforead, macro files, macroread, matread, read, readcols, vecread
LOGICAL Variables	alltrue, anytrue, islogic, logic, syntax, variables
NULL Variables	anymissing, cat, dim, hconcat, ismissing, isnull, length, ndims, save, syntax, vconcat, vector

Continued from previous page

Search Key	References
Macros	addmacrofile, evaluate, getmacros, isarray, ischar, isdefined, isfactor, isgraph, islogic, ismacro, ismatrix, ismissing, isnull, isreal, isscalar, isstruc, isvector, keyvalue, macro, macro files, macro syntax, macroread, macros, macrousage, macrowrite, read
Matrix Algebra	bcprd, cholesky, det, diag, dmat, eigen, eigenvals, matrices, matrix, outer, qr, releigen, releigenvals, svd, swp, toeplitz, trace, transpose, trideigen, trilower, triunpack, triupper
Missing Values	anymissing, arithmetic, bit operations, files, ismissing, logic, matprint, matread, paste, print, read, setoptions, syntax
Multivariate Analysis	cluster, glm, kmeans, manova, rotation
Operations	arithmetic, bit operations, logic, matrices, nbits, transpose
Ordering	grade, halfnorm, match, rank, rankits, sort, unique
Output	asciisave, clipboard, data files, error, files, fprint, fwrite, graph files, labels, macro files, macrowrite, matprint, matwrite, more, paste, print, putascii, setoptions, spool, toclip, write
Plotting	addchars, addlines, addpoints, addstrings, boxplot, chplot, colplot, fboxplot, fchplot, fcolplot, flineplot, fplot, frowplot, graphs, graph files, graph keys, graph ticks, hist, lineplot, plot, resvsindex, resvsrankits, resvsyhat, rowplot, showplot, stemleaf, tek, tekx, vboxplot, vt, vtx
Probabilities	cumbeta, cumbin, cumchi, cumdunnett, cumF, cumgamma, cumnor, cumpoi, cumstu, cumstudrng, invbeta, invchi, invdunnett, invF, invgamma, invnor, invstu, invstudrng, power, power2, samplesize, t2int, t2val, tint, tval, twotailt
Random Numbers	getseeds, invbeta, invchi, invF, invgamma, invstu, rbin, rnorm, rpoi, runi, setoptions, setseeds
Regression	coefs, design, glm, glmfit, glmpred, logistic, models, poisson, power2, probit, regcoefs, regpred, regress, regs, resid, resvsindex, resvsrankits, resvsyhat, robust, screen, secoefs, wtregress, yhat
Residuals	resid, resvsindex, resvsrankits, resvsyhat

Continued from previous page

Search Key	References
Structures	changestr, compnames, isstruc, makestr, ncomps, split, strconcat, structure, structures
Syntax	alltrue, anytrue, arithmetic, assignment, batch, break, breakall, breakif, clipboard, comments, else, elseif, evaluate, for, if, keyvalue, keywords, logic, macro, macro syntax, macros, structures, subscripts, syntax, variables, vectors, while
Time Series	autoreg, cconj, cft, cimag, cmplx, complex, convolve, cpolar, cprdc, cprdcj, creal, crect, ctoh, hconj, hft, himag, hpolar, hprdh, hprdhj, hreal, hrect, htoc, movavg, padto, partacf, polyroot, reverse, rft, rotate, time series, trideigen, unwind, yulewalker
Transformations	asLong, atan, boxcox, ceiling, floor, halfnorm, hypot, nbits, rankits, rational, round, transformations
Variables	array, asLong, cat, data files, delete, diag, dim, dmat, getdata, getlabels, haslabels, hconcat, isarray, ischar, isdefined, isfactor, isgraph, islogic, ismacro, ismatrix, ismissing, isnull, isreal, isscalar, isvector, labels, length, logic, match, matrices, matrix, nameof, ncols, ncomps, ndims, nrows, rep, run, save, select, structures, syntax, trilower, triunpack, triupper, unique, variables, varnames, vconcat, vector, vectors

Table 5.2: Search Keys and References from Design.hlp

Search Key	References
Aliasing	aliases2, aliases3, allaliases2, choosegen2, ffdesign2
Analysis	boxcoxvec, ems, mixed, pairedcomp, quadmax, randsign, randt, rscanon, varcomp
ANOVA	boxcoxvec, ems, mixed, pairedcomp, varcomp
Confounding	confound2, confound3
Design	aliases2, aliases3, allaliases2, choosegen2, confound2, confound3, ffdesign2
Factorial	aliases2, aliases3, allaliases2, choosegen2, confound2, confound3, ems, ffdesign2, mixed, varcomp
Permutation test	randsign, randt
Random effects	mixed, varcomp

Table 5.3: Search Keys and References from Userfun.hlp

Search Key	References
Coding	arginfo fun, c macros, callback fun, type codes, user fun
Compiling	compile dos, compile mac, compile unix, compile win
Executing	User
Loading	loadUser
Sample source	arginfo fun, c macros, callback fun, user fun
User functions	arginfo fun, c macros, callback fun, compile dos, compile mac, compile unix, compile win, loadUser, type codes, User, user fun