An Introduction to MacAnova

by
Christopher Bingham
Gary Oehlert

University of Minnesota
School of Statistics
Technical Report Number 600

Revised September 1995

Contents

An Introduction to MacAnova

1 Getting Started

1.1 What is MacAnova?

MacAnova is an interactive computer program for statistical analysis of data. It features powerful commands for such things as regression analysis, analysis of variance, multivariate analysis and time series analysis. Moreover, it is an excellent tool for more elementary analyses, including graphical displays and computation of summary statistics.

Many statistical programs are primarily menu driven, with the user selecting what to do almost exclusively by choosing items from a menu. Although this can provide for an easy interface, it can restrict possible analyses to the specific set built in to the program.

MacAnova, by contrast, is primarily command driven, although the Macintosh version makes some use of menus. To do most things, you must use the keyboard to type commands into a command window or screen. Because MacAnova has a very wide set of commands and functions, as well as a means to combine them to make new commands (so called macros), you are not limited to a predefined set of analyses.

One attractive feature of MacAnova is that you can easily directly translate many statistical formulas to a form that MacAnova recognizes. For example, to compute a sample mean of data named `x`, you can type `sum(x)/nrows(x)` or, if `n` has earlier been set to the sample size, you can type `sum(x)/n`.

You can easily make high quality scatter plots and other graphs, for example, as simply as typing `plot(x,y)`.

Although there is extensive built in help, MacAnova takes some getting used to. You need to have a certain minimum level of knowledge and practice before you can make *full* use of it. The purpose of this document is to introduce you to the most important features of MacAnova, illustrating them with examples. Probably a good way to proceed is to read this at the computer, trying out things as they are introduced and using the `help()` [1] command to get more detailed information on each command as it is introduced. Some general help topics you may find useful later, but probably not at the start, are `arithmetic`, `array`, `batch`, `comments`, `keywords`, `logic`, `macros`, `matrices`, `models`, `structures`, `subscripts`, `syntax`, `transformations`, and `vectors`. Help on the special features of the different versions may be found in topics `dos`, `launching`, `macintosh`, and `unix`. Yet more complete information is available in the *MacAnova Users Guide*, the most recent version of which (dated March 1993) is for MacAnova 3.11.

[1] The parentheses are part of the name of the command. When you use a command, you will usually have stuff inside the parentheses.

The most recent version of MacAnova is Version 3.35, the most recent release of which is September 1995. Although it has many new features that are not described in the MacAnova Users' Guide, almost everything in the manual is still appropriate, except for some of the information in the Appendices. File Changes.txt distributed with MacAnova summarizes the changes.

The most recent version of MacAnova is the September 1995 release of Version 3.35. Although it has many new features that are not described in the MacAnova Users Guide, almost everything in the manual is still appropriate, except for some of the information in the Appendices. File `Changes.txt` distributed with MacAnova summarizes the changes. MacAnova is available via anonymous ftp from `umnstat.stat.umn.edu` in the `pub/macanova` directory. At the University of Minnesota, MacAnova can also be purchased on disks at the campus bookstores. For information, contact

MacAnova Project
Department of Applied Statistics
University of Minnesota
352 Classroom Office Building
1994 Buford Avenue
St. Paul, MN 55108

MacAnova was originally designed for use on Macintosh computers and the Macintosh version is still the most convenient to use. Other versions, for DOS and Unix, have the same capabilities, but the interface is somewhat different. The principal differences are these:

(a) The Macintosh version makes some use of menus; other versions do not.

(b) On the Macintosh, what you type and MacAnova's responses go into an editable document in the *command window*. This can be scrolled back so you can see stuff that has disappeared off the top of the screen. It can also be saved on a hard or floppy disk for later printing or editing. On other computers, what scrolls off the screen is lost, although MacAnova pauses after every screenful so that you can read the output. You can write your input and output to disk using the `spool()` command (see Sec. 4.2 below).

(c) The Macintosh version has up to four windows for graphs and you can switch between them and the command window. In fact, you can have up to nine simultaneous command windows and switch between them. You can print any of the windows, graphic or command, or transfer their contents to a word processor. Non-Macintosh versions have a single graphics window whose contents are lost as soon as you switch back to command mode. There is no easy way to print graphs or import them into a word processor, although the protected mode DOS version allows you to save the contents of the graphics screen to disk.

All the examples here were done on a Macintosh, and the computer output, including high resolutions graphs, copied into a word processor document.

1.2 Launching Macanova

On a Macintosh, double click on the MacAnova Icon which looks like this .

On DOS, usually MacAnova will be installed as `MACANOVA.EXE` in its own directory, say `C:\MACANOVA`. If this is the current directory, or if it has been added to your "path" in the `AUTOEXEC.BAT` file, just type `macanova` at the DOS prompt to start. Otherwise, type `c:\macanova\macanova`, for example.

More details on launching MacAnova are in Appendix A (for Macintosh) and Appendix B (for DOS) and are given in `help` topic `launching`.

After launching, a start up message like the following appears

```
                         M A C A N O V A   3.35

  An Interactive Program for Statistical Analysis and Matrix Algebra
      For information on major features, type 'help(macanova3)'
     For information on linear models and GLM's, type 'help(glm)'
        For latest information on changes, type 'help(news)'
             Version of 08/29/95 (Mac with Co-Processor)
       Type 'help(copyright)' for copyright and warranty info
    Copyright 1994, 1995, Gary W. Oehlert and Christopher Bingham


Cmd>
```

"`Cmd>`" is the standard prompt requesting that you type a command. On the Macintosh, you can use the mouse to put the cursor anywhere in the window and type in whatever you want, but Macanova obeys only what you type after the prompt (actually 1 space after the prompt). With DOS you don't have any choice; you can only type after the prompt.

Commands are typed in as sequences of letters and symbols. You can use Delete or Backspace to correct mistakes. On the Macintosh, you can use arrow keys or the mouse to reposition the cursor to edit a command before executing. Until Return or Enter is pressed, a command is not executed and may be edited.

On the Macintosh, if the cursor is not at the very end of the command line, pressing Return will just split the line, but MacAnova won't do anything. However, no matter where the cursor is, pressing Enter works the same as moving the cursor to the end of the command line and then typing Return.

Any information typed after a `#` is considered to be a comment and is ignored by MacAnova. This feature is used in many examples below to explain what is being done.

Example

```
Cmd>   x <- cat(1.2, 3.5, 2.3) # entering 3 values of data<cr>
```

creates a variable named x with values $x_1 = 1.2$, $x_2 = 3.5$, and $x_3 = 2.3$. A convention that will be used throughout this document is to use italic *Courier* font for what is typed by the user and non-italic Courier font for what the computer prints. The <cr> indicates a Return or Enter and will not appear in later examples. Often, just as here, example lines will have descriptive comments starting with "#". Comments that are not part of the MacAnova session will be in **bold face**.

If you need to type a command that is longer than the screen or window width, you can just keep typing and it will wrap around to the next line. Alternatively, you can split it yourself at a convenient spot by typing a backslash "\" followed by Return. Thus , as long as you don't type Return at the end of the first line,

```
Cmd>  y < cat(113.7,91.4,89,133.3,90.6,87.4,96.8,78.4,81,113.9,
120,110, 92.7,131.5,100.9,120.5,87.3,97.9,83.2,81.2)
```

has the same effect as

```
Cmd>  y <- cat(113.7,91.4,89,133.3,90.6,87.4,96.8,78.4,81,113.9,\
 120,110,92.7,131.5,100.9,120.5,87.3,97.9,83.2,81.2)
```

where Return was typed after "\". To repeat, don't use Return without a backslash in the middle of a command, except inside quotes "..." or curly brackets { ...} .

Note: If you are using a DOS computer, the use of "\" to continue a line is recommended because DOS has an inherent limit on the length of any line that can be typed in.

1.3 Quitting

It is just as important to know how to stop MacAnova as to how to start it. On all versions of MacAnova you can stop by typing quit, end, stop or bye. On the Macintosh, selecting **Quit** on the **File** menu also ends the run, as does closing the command window when there is only one such window. When you leave MacAnova, all the data and results in MacAnova's memory (the so-called *workspace*) will disappear. You can save them on disk for later use by using commands save() or asciisave() before quitting (see Sec. 4.3 below). The Macintosh version asks if you want to save the workspace and the output window when you quit.

Example

```
Cmd>  quit # you could also use "bye", "stop" or "end"
```

1.4 Learning more about MacAnova

Your first resource is probably command help() (see Sec. 3.6 below). Help topics provide the most up-to-date information on functions, commands, and syntax. Get in the habit of using help().

As of now (September 1995) there is no comprehensive manual for MacAnova 3.35 or later, although one should be ready in the next couple of months. However, the *MacAnova Users Guide*, University of Minnesota School of Statistics Technical Report 493 as revised March 1993, is a complete reference for MacAnova 3.11. Virtually everything there is still applicable with the exception of the appendices. Of course, it does not include the features and enhancements introduced with version 3.35. File `changes.txt` distributed with MacAnova summarizes these changes.

2 The Basics

2.1 MacAnova as numeric calculator

By typing numbers and algebraic symbols you can use MacAnova as a simple calculator.

```
Cmd>  3 + 5  # space between items is o.k.; simply 3+5 is o.k.
(1)           8

Cmd>  2 + 1 0 # but not spaces between digits in a number
ERROR: problem with input near 2 + 1 0

Cmd>  4*7e3 ; 3^4 ; 14.5 %% 5 # expressions separated by ';'
(1)        28000     4 x 7000
(1)           81     3 to the 4th power
(1)          4.5     remainder when 14.5 is divided by 5

Cmd>  sqrt(20); sq rt(20) # space in name is an error
(1)       4.4721     √20           "sqrt" means square root
ERROR: problem with input near sqrt(20); sq rt

Cmd>  log(3)*(5+sqrt(6)) #combination of functions and arithmetic
(1)       8.1841     ln(3) x (5+√6)
```

These lines illustrate several features (we'll explain about the "`(1)`" in the output later).

- When you type a number or an expression at the `Cmd>` prompt you get immediate output.
- Spaces are generally ignored, except that you can't embed them in numbers or names.
- You can use parentheses to force addition or subtraction to be done before other operations.
- You can do several things on a single line, separating them by `;`. Each part can produce its own output on a new line as if they were typed after different prompts.

- You can compute things like square roots by typing their names with a number in parentheses. Among other so called transformations available are `exp()`, `cos()`, `sin()`, `tan()`, `log10()`, and `atanh()`. Type `help("transformations")` for a full list.
- You can enter numbers using computer scientific notation: `1.3e4` is 1.3×10^4, `3.231e-17` is 3.231×10^{-17}, etc.

The full set of arithmetic operators consists of "+", "–", "*" (multiplication), "/" (division), "^" (exponentiation), and "%%" (modular division: `14.5 %% 5` computes 4.5, the remainder of 14.5 when divided by 5).

MacAnova more or less follows the normal rules of algebra in evaluating expressions. For example `3 + 4*3` is evaluated as `3 + 12`, while `(3 + 4)*3` is evaluated as `7*3`. Exponentiation is a little tricky in that `2^4*3` is interpreted as `16*3`, while `2^(4*3)` is evaluated as `2^12`. Also `*` and `/` are evaluated step by step from left to right so that `3*4/5*6` is evaluated as `((3*4)/5)*6`.

2.2 MacAnova as symbolic calculator

Besides simple arithmetic such as `3+sqrt(2)` involving only numbers, you can create named *variables* with numerical values and do arithmetic and other computations on them as illustrated by the following **example**

```
Cmd>  x <- 10; a <- 1101.1; b <- -2 # store values in variables

Cmd>  a + b * x # use variables in an expression
(1)       1081.1
```

Here we have created variables `x`, `a`, and `b` with specific values and then used them in an algebraic formula or expression. These values remain available under these names until you change them, delete them, or you quit MacAnova. Expressions can be almost arbitrarily complicated and can contain transformations as well as other MacAnova functions.

The combined symbol "<–" (less than followed by minus) is the *assignment operator*. The value of the number or expression to the right of <– is saved in the variable to the left. Its use is analogous to the use of "=" in certain computer languages such as Fortran and C or to ":=" in Pascal. A very common mistake made by users with programming experience is to use "=" when you mean "<–".

A few variables such as `PI` and `E`, are *predefined* although you can change their values.

```
Cmd> PI; E # E is base of natural logarithms
(1)       3.1416
(1)       2.7183
```

The names of variables must begin with a letter, but subsequent characters may be numbers or letters or the underscore character "_". Variable names are case sensitive. This means that `residuals`, `Residuals`, and `RESIDUALS` are all different names. It is a good idea to avoid names with all capital letters, as MacAnova deletes and creates certain variables with all capital names (for example, `RESIDUALS`). You can get an alphabetized listing of all active variables using commands `listbrief()` or `list()`.

A variable's name may also start with the character "`@`" (as in `@mean`). Such a variable is called a *temporary variable* because it will be automatically deleted at the next `Cmd>` as in the following **example**

```
Cmd>  @xbar <- sum(x)/n; var <- sum((x-@xbar)^2)/(n-1)

Cmd>  @xbar # MacAnova has automatically deleted @xbar
UNDEFINED
```

Sometimes you may want to remove a variable from computer memory, perhaps because you have gotten the warning message

```
ERROR: not enough memory, try deleting variables
```

Command `delete()` does the trick.

```
Cmd>  delete(x, n) #deletes variables x and n

Cmd>  x # x is no longer defined
UNDEFINED
```

You can delete as many variables as you like in a single use of `delete()`.

2.3 MacAnova as computing language – functions and macros

In a technical sense, MacAnova commands are *functional*. Transformations such as `sqrt()` or `log10()` are particular cases of functions. When they are used they have inputs called *arguments* (in `log10(3.145)`, `3.145` is an argument), and they *return* (produce as output) results which can be saved in a variable, combined with other values in an expression, or printed.

Many functions have several arguments, separated by commas, the whole list being contained between parentheses as in the following **examples**

```
Cmd>  round(17/3,3) # round to three decimals
(1)        5.667

Cmd>  hypot(3,4) # computes sqrt(3^2+4^2)
(1)            5
```

Some functions just do things, but don't return any value that can be assigned to a variable. We usually call such functions simply *commands*. For example `print()` is a command that can be used to print several variables or expressions, perhaps with an increased number of significant digits, as in the following **example**.

```
Cmd>  w <- sqrt(10); w; print(nsig:12,w)
(1)       3.1623                 Output from w
w:
(1)       3.16227766017          Output from print()
```

The argument `nsig:12` for `print()` is an example of a *keyword phrase* of the form `name:value`, and is typical of the syntax used to control the behavior of many commands and functions. Because `print()` does not return a value, we get an error message if we try to use it as if it does:

```
Cmd>  z <- print(sqrt(3)) # erroneous try to assign print value
NUMBER:
(1)       1.7321                          Produced by print()
ERROR: assignment from null operation     Because of use of <-
```

A few functions or commands can be used with no arguments. They still need a pair of parenthesis, but with nothing between them. For **example**

```
Cmd>  listbrief() # list all MacAnova objects
boxcox      colplot     console     CONSOLE     DATAFILE    DEGPERRAD
DELTAT      E           fcolplot    frowplot    getdata     getmacros
MACROFILE   makecols    makefactor  model       PI          readcols
regs        resid       resvsindex  resvsrankits resvsyhat  rowplot
twotailt    yhat
```

Commands that do not return a value usually produce what we call *side effects*. For example, the side effects of `print()` are the printed values of its arguments and one side effect of `regress()` is a printed regression analysis. In addition, some commands like `regress()` create, as side effects, variables with standard names such as `SS`, `DF`, `RESIDUALS` or `COEF` containing quantities related to the analysis.

In addition to functions, MacAnova has what are known as *macros*. Their usage is almost indistinguishable from that of functions. To use a macro, you type its name followed by arguments separated by commas and enclosed in parentheses. Like functions, macros may return values or have side effects. Macros do differ from functions in some important ways, but most of the time you can treat them the same. Here is an example of the use of the useful macro `boxcox` which takes two arguments, here `cat(3.03,3.01,3.32,3.65,4.42)` (`cat()` will be explained later) and `.5`.

```
Cmd>  boxcox(cat(3.03,3.01,3.32,3.65,4.42),.5)
(1)     2.7514       2.73     3.0538     3.3822     4.0949
```

Probably the most important difference between functions and macros is their availability. A function can always be used. There is nothing you can do to delete it. Macros, on the other hand, are more like variables – they can be predefined, deleted, read in from a file, entered at the keyboard and printed. Predefined macros like `boxcox`, `getmacros`, `hist` and `resvsrankits` are automatically available. Others such as `covar` must be read from a file on a floppy or hard disk using `getmacros` or `macroread()`. When you have some experience with using MacAnova, you can create your own macros to extend the range of what MacAnova can do (see Sec. 7.7 of the Users Guide). File `MacAnova.mac`, usually in the same Folder or directory as MacAnova, contains dozens of macros you may find useful. You can get a list of all the macros already in MacAnova by command `listbrief(macros:T)`.

```
Cmd>  listbrief(macros:T)
boxcox      colplot     console     fcolplot   frowplot    getdata
getmacros   makecols    makefactor  model      readcols    regs
resid       resvsindex  resvsrankits resvsyhat rowplot     twotailt
yhat
```

You may already have spotted one of the conventions of this document – when a MacAnova function is referred to by name, it always has a pair or parentheses attached, as in `print()`. On the other hand, when a macro is referred to by name, just its name is given, as in `boxcox`. Whenever you use it, a function or a macro must be followed by parentheses enclosing the arguments, or if there are no arguments, by an empty pair of parentheses.

A function or macro returning a value can be used anywhere a simple number or variable name can be used – in expressions and as an argument to another function or macro. A simple example is `sqrt(a + 3*boxcox(x,.5))`.

2.4 More on Variables – REAL, LOGICAL, and CHARACTER data

Named variables can contain several types of data. The most common are numbers such as 2.4, -1, or 3.1478×10^{-8}. In MacAnova this type is called `REAL`. Almost as common are variables which have only two possible values, True and False. In MacAnova these are called `LOGICAL`, and True and False are typed or printed simply as `T` and `F`.

The values of some variables are sequences of characters. In MacAnova they are called `CHARACTER` variables. When typing `CHARACTER` data on the keyboard, the strings must be enclosed in double quotes, for **example**,

```
Cmd>  sentence <- "This is CHARACTER data"
```

In some MacAnova output, `LOGICAL` and `CHARACTER` are abbreviated to `LOGIC` and `CHAR`. Some more advanced data types are `STRUCTURE` and `GRAPH`.

Function `list()` may be used to list the names of variables, together with their data types and their dimensions, while `listbrief()` just lists their names. If you use one of the keyword phrases `real:T`, `char:T` or `logic:T` as an argument to `list()` or `listbrief()`, only variables of the specified type are listed.

Here are some **examples**.

```
Cmd>  Cmd> x <- 1/3; y <- T; z <- "Hi there!";print(x,y,z)
x:
(1)     0.33333                 REAL data
y:
(1)  T                          LOGICAL data
z:
(1)  "Hi there!"          CHARACTER data

Cmd>  list(x,y,z,PI,DATAFILE,boxcox)
boxcox          MACRO             Order is alphabetical
DATAFILE        CHAR   1
PI              REAL   1
x               REAL   1
y               LOGIC  1
z               CHAR   1

Cmd>  listbrief(x,y,z,PI,DATAFILE,boxcox)
boxcox      DATAFILE    PI          x           y           z

Cmd>  list(real:T)
data            REAL   8     2
DEGPERRAD       REAL   1
DELTAT          REAL   1
E               REAL   1
PI              REAL   1
x               REAL   1
```

As in this usage, `LOGICAL` values `T` and `F` are often used to represent yes and no in keyword phrases specifying command options. You'll see other examples as we go along.

`LOGICAL` values can even be used in arithmetic expressions and a few functions (but not as arguments to functions like `sqrt()` or `log()`), with True being treated as 1 and False as 0.

```
Cmd>  cat(T,F) + 3 # same as cat(1,0) + 3
(1)           4           3
```

Any character is allowed inside a `CHARACTER` variable, even a Return character. One common source of trouble is *forgetting to add the closing double quote to a character string* *and hitting* Return. You expect MacAnova to respond but it doesn't. Without the closing double quote, MacAnova is just waiting for you to put more characters inside the string variable. Type the closing double quote or nothing will ever happen. You may include a double quote in a `CHARACTER` variable by preceding it with a backslash.

```
Cmd>  print("Hello");print("\"Hello\"")
Hello
"Hello"
```

2.5 Comparisons of numbers and combining LOGICAL values

One place where `LOGICAL` values naturally arise is when you want to compare the value of one variable with another as in the following **example**

```
Cmd>  a <- 5; b <- 6; c <- 5 # assign values to a, b, and c

Cmd>  a < b; a > b; a == c # less than, greater than, equal to
(1) T        True because a is less than b
(1) F        False because a is not greater than b
(1) T        True because a equals c

Cmd>  cat(a >= b, a <= c, b != c) # a ≥ b, a ≤ a, b ≠ c
(1) F        T        T
```

Note that "==" means is equal to, ">=" and "<=" mean "greater than or equal to" and "less than or equal to", and "`!=`" means "not equal to". On a Macintosh you can use "≤", "≥" and "≠" instead of ">=", "<=" and "`!=`".

There are three logical operators, `!`, `&&` and `||`.

`!` is the *not* operator, changing True to False and vice versa.

```
Cmd>  3 <= cat(2,3);!(3 <= cat(2,3))
(1) F        T      3 ≤ 2 is False, 3 ≤  3 is True
(1) T        F      not(3 ≤ 2) is True, not(3 ≤ 3) is False
```

`&&` is the *and* operator. `c && d` has value True if and only if *both* `c` and `d` have value True:

```
Cmd>  cat(T && T, T && F, F && T, F && F)
(1) T        F        F        F
```

`||` is the *or* operator. `c || d` has value True if and only if either `c` or `d` or both have value True:

```
Cmd>  cat(T || T, T || F, F || T, F || F)
(1) T        T        T        F
```

2.6 Variables with several values – Vectors and Matrices

In most of the examples so far, a MacAnova variable contains only a single value, whether `REAL`, `LOGICAL`, or `CHARACTER`. Such variables are usually called *scalar* variables or simply *scalars*. Obviously, this does not get you very far in statistics. What we need are variables that can contain several numbers, perhaps an entire sample of data. The simplest such variable in MacAnova is a *vector*, a variable containing several numbers, several True/False values, or several character strings.

The basic function for creating a vector is `cat()`, which can have any number of arguments, all of the same type. There have been a number of usages of `cat()` presented without comment above. Consider the following simple **examples**.

```
Cmd>  x<-cat(33.5, 27.3, 36.7, 30.5); x# 3.10 in Devore & Peck
(1)         33.5         27.3         36.7         30.5

Cmd>  w <- cat(T,T,F,T,T,T,F,F,T,T); w # 3.12 in D&P
 (1) T      T      F      T      T      T      F      F      T
(10) T                        T = Success, F = Failure

Cmd>  dakotas <- cat("South Dakota","North Dakota"); dakotas
(1) "South Dakota"
(2) "North Dakota"
```

You extract elements of a vector using *subscripts*, values enclosed in square brackets `[...]`, as in the following **examples**.

```
Cmd>  x[3]; w[7]; j <- 2; dakotas[j]
(1)         36.7
(1) F
(1) "North Dakota"

Cmd>  print(x[cat(3,4)], x[cat(-1, -2)], x[cat(F,F,T,T)])
VECTOR:
(1)         36.7         30.5
VECTOR:
(1)         36.7         30.5
VECTOR:
(1)         36.7         30.5
```

As the first example shows, subscripts can be variables as well as numbers. The second example shows that you can extract several values at once, in several different ways.

If a subscript is itself a *vector of positive integers* such as `cat(3,4)`, the corresponding elements of the variable are extracted as if you had typed, say, `cat(x[3],x[4])`. There can even be duplicates in the subscript vector – `x[cat(1,1,3)]` is the same as `cat(x[1],x[1],x[3])`.

If the subscript is a *vector of negative integers*, you extract all the *other* elements. Thus `x[cat(-1, -2)]` extracts all the elements of `x` except the first and the second. You cannot have duplicate negative subscripts nor can you mix them with positive subscripts.

If the subscript is a *LOGICAL vector*, the values whose positions correspond to True are extracted and the remaining values omitted. A LOGICAL subscript vector must be the same length as the vector it is subscripting.

Sometimes a data set consists of several variables. It is often convenient to combine them in a single MacAnova variable in a sort of table form, with rows corresponding to different cases and columns corresponding to variables. Such a rectangular array is called a *matrix*. We'll see shortly how to create a matrix. In the meantime here is a very simple example of a MacAnova matrix. Its name is prob_1 and it contains data from Problem 1 in Chapter 4 of Devore and Peck.

```
Cmd>  prob_1 # print out the matrix
(1,1)         0.34         2.8
(2,1)         0.35         1.9
(3,1)         0.39         3.3
(4,1)         0.39         5.6
(5,1)         0.41         4.2
(6,1)         0.41         5.6
(7,1)         0.49         4.2
(8,1)         0.68         7.9
```

There are 8 observations on bivariate data and we require *two* subscripts to extract a particular value. The numbers in parentheses (for example (3,1)) are the row and column number of the first value in the line.

```
Cmd>  prob_1[3,2] # row 3 and column 2
(1,1)          3.3

Cmd>  prob_1[,1] # all of column 1
(1,1)         0.34
(2,1)         0.35
(3,1)         0.39
(4,1)         0.39
(5,1)         0.41
(6,1)         0.41
(7,1)         0.49
(8,1)         0.68

Cmd>  prob_1[4,cat(1,2)];prob_1[4,]# 2 ways to extract of row 4
(1,1)         0.39         5.6
(1,1)         0.39         5.6
```

As you can see, an empty subscript selects all the values of that subscript. You can also use negative subscripts or LOGICAL vectors as subscripts:

```
Cmd>  prob_1[cat(F,F,F,T,F,F,F,F),];prob_1[-cat(1,2,3,5,6,7,8),]
(1,1)          0.39          5.6
(1,1)          0.39          5.6
```

You can create a matrix from a vector using function `matrix()`. For example, to create matrix `prob_1` at the keyboard you can type

```
Cmd>  prob_1<-matrix(cat(.34,.35,.39,.39,.41,.41,.49,.68,\
 2.8,1.9,3.3,5.6,4.2,5.6,4.2,7.9), 8)
```

The general form is `matrix(v, nr)`, where `v` is a vector and `nr` is the number of rows. It is an error if the length of `v` is not divisible by `nr`. The values in `v` are entered into the result column by column.

You would create vectors corresponding to the columns of `prob_1` by

```
Cmd>  x <- cat(prob_1[,1]); y <- cat(prob_1[,2]); print(x,y)
x:
(1)         0.34          0.35          0.39          0.39          0.41
(6)         0.41          0.49          0.68
y:
(1)          2.8           1.9           3.3           5.6           4.2
(6)          5.6           4.2           7.9
```

Here `cat()` has changed each column to a vector with only 1 subscript. The numbers in parentheses (`(1)` and `(6)`) are the subscripts of the first number on that line. Thus `x[6]` has value 0.41.

The comparison operators introduced in Sec. 2.5 can be used to compute `LOGICAL` vectors which can be used as subscripts. For example, suppose you want to select only the elements `y` of corresponding to values of $x > .35$ and $x \leq .45$. You can use comparison operators and the `&&` operator.

```
Cmd>  y[x > .35 && x <= .45] # same as y[cat(F,F,T,T,T,T,F,F)]
(1)          3.3           5.6           4.2           5.6
```

2.7 Missing values

Missing values are a common occurrence when you are dealing with real data. You may have measurements of both the height and weight of each of 19 people, but for one reason or another, for two other people, you have only their weights so that two heights are missing. Or, when entering data for computer analysis, you may notice a value is impossible (for example, a human height of 61 feet). MacAnova has a special value, `MISSING`, that can be used to replace such values. When typed from the keyboard, the code for `MISSING` is a question mark, "`?`", but printed, it prints as

```
Cmd>  a <- cat(3,?,-7,?,4); a
(1)           3     MISSING          -7     MISSING            4
```

Many commands do something fairly sensible with values of MISSING. A few give you a warning:

```
Cmd>  5 + a
WARNING: arithmetic with missing values(s)
(1)           8     MISSING          -2     MISSING            9
```

This illustrates that a value of MISSING combined arithmetically with anything else gives a missing value.

LOGICAL values of MISSING are also possible, although the only way to create them is by a comparison of a number with MISSING:

```
Cmd>  a > 0
WARNING: arithmetic comparison with missing values(s)
(1) T       MISSING F       MISSING T
```

Function ismissing() can be used to find which values in a REAL or LOGICAL data set are MISSING. ismissing(x) returns a LOGICAL variable with the same size and shape (dimensions) as x whose values are True where x is MISSING, and False where x is not missing.

```
Cmd>  ismissing(cat(1,?,3))
(1) F       T       F
```

3 Building on the Basics

3.1 Combining vectors and matrices – cat(), hconcat() and vconcat()

We learned above in Sec. 2.6 how to use cat() to create vectors from several individual values. You can also use cat() to combine several vectors to make a longer vector.

```
Cmd>  a1 <- cat(1,3,5); a2 <- cat(6,7,8); a3 <- 10

Cmd>  cat(a1,a2,a3) # make longer vector like cat(1,3,5,6,7,8,10)
(1)           1           3           5           6            7
(6)           8          10
```

Function cat() can also be used to change a matrix to a vector.

```
Cmd>  cat(prob_1) # prob_1 is
the 8 by 2 matrix used before
 (1)        0.34        0.35        0.39        0.39        0.41
 (6)        0.41        0.49        0.68         2.8         1.9
(11)         3.3         5.6         4.2         5.6         4.2
(16)         7.9
```

First we get all the values in column 1 of `data`, followed by the values in column 2.

Sometimes you will want to make a larger matrix by combining together *side by side* two or more vectors or matrices. Obviously all the pieces must have the same number of rows. For example, suppose you want to put vectors `x` and `y` back together in a matrix, together with a column containing the case numbers.

```
Cmd>  data1<- hconcat(cat(1,2,3,4,5,6,7,8),x,y); data1
(1,1)          1        0.34         2.8
(2,1)          2        0.35         1.9
(3,1)          3        0.39         3.3
(4,1)          4        0.39         5.6
(5,1)          5        0.41         4.2
(6,1)          6        0.41         5.6
(7,1)          7        0.49         4.2
(8,1)          8        0.68         7.9
```

This has produced the 8 by 3 matrix `data1`. The general usage of `hconcat()` is `hconcat(a,b,c,...)`, where each argument is a vector or matrix with the same number of rows.

If you want to combine two or more matrices all with the same number of columns by stacking them *one above the other*, you can use `vconcat()`:

```
Cmd>  vconcat(data[cat(5,6,7,8),], data[cat(1,2,3,4),])
(1,1)        0.41         4.2
(2,1)        0.41         5.6
(3,1)        0.49         4.2
(4,1)        0.68         7.9
(5,1)        0.34         2.8
(6,1)        0.35         1.9
(7,1)        0.39         3.3
(8,1)        0.39         5.6
```

This reorders the rows of `data`, putting the rows 5 through 8 ahead of rows 1 through 4.

3.2 Creating patterned vectors – run() and rep()

Suppose you want a vector with values 1, 2, 3, 4, and 5. You learned in Sec. 2.6 that this could be done by `y <- cat(1,2,3,4,5)`. You could enter a vector with values 1, 2, 3, ..., 100 the same way but it would be tedious and easy to make a mistake. Function `run()` provides a short cut. Here is the simplest usage of `run()`:

```
Cmd>  run(8) # produce vector with values 1, 2, 3, ..., 8
(1)          1          2          3          4          5
(6)          6          7          8
```

Here are other, fairly self explanatory, uses of `run()`:

```
Cmd>  run(3,11) # values from 3 to 11
(1)          3          4          5          6          7
(6)          8          9         10         11

Cmd>  run(3.5,5,.5) # values from 3.5 to 5.0 stepping by 0.5
(1)        3.5          4        4.5          5

Cmd>  run(5,3.5,-.5) # backwards from 5.0 to 3.5, stepping by -.5
(1)          5        4.5          4        3.5
```

Sometimes you may want a vector consisting of all 1's or some other value. You could use `cat(1,1,1,1,1,1,1)`, say, to get seven 1s, but function `rep()` is easier.

```
Cmd>  rep(1,7) # vector of 7 1's
(1)          1          1          1          1          1
(6)          1          1
```

Instead of repeating a single number, you can repeat a vector:

```
Cmd>  rep(run(4),2) # like cat(run(4),run(4))
(1)          1          2          3          4          1
(6)          2          3          4
```

There is a more elaborate use of `rep()` that is sometimes useful:

```
Cmd>  rep(run(3), rep(2,3)) # same as rep(run(3), cat(2,2,2))
(1)          1          1          2          2          3
(6)          3
```

Each of the numbers in `run(3)` is repeated 2 times. The second argument of `rep()` here is the same length as the first and provides the number of repetitions. Here is an even fancier usage:

```
Cmd>  rep(run(3),cat(2,3,4))
(1)          1          1          2          2          2
(6)          3          3          3          3
```

This repeats 1 two times, 2 three times, and 3 four times.

3.3 Assigning values to the elements of a vector or matrix

If you want to change one or a few elements in a vector or matrix, you can assign new values directly using subscripts. For example, suppose you learned the value in row 8 of column 2 of the matrix `prob_1` was incorrect and that the correct value should have been .58 instead of .69. You could make the change by

```
Cmd>  prob_1[8,1] <- .58; prob_1
(1,1)        0.34        2.8
(2,1)        0.35        1.9
(3,1)        0.39        3.3
(4,1)        0.39        5.6
(5,1)        0.41        4.2
(6,1)        0.41        5.6
(7,1)        0.49        4.2
(8,1)        0.58        7.9
```

You would make the same change in vector `x` by

```
Cmd>  x[8] <- .58; x
(1)        0.34        0.35        0.39        0.39        0.41
(6)        0.41        0.49        0.58
```

You can change several values at once:

```
Cmd>  w <- run(5); w
(1)           1           2           3           4           5

Cmd>  w[run(2)] <- cat(-10, ?); w[-run(3)] <- 17; w
(1)         -10     MISSING           3          17          17
```

In this last command line, we first replace elements 1 and 2 (`run(2)`) of `w` by -10 and a missing value, and then replace elements 4 and 5 (all except elements 1, 2, and 3) by 17. For future use we change back `prob_1[8,1]` and `x[8]` to their original values.

```
Cmd>  prob_1[8,1] <- .68; x[8] <- .68
```

3.4 Simple summaries of data in vectors and matrices – sum(), prod(), min(), max(), sort() and rank()

Functions `sum()`, `prod()`, `min()`, and `max()` compute the sum of, the product of, the minimum of, and the maximum of the values in a vector.

```
Cmd>  cat(sum(y), prod(y), min(y), max(y))
(1)        35.5       76723         1.9         7.9
```

For comparison, here are the sum and product as computed “by hand”:

```
Cmd>  cat(2.8 + 1.9 + 3.3 + 5.6 + 4.2 + 5.6 + 4.2 + 7.9,\
 2.8 * 1.9 * 3.3 * 5.6 * 4.2 * 5.6 * 4.2 * 7.9)
(1)        35.5       76723
```

Function `sum()` is useful for computing means, variances and standard deviations from the basic formulas $\bar{y} = \sum_{i=1}^{n} y_i/n$ and $s_y^2 = \left(\sum_{i=1}^{n}(y_i - \bar{y})^2\right)/(n-1)$.

```
Cmd>  n <- nrows(y); ybar <- sum(y)/n # mean
Cmd>  yvar <- sum((y- ybar)^2)/(n-1) # variance

Cmd>  cat(ybar, yvar) # mean and variance
(1)      4.4375      3.6027
```

You can compute the range () by

```
Cmd>  max(y) - min(y)
(1)           6
```

Function `sort()` allows you to reorder the elements in a vector in increasing order:

```
Cmd>  yup <- sort(y); ydown <- sort(y,down:T); hconcat(yup,ydown)
(1,1)          1.9          7.9      Col. 1 is y sorted up
(2,1)          2.8          5.6      Col. 2 is y sorted down
(3,1)          3.3          5.6
(4,1)          4.2          4.2
(5,1)          4.2          4.2
(6,1)          5.6          3.3
(7,1)          5.6          2.8
(8,1)          7.9          1.9
```

Of course, these also give you another way to compute the minimum and the maximum:

```
Cmd>  cat(sort(y)[1], sort(y,down:T)[1])
(1)         1.9          7.9
```

This last also shows that you can use subscripts directly on the values of functions.

One important way of summarizing a vector of numbers, is to compute the *ranks* within the vector of each of the elements, with the smallest value getting rank 1, the next smallest getting rank 2, etc. You can do this in MacAnova using function `rank()`:

```
Cmd>  a <- cat( 14.7, 13.1, 16.6, 12.9, 12.5); rank(a)
(1)           4            3            5            2            1
```

Since 14.7 is the 4th number in order of size (there are 4-1 = 3 numbers less than 14.7) it gets rank 4, etc. When there are ties, the ranks of the tied values are the averages of what would be the ranks of any tied values if they were very slightly changed so as to break the tie:

```
Cmd>  b <- cat( 14.7, 13.1, 16.6, 12.5, 12.5); rank(b)
(1)          4           3           5         1.5         1.5
```

If the second 12.5 were modified, say to 12.500001, the two last ranks would be 1 and 2 which average to 1.5.

All these functions can have matrices as arguments. In the case of `sum()`,`prod()`,`min()`, and `max()` the result is a row vector, a matrix with only one row, containing the result of applying the function to each column. Both `sort()` and `rank()` compute a new matrix, the same size as the argument, with the ordered values or ranks of each column. Let's apply these to the matrix `prob_1`:

```
Cmd>  sum(prob_1); prod(prob_1); min(prob_1); max(prob_1)
(1,1)        3.46        35.5    Sums
(1,1)   0.0010138       76723    Products
(1,1)        0.34         1.9    Minima
(1,1)        0.68         7.9    Maxima

Cmd>  hconcat(sort(data), rank(data))
(1,1)        0.34         1.9           1           2
(2,1)        0.35         2.8           2           1
(3,1)        0.39         3.3         3.5           3
(4,1)        0.39         4.2         3.5         6.5
(5,1)        0.41         4.2         5.5         4.5
(6,1)        0.41         5.6         5.5         6.5
(7,1)        0.49         5.6           7         4.5
(8,1)        0.68         7.9           8           8
```

In the last, the first two columns are the ordered values in each column of `data`, and the last two columns are the ranks of each column of `data`.

Other descriptive descriptive statistics that are computed from the ordered values in a sample are the lower and upper (first and third) quartiles and the median (second quartile). The median is the central value in order of size if n is odd, and is the average of the two central values if n is even. A common way to define the lower and upper quartiles is the medians of the lower and upper halves of the data, putting the middle value in both halves if n is odd. As an example we compute the 3 quartiles of `y`, as well as the inter-quartile range (IQR), a measure of the spread or dispersion of the sample:

```
Cmd>  q1 <- sum(sort(y)[cat(2,3)])/2 # lower quartile

Cmd>  q2 <- sum(sort(y)[cat(4,5)])/2 # median

Cmd>  q3 <- sum(sort(y)[cat(6,7)])/2 # upper quartile
```

```
Cmd>  cat(q1,q2,q3)
(1)         3.05          4.2          5.6

Cmd>  iqr <- q3 - q1; iqr # Interquartile range
(1)         2.55
```

3.5 Simple descriptive statistics – describe()

In the previous section you learned how to compute some basic descriptive statistics from their definitions using basic MacAnova functions such as `sum()`, `max()` and `min()`. In fact, if you try hard enough, practically any statistical analysis that you might do using MacAnova can be accomplished this way. But, as promised at the outset, for many analyses there are *built-in* commands and functions that do complete analyses without you having to know computing formulas.

Function `describe()` enables you to compute in one line almost all the descriptive summaries we have seen above. It is best introduced by an **example**:

```
Cmd>  describe(y) # y is the second column of data from D&P 4.1
component: n
(1)            8          Sample size
component: min
(1)          1.9          Minimum
component: q1
(1)         3.05          Lower quartile
component: median
(1)          4.2          Median or 2nd quartile
component: q3
(1)          5.6          Upper quartile
component: max
(1)          7.9          Maximum
component: mean
(1)       4.4375          Mean (average)
component: var
(1)       3.6027          Variance
```

As printed output this is pretty self-explanatory. It looks as if it might be a side-effect of `describe()`. However, what is printed is actually the *value* that `describe()` returns. The value can, in fact be saved in a variable.

```
Cmd>  results <- describe(y); results$mean; results$var
(1)       4.4375      Compare with mean above
(1)       3.6027      Compare with variance above
```

Variable `results` is an example of a new type of variable called a *structure* (sometimes abbreviated `STRUC` in MacAnova).

```
Cmd>  list(results)
results          STRUC  8
```

Structures are made up of several named *components*. Here there are eight of them, `n`, `min`, `q1`, `median`, `q3`, `max`, `mean`, and `var`. Individual components can be extracted as shown, by appending `$cname` to the name of the structure, where `cname` is the component name. You can find all names of all the components in a structure by function `compnames()`.

```
Cmd>  compnames(results)
(1) "n"
(2) "min"
(3) "q1"
(4) "median"
(5) "q3"
(6) "max"
(7) "mean"
(8) "var"
```

If all you want is one summary value, say the median, you can follow this **example**:

```
Cmd>  describe(y)$median # or describe(y,median:T)
(1)           4.2
```

or to compute just the mean and variance

```
Cmd>  describe(y,mean:T,var:T)
component: mean
(1)        4.4375
component: var
(1)        3.6027
```

You can use `describe()` with a matrix argument, too:

```
Cmd>  describe(data)
component: n
(1)             8            8
component: min
(1)          0.34          1.9
component: q1
(1)          0.37         3.05
component: median
(1)           0.4          4.2
component: q3
(1)          0.45          5.6
component: max
(1)          0.68          7.9
component: mean
(1)        0.4325       4.4375
component: var
(1)      0.012079       3.6027
```

Each component is now a vector with one value for each column of the argument matrix.

Several other MacAnova functions, including `split()`, `coefs()`, `secoefs()`, `cellstats()`, and `regpred()`, compute structures as their values. See Sec. 7.2 in the Users' Guide for information about functions `makestr()`, `strconcat()`, and `changestr()` which allow you to create or modify structures directly.

You can use structures are arguments to many MacAnova functions, including `describe()`, `sum()`, `max()`, and `min()`. The result is a structure with information on each of the components of the argument. For instance, since `makestr(x,y)` creates a structure with components `x` and `y`, `describe(makestr(x,y))` computes statistics on both `x` and `y` in one command. See Sec. 6.4 for an example.

3.6 Getting help – MacAnova command help()

If you need to refresh your memory about any function or command or about general topics like syntax, you can use MacAnova's `help()` command. Suppose you wanted more detail on the function `round()` used in one of the examples above.

```
Cmd>  help(round)
round(x) rounds the elements of the numerical 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 is a positive integer, this rounds to
n decimal places.

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.

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

Typically, as here, the first line gives the most standard usage, with more complex usages given later. Often, as in the last line, there are cross references to related topics.

There are hundreds of help topics. Here is how you get a list of all the topics:

```
Cmd>  help() # help() with no argument
Help is available on the following topics:
addchars     cumnor       if           nbits        shell
addlines     cumpoi       inforead     ncols        showplot
addpoints    cumstu       invbeta      ncomps       solve
addstrings   customize    invchi       ndims        sort
anova        delete       invF         news         split
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

On the Macintosh, selecting **Help** from the menu or pressing ⌘-**H** is equivalent to typing `help()`. See Appendix A.

The topics available include all the commands and functions, plus some more general topics such as `matrices`, `subscripts`, and `transformations`. To get help on a particular topic, use `help()` with one or more topics as arguments. You have to quote any topic names longer than 12 characters (for example, `"transformations"`), or the names of syntactic elements (`while`, `for`, `break`, `breakall`, `if`, `else`, `elseif`). On the Macintosh, if you use the mouse to select a topic in the command window, **Help** from the menu or ⌘-**H** gets help on the selected topic. Chapter 8 of the Users' Guide largely overlaps the help information. Command `help()` reads the information it prints from a file, `MacAnova.hlp`.

Sometimes you may remember only part of the name of a topic, but not the full name. You can use the wild card character `*` in a string to get a list of topics matching certain patterns. The three types of patterns recognized are `"part*"`, `"*part"`, and `"*part*"` which match topic names starting with, ending with, or containing part.

```
Cmd>  help("res*") # find all topics starting with "res"
resid         restore       resvsindex    resvsrankits resvsyhat

Cmd>  help("*plot*") # find all topics containing "plot"
boxplot    colplot    fchplot    flineplot frowplot  plot     showplot
chplot     fboxplot   fcolplot   fplot     lineplot  rowplot
```

Especially when you're new to MacAnova, you may not even know enough commands to use intelligently the pattern matching just described. All you know is you want to compute some descriptive statistics, or make a graph, or do some sort of residual analysis. You can look for help topics by keys. Here is an example:

```
Cmd>  help(key:"residuals") # find some topics about residuals
The following help topics concern Residuals
resid         resvsindex    resvsrankits resvsyhat
For help on topic foo, enter help(foo) or help("foo")
```

MacAnova found four topics. You could now get help on topic `resid`, say, by typing `help(resid)`. If you have no idea of the keys available, just do this:

```
Cmd>  help(key:"?")
Type 'help(key:"heading")', where heading is in following list:
ANOVA                   Input                   Probabilities
Categorical Data        LOGICAL Variables       Random Numbers
CHARACTER Variables     Macros                  Regression
Combining Variables     Matrix Algebra          Residuals
Complex Arithmetic      Missing Values          Structures
Control                 Multivariate Analysis   Syntax
Descriptive Statistics  Operations              Time Series
Files                   Ordering                Transformations
GLM                     Output                  Variables
General                 Plotting
```

When you specify a key, you only need as many letters as will make it unique. This `help(key:"des")` is as good as `help(key:"descriptive statistics")` (upper and lower case doesn't matter here).

If you learn how to use `help()`, it isn't that important to have a manual, since most of the details of commands are summarized in their help entries.

4 Using files

4.1 General

Although you can do a lot of work without using any of the commands that have to do with external (disk) files, commands that read or write files add a lot of capability to MacAnova. You can record your session in a file on disk using `spool()`, save all your variables in a file using `save()` and `asciisave()`, read data from a file using `vecread()` or `matread()`, and write data and results to a file using `print()` and `matwrite()`.

In all these commands, you need to specify a file name in quotation marks, for example, `matread("mydata.dat")`. MacAnova has no special conventions for file names as long as they are legal for the system on which it is running. File names used in examples below, are just that, examples. There is no need to use the same names, or end them the same same way (like `.dat`).

Things are a little easier on a Macintosh, since you can always use the null file name `""` (two double quotes with nothing between them). This brings up the usual Macintosh scrolling dialog box to let you select or specify a file.

On most commands *except* `save()` and `asciisave()`, if a file you are writing already exists, whatever you write to the file goes at the end of the information there, leaving what is there intact. If that is not what you want, use the keyword phrase `new:T` as an argument, as in `matwrite("mydata",x,new:T)`. (On a Macintosh, when you save the command window, it always as if you specified `new:T`.)

4.2 Recording your MacAnova session – spool()

Often when you use MacAnova you want some sort of record of what you have done, or at least a permanent copy of those answers you want. Command `spool()` makes this quite easy in MacAnova – as long as you remember to use it. It works on all computer systems, Macintosh, DOS or Unix.

Command `spool()` keeps a record of some or all of your MacAnova session on a hard disk or floppy. When as you want to start saving stuff, you need to type something like

```
Cmd>  spool("spool.txt") # start spooling on file spool.txt
```

This starts recording (spooling) on file `spool.txt`. From now on, everything you type and everything MacAnova replies except high resolution plots will be written in the file. Of course, this includes any mistakes you make and your false starts. On a Macintosh, you don't need to type the file name but can type

```
Cmd>  spool("") # Null file name allowed only on Macintosh
```

If at some point, you want to suspend spooling, simply type

```
Cmd>  spool() # with no file name
Spooling on spool.txt suspended
```

When you want to start recording again, you can again type

```
Cmd>  spool() # again with no file name
Resume spooling on spool.txt
```

Effectively, once you have started spooling, `spool()`, with no argument, toggles it off and on.

When you are done with your session, you can read the spool file into almost any word processor or text editor, edit out what you don't want to keep and add your commentary. One tip – if your word processor allows a choice of fonts, use a font that has equal width characters such as Courier or Monaco. If you don't, things that lined up on the computer screen will not line up in your document and may be hard to read.

On a Macintosh the use of `spool()` is less necessary since all your input and MacAnova's output remain in the command window. At any time you can select **Save Window (⌘-S)** or **Save Window As...** on the **File** menu and the entire contents of the window are written to disk for later editing. After the first time, the name of the file becomes the window title and you can re-save the window just by pressing **⌘-S**. When you quit, you are given a chance to save the command window on disk. A dialog box asks **Save Changes to Window "xxxxx" before closing?** To save, just click on the **OK** button (or the **Don't Save** button if you don't want to).

You can also use commands `print(file:"fileName",...)` and `write(file:"fileName",...)` to print the values of individual variables in a file. See the Users' Guide or type `help(print,write)`.

4.3 Saving what you've done – save(), and asciisave()

Sometimes you may not have time to do everything you want or need to in one session, and still have more to do when you have to quit. Commands `save()` and `asciisave()` are specifically designed to help you in this situation. They allow you to save all the current variables and macros in a file so that they can be restored (by command `restore()`) at the start of a later MacAnova session. They do *not* save your commands and output. Use `spool()` for that.

Here is a typical use of `save()`:

```
Cmd>  save("savework.sav")
Workspace saved on file savework.sav
```

You can now quit MacAnova. Later, when you restart, everything can be restored to the way it was by

```
Cmd>  restore("savework.sav") # restore("") on Macintosh
Restoring workspace from file savework.sav
Workspace saved Fri Aug 25 16:00:03 1995
```

On a Macintosh, instead of `save()` you can use either **Save Workspace (⌘-K)** or **Save Workspace As...** on the **File** menu. When you next start up you can just double click on the saved file and MacAnova will be launched with everything restored. At the same time as saving your workspace on a Macintosh, you may also want to save the command window using items **Save Window (⌘-S)**or **Save Window As...** on the **File** menu.

If you use MacAnova on more than one type of computer, you might like to start on one computer, say a Macintosh, save your work, and then finish it later on another type, say a PC compatible computer. The file produced by `save()` isn't any good for this. Instead, use command `asciisave()`:

```
Cmd>  asciisave("savework.asc")
Workspace saved on file savework.asc
```

This produces an ordinary text file which can be restored by MacAnova on any computer (by `restore("savefile.asc")`) or even sent via E-mail.

Command `save()` can also be a life saver if your computer is unstable and prone to crashing. If you save your work from time to time, or if, perish the thought, your copy of MacAnova has a bug that causes occasional crashes. If you save your work from time to time, you never will lose much if the computer or program goes down. After using `save("savework.sav")` or `asciisave("savework.sav")` once, you can update the save file simply by `save()` or `asciisave()`, without specifying a file name. On a Macintosh, simply press **⌘-K**.

4.4 Reading data from files – functions vecread() and matread() and macro readcols

Except when analyzing small amounts of data that can easily be typed in at the keyboard, you will probably want to work with data that are in a file on disk. The data may have been provided by someone else, entered by you using a word processor or editor or possibly exported from a spreadsheet. The data file must be a text file (see below).

The simplest type of data file readable by MacAnova consists of just numbers. Here is a listing of file `rabbit.dat` containing 12 determinations of the survival time in minutes under anaerobic conditions of certain rabbit nerves:

```
16.2  22.5  21.4  19.6  24.8  21.4
19.0  14.7  13.3  23.0  16.8  20.1
```

You can read these data into MacAnova by

```
Cmd>  x<-vecread("rabbit.dat" )# form is vecread(fileName)
Cmd>  x # print out what you've read in
 (1)          16.2         22.5         21.4         19.6         24.8
 (6)          21.4           19         14.7         13.3           23
(11)          16.8         20.1
```

Function `vecread()` reads the numbers in the file sequentially, line by line to the end, and returns them as a vector. A question mark (`?`) in the file is interpreted as `MISSING`. If `vecread()` finds anything it can't read, it skips it, printing a warning message the first time something unreadable is hit. The one exception to this is that an exclamation point "`!`" in the file stops the reading right there.

Suppose `rabbit.dat` actually looked like this:

```
 Data on twelve rabbit nerves
16.2  22.5  21.4  19.6  24.8  21.4
19.0  14.7  13.3  23.0  16.8  20.1
```

Then, after `x<-vecread("rabbit.dat")`, Macanova will read the data correctly but will also print out:

```
WARNING: nonnumeric character(s) ignored on standard input
```

You would be in trouble if the first line read `Data on 12 rabbit nerves` because the `12` would be read as a number and vecread() would return 13 numbers, the `12` followed by the actual data. So, to be on the safe side, a file to be read by `vecread()` should contain only numbers.

The most common type of data file consists of several columns, with each column corresponding to a variable and each line to the data for a case. Consider the following file `crops.dat`:

```
1926  20.1   7.2
1927  23.6   7.1
```

```
 1928  26.3   7.4
 1929  19.9   6.1
 1930  16.7   6.0
 1931  23.2   7.3
 1932  31.4   9.4
 1933  33.5   9.2
 1934  28.2   8.8
 1935  35.3  10.4
 1936  29.3   8.0
 1937  30.5   9.7
```

The number in column 1 is a year and columns 2 and 3 are the average yields of wheat and potatoes, respectively, in each year. You can use macro `readcols` to read `crops.dat` into three MacAnova variables:

```
Cmd>  readcols("crops.dat",year,wheat,potatoes)

Cmd>  print(year,wheat,potatoes)
year:
 (1)       1926       1927       1928       1929       1930
 (6)       1931       1932       1933       1934       1935
(11)       1936       1937
wheat:
 (1)       20.1       23.6       26.3       19.9       16.7
 (6)       23.2       31.4       33.5       28.2       35.3
(11)       29.3       30.5
potatoes:
 (1)        7.2        7.1        7.4        6.1          6
 (6)        7.3        9.4        9.2        8.8       10.4
(11)          8        9.7
```

Since `readcols` uses `vecread()`, it recognizes "`?`" as `MISSING`, skips everything that is not a number and is terminated by "`!`".

Although MacAnova can't read spreadsheet files directly, there is a good chance your spreadsheet program can create a file that `vecread()` and probably `readcols` read. Since different programs do things different ways, there is no single set of instructions that can be given. What you need to look for is a **Save** or **Save As** option such as **Text**, **Text File**, **Ascii File** or **Formatted Text**. In Excel on a Macintosh, after selecting item **Save** on the **File** menu, you need to click on an **Options** button to display the choices. In Excel on a PC, you select **Save As** on the **File** menu and then choose "Formatted Text" as the **Save File As Type** option. In Claris Works on a Macintosh, you make the selection right in the dialog box brought up by **Save** on the **File** menu. If you do it right, this should produce a file with the same rows and columns as your spread sheet, probably with the numbers in each row separated by tabs. This is exactly in the form expected by `vecread()` and `readcols`.

Command `vecread()` is useful only when a file contains a single data set. Another MacAnova command, `matread()`, is designed to read data sets from files containing many data sets, all in a special format. It cannot read files that `vecread()` can read, and vice versa. See the Users' Guide or type `help(matread)` for details on the actual format `matread()` expects. Suffice it to say here, that in the file to be read by `matread()`, every data set has a name and may have descriptive comments. To read in a particular data set you need to specify the name of the file *and* the name of the data set. In the following example, file `crops.mat` contains several data sets including a data set named `yields` which contains the same data as in `crops.dat`.

Here is how you read read the data in `crops.mat`:

```
Cmd>  data<-matread("crops.mat","yields")
yields      12     3
) Col. 1: year
) Col. 2: wheat = wheat harvest
) Col. 3: potatoes = potato harvest

Cmd>  dim(data) # data is a matrix with 12 rows and 3 columns
(1)          12          3
```

The first line that is printed echoes the line in the file with the data set name and dimensions. The remaining lines, all starting with "`)`", are descriptive comments in the file.

To get each column in a separate variable, you can either to it by hand

```
Cmd>  year <- data[,1]; wheat<- data[,2]; potatoes <- data[,3]
```

or use macro `makecols`:

```
Cmd>  makecols(data,year,wheat,potatoes)

Cmd>  list(year,wheat,potatoes) # now we have 3 vectors
potatoes        REAL   12
wheat           REAL   12
year            REAL   12
```

The header line and descriptive comments are echoed to the screen.

The file might have been created in a word processor such as Microsoft Word or a text editor such as Edit in DOS. If you use a word processor to create or edit data files to be read by MacAnova, it is *essential* that they be saved as Text or ASCII files. In some programs such as Microsoft Word, you may have to click on a **File Format** button to display choices; In others, the options may be displayed in the dialog box brought up by **Save**. If you do not choose a text format, MacAnova will probably not be able to read the file.

5 Visualizing numbers – drawing graphs

5.1 Basic graphic commands

MacAnova has quite good capability for making high resolution graphical displays and then modifying them. The basic commands for drawing graphs are as follows

Command	Description
`plot(x,y)`	Plot each column of matrix `y` against vector `x` using standard symbols
`chplot(x,y,symb)`	Same as `plot()` except you can specify plotting symbols
`lineplot(x,y)`	Same as `plot()` without plotting symbols but with lines
`addpoints(x,y)`	Add points to previously plotted graph using standard symbols
`addchars(x,y,symb)`	Add points to previously plotted graph using custom symbols
`addlines(x,y)`	Add lines to previously plotted graph
`addstrings(x,y,chars)`	Add character strings on previously plotted graph
`boxplot(arg)`	Make parallel boxplots of data in components of structure argument or columns of matrix argument
`showplot()`	Re-display most recent plot, possibly with new labels and changed minima or maxima

For the first seven of these commands (`plot()` through `addchars()`), `x` must be a `REAL` vector and `y` must be a `REAL` vector or matrix (vector only for `addstrings()`). The values in each column of `y` are plotted against the values in `x`. Ordinarily, `x` and `y` must have the same number of rows. For `chplot()` and `addchars()`, `symb` is a `CHARACTER` vector or a vector of integers; for `addstrings()`, `chars` is a `CHARACTER` vector.

The commands starting with "`add`" plot additional information on a previously constructed graph. See Sec. 5.3.

When `x` is just a single number `x0`, it is implicitly expanded to the vector `cat(x0, x0+1,...)` with the same number of rows as `y`. Thus, say, `plot(1,y)` plots `y` against 1, 2, 3, ..., and `plot(100,y)` plots `y` agains 100, 101, 102, ...

When `x` is a vector `cat(x0,inc)` of length 2, it is implicitly expanded to the vector `cat(x0, x0+inc, x0+2*inc,...)` with the same number of rows as `y`. For example, `plot(cat(0,1/60),y)` plots `y` against 0, 1/60, 2/60, ... and `lineplot(cat(1967, 1/12))` plots `y` against 1967, 1967+1/12, 1967+2/12, ... This last would be appropriate when the rows of `y` are monthly values starting January 1967 and you want to plot `y` against time in years.

These abbreviated forms are not legal for addstrings().

Use the `help()` (for example, type `help(plot)`), to get details on these commands.

5.2 Controlling the appearance of graphs using keywords

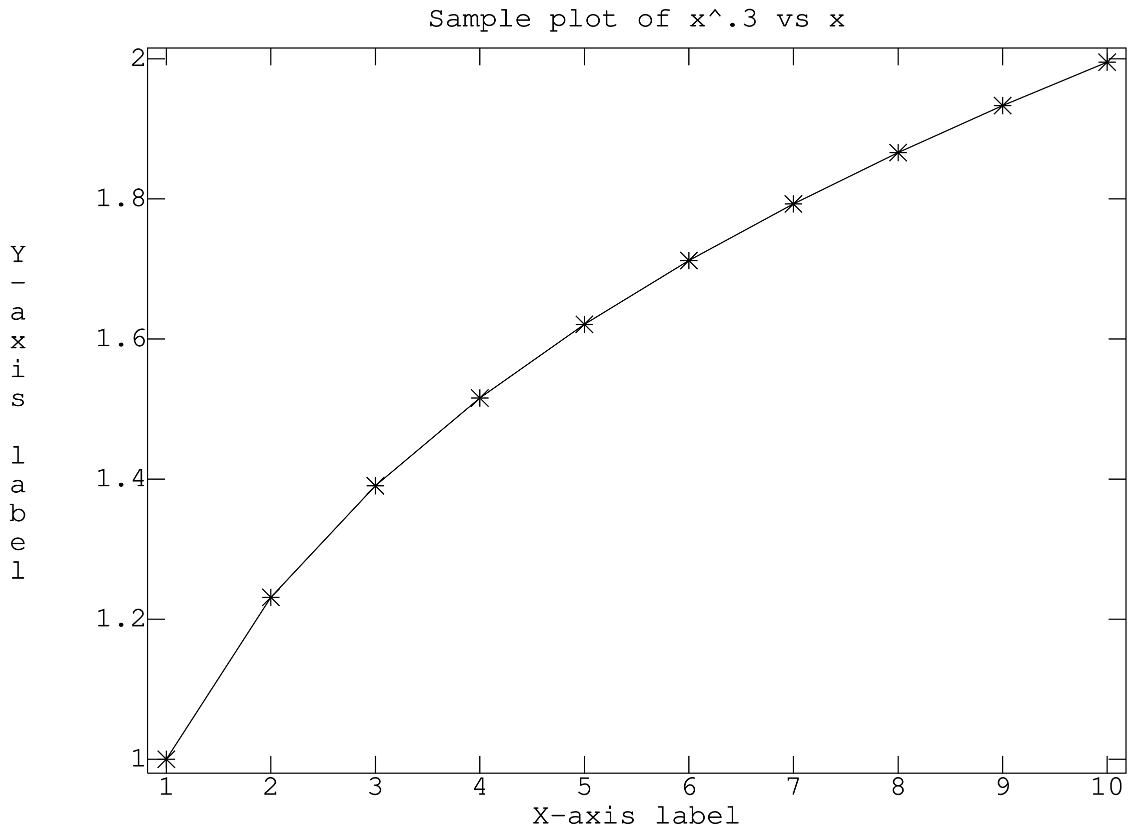
All the above graphing commands recognize many of the same keywords. Here is a list of keyword phrases that affect the appearance of graphs.

Keyword phrases	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`	Minimum value for y-axis
`xaxis:F`	Do not draw x axis (line y = 0)
`yaxis:F`	Do not draw y axis (line x = 0)
`lines:T`	Connect points with lines
`impulse:T`	Draw lines from points to y = 0 line
`linetype:n`	Sets the linetype to `n`, default is 1. `n` must be integer $1 \leq n < 100$
`thickness:w`	Sets the line thickness to `w` times normal thickness, default is 1. `w` must be between .1 and 10. Has no effect when `dumb:T` or where otherwise not feasible
`dumb:T`	Use printable characters only, producing crude plot suitable for printing on a line printer

These are generally self-explanatory. Most are illustrated in examples below. Keywords `linetype` and `thickness` are only legal on plots on which lines are drawn, and their effect depends on the particular Macanova version – Macintosh, Unix, DOS real mode, or DOS protected mode. Keyword `dumb` is particularly useful on DOS or Unix where it may be difficult to get hard copy of high resolution graphics.

Here is a simple example of the use of `plot()`, making use of some of these keywords.

```
Cmd>  x<-run(10);y<-x^.3 # y is x to the 0.3 power

Cmd>  plot(x,y,lines:T,xlab:"X-axis label",ylab:"Y-axis label",\
 title:"Sample plot of x^.3 vs x")
```



This plots vector `y` against vector `x` using a default plotting symbol, connecting the points with lines (because `lines:T` was an argument). A title was specified by keyword `title`, and X-axis and Y-axis labels were specified by keywords `xlab` and `ylab`, respectively.

5.3 GRAPH variables and modifying graphs

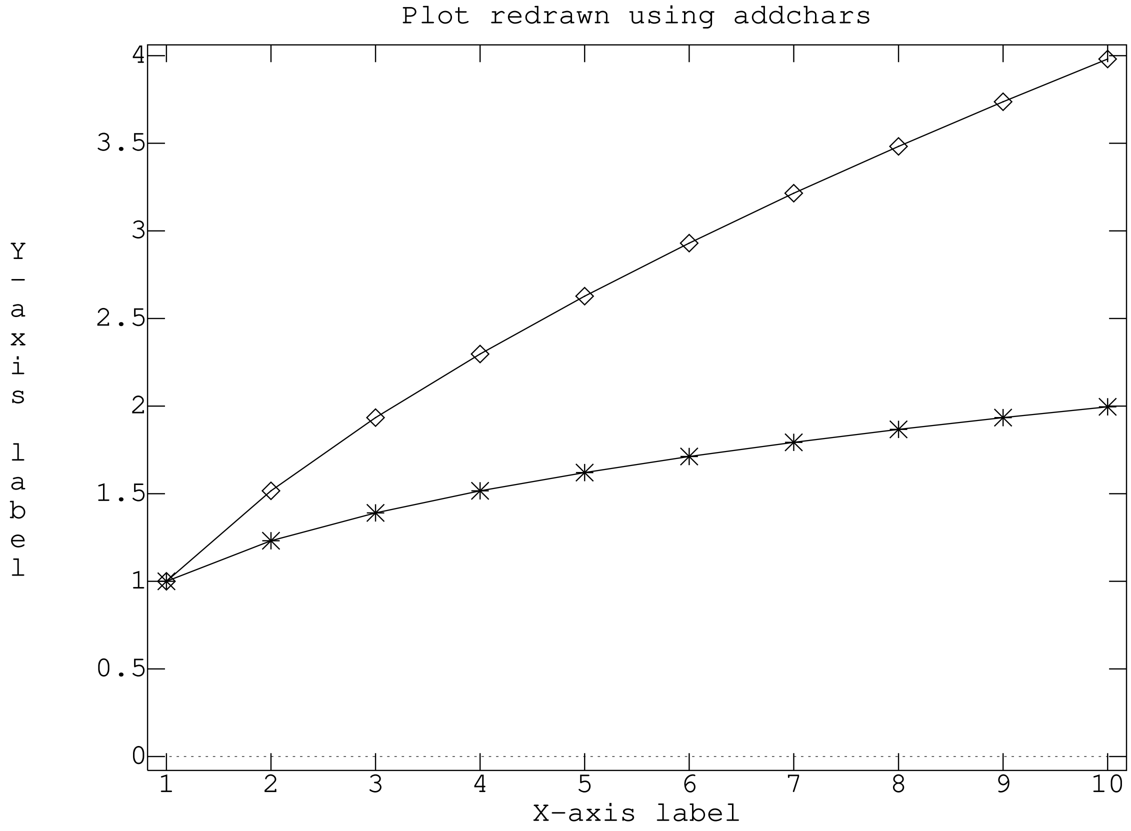
Normally, whenever MacAnova draws a graph, it also creates a side effect variable `LASTPLOT` which encapsulates all the information used to create the plot. `LASTPLOT` has special type `GRAPH`. A `GRAPH` variable can be assigned to another variable (for example, `plot1 <- LASTPLOT`). Typing its name or outputting it with `print()` or `write()` produces a plot similar to that produced when `dumb:T` is used.

Command `showplot()` with no arguments redisplays the graph encapsulated in `LASTPLOT`. Because all the keyword phrases listed above except `lines`, `impulse`, `linetype` and `thickness` can be used with `showplot()` you can change or add labelling information or to set the minimum or maximum values to appear in the graph.

By default, `addpoints()`, `addlines()`, `addchars()`, and `addstrings()` display the graph encapsulated in `LASTPLOT` with additional data included in the plot. The first three are are used identically to `plot()`, `lineplot()`, and `chplot()`, except they may have an optional `GRAPH` variable argument before `x`.

Here is an example in which we add to the plot in Sec. 5.2 additional line-connected points, using a special plotting symbol specified by the string `"\1"`. This is interpreted as a diamond. The same plotting symbol as in the first plot would be specified by `"\6"`. A simple dot would be specified by `"\7"`. We also reset the minimum and the maximum for the y-axis using keywords `ymin` and `ymax` (keywords `xmin` and `xmax` would do the same for the x-axis). Keyword `title` is used to provide a new title.

```
Cmd>  addchars(x,y^2,"\1",lines:T,\
 title:"Plot redrawn using addchars", ymin:0,ymax:y[10]^2)
```



An alternative plot, illustrating the use of `showplot()` is the following.

```
Cmd>  showplot(dumb:T,title:"Plot redrawn using dumb:T")
                            Plot redrawn using dumb:T
           ++-----+-----+-----+-----+-----+-----+-----+-----+------++
          4+                                                      .++
           |                                                  ....  |
           |                                              ..+.      |
        3.5+                                        ..+...          +
 Y         |                                  ..+...                |
 -        3+                            ..+...                      +
 a         |                       .+...                            |
 x      2.5+                    ...                                 +
 i         |                ..+.                                    |
 s        2+           .+...                              ..*......*+
           |        ...                 ..*.....*.....*...          |
 l      1.5+     .+.        ..*.....*...                            +
 a         |  ....*.....*...                                        |
 b        1++...                                                    +
 e         |                                                        |
 l      0.5+                                                        +
           |                                                        |
          0+........................................................+
           ++-----+-----+-----+-----+-----+-----+-----+-----+------++
            1     2     3     4     5     6     7     8     9     10
                                 X-axis label
```

This usage also illustrates the use of the keyword phrase `dumb:T` to produce a *low resolution* plot constructed from ordinary printing symbols. A dumb plot is not as elegant as the higher resolution plot, but it can be printed on an ordinary printer, and included in any word processor document. If you are saving input and output using `spool()`, dumb plots are written to the spooling file, but high resolution plots are not. The plot was printed slightly smaller than normal so as to fit on the page.

To modify or display another `GRAPH` variable, use it as the first argument to the command, for example, `showplot(plot1,title:"This is a new title")` or `addlines(plot1,x1,y1)`. These do not actually change `plot1`, but make a new version of `LASTPLOT` reflecting the new graph.

There are two additional keywords that may be helpful when modifying or re-displaying graphs:

Keyword phrases	Description
`keep:F`	Do not save plot as `LASTPLOT`.
`show:F`	Do not display plot, only save.

A `GRAPH` variable encapsulating a complicated graph can use a lot of room in your workspace. If you do not plan to modify the plot you can save space by including `keep:F` as an argument on any graphics command

When you are building a complicated graph in stages by adding different types of information, you may not want to see the graph until it is finished. If so, use `show:F` on every plotting command except the last one. Thus, for example, you could produce the second plot above by

```
Cmd> plot(x,y,lines:T,xlab:"X-axis label",ylab:"Y-axis label",\
 show:F)
Cmd> addchars(plot1,x,y^2,"\ 1",lines:T,\
 title:"Plot redrawn using addchars", ymin:0,ymax:y[10]^2)
```

It is illegal to use both `show:F` and `keep:F`.

5.4 Writing graphs to files

In addition to displaying graphs, you can write the graphical information to a file in the form of PostScript commands. PostScript is a powerful page description language recognized by certain printers, including Apple LaserWriters. It is capable of representing any graph that MacAnova could produce.

Alternative forms in which a graph may be saved in a file is as a dumb plot in a text file or as a sequence of Tektronix 4014 (a once popular computer graphics terminal) commands (Unix only).

Here are keywords phrases associated with writing graphs to files:

Keyword phrases	Description
`file:fileName`	Write PostScript to file `fileName`.
`new:T`	Clear file `fileName` before writing and write PostScript "prolog".
`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 this results in the Tektronix plotting commands being written to the file. On other computers, a dumb plot is written to the file.

If `dumb:T` is also an argument along with `file:fileName`, a low resolution plot is written to the file instead of PostScript.

An example would be the following:

```
Cmd> showplot(file:"myplots.ps",new:T)
```

MacAnova will write a PostScript description of the plot in `LASTPLOT` to file `myplots.ps`. On the first use (and only the first use) of `file:"myplots.ps"`, you should also include `new:T` as an argument, as in the example. This initializes the file with a prolog, without which the remainder of the file would be unintelligible. Subsequent writes should *omit* `new` or have `new:F`. It is beyond the scope of this document to explain in detail what you can do with the PostScript file when you leave MacAnova. On some Unix computers, the file can be printed using command `lpr`. For Macintoshes, there exist utilities (for example, `sendPS` and `LaserWriter Utility`) which can be used to send PostScript directly to a LaserWriter. In addition, there exist various programs for displaying postscript and for translating it into various other graphics formats. It is possible to include PostScript directly into documents processed by some programs, including LaTeX.

5.5 Making a copy of a graph for use in a document

The only universally applicable way to do this is to write a dumb plot to a file. If you have started spooling your output using `spool()`, dumb plots are included in the spooled output automatically. Alternatively, you can use `showplot()` to write a `GRAPH` variable to a file as a low resolution plot. Thus

```
Cmd>  showplot(file:"myplots",dumb:T)
```

writes the plot encapsulated in `LASTPLOT` to file `myplots` as a dumb plot.

On a Macintosh, when a graphics window is the front window, **Copy** on the **Edit** menu or **⌘-C** copies the graph to the Clipboard. You can then paste it (using **Paste** on the **Edit** menu or **⌘-V**) into the **Scrapbook** desk accessory, a word processor document or a graphics editor window.

On a PC compatible, no such direct copying of the graphics screen is possible at present. If you are using the protected mode DOS version, `screendump:"name.pcx"` can be an argument to any graphics command. This saves a copy of the plot in file `name.pcx` in so-called PCX format. This can be edited by various PC graphics programs and incorporated in some PC word processors documents. Unfortunately, at present, this version of MacAnova DOS has only two different line types, and the dashed one doesn't always come out the way you would like it.

Alternatively, there are several DOS "screengrabber" programs that may be useful. You need to start one of these before launching MacAnova. After you have displayed a high resolution graph you press a special key combination to write the contents of the screen to a file in special format. See Sec. B.4 in Appendix B for more information.

5.6 Making graphs on a Macintosh

On a Macintosh, there are four basic graphics windows, **Graph 1**, **Graph 2**, **Graph 3**, and **Graph 4**. A new plot normally goes in the lowest numbered available window. However, you can specify the window by keyword phrase `window:n`, where `n` is an integer between 0 and 4 (`0` means put the plot in the most recently used window). There is also a **Panel of Graphs** window, displaying the contents of the other windows in reduced form. When any of these graphics windows is the front window, **Copy** on the **Edit** menu (**⌘-C**) copies the picture to the Clipboard. Similarly, when a graphics window is at the front, you can print it by selecting **Print Graph...** on the **File** menu.

When a graphics window is in front, hitting Return or Enter will bring the command window forward. You can switch to any graphics window by selecting it on the **Windows** menu, or equivalently by pressing key combinations **⌘-1**, **⌘-2**, **⌘-3**, **⌘-4**, or **⌘-G** (for **Panel of Graphs** window). When the **Panel of Graphs** window is in front, clicking on any of the panels brings the corresponding graph window to the front.

The size of a dumb graph is determined by the size of the command window. To make a larger or smaller than normal dumb plot, resize the window. This is how the smaller than normal dumb plot above was created.

5.7 Making graphs under DOS

When running under DOS, there is only one window, namely the whole screen. When you make a high resolution plot, the contents of the screen are replaced by the graph. Hitting Return, erases the graph and returns to command mode, refreshing any commands and output that were on the screen before the plot. The protected mode version cannot currently make high resolution plots when Windows is active; it crashes. To make high resolution plots, quit Windows and then start up MacAnova at the DOS prompt.

DOS has no automatic facilities for doing anything with a graph. As mentioned above, the protected mode version of MacAnova allows you to save a copy of the graph in a file on disk using keyword phrase `screendump:fileName` on any plotting command. And, undoubtedly there are DOS utilities that can make use of the PostScript produced when you use keyword phrase `file:fileName`. See Sec. B.4 in Appendix B for information on other possibilities that may enable you to save or print high resolution graphs.

The real mode version of MacAnova supports CGA, EGA, VGA, 8514, and Hercules graphics adaptors; the protected mode supports only VGA.

5.8 Making graphs under Unix

MacAnova on Unix assumes that you are using a terminal or windowing system that can emulate a Tektronix 4014 terminal, a fairly standard type of graphical device. In particular, the popular `xterm` pseudo VT100 window on many workstations can emulate a Tektronix 4014. MacAnova translates the information to be plotted, including plot frames, boundary ticks, labels, lines, and points into the arcane sequences of characters that a Tektronix 4014 terminal expects. If MacAnova is running in a `xterm` window, a Tektronix graphics window is opened and drawn to. After a Return is hit, it then switches back to the VT100 window.

If `file:fileName,ps:F` are arguments to a plotting command, the literal Tektronix plotting sequences are saved in the file. There exist programs that can translate such a file to other formats such as PostScript.

With some terminals that emulate both a VT100 and a Textronix terminal, particular sequences of characters automatically switch the terminal from one emulation mode to the other. On Unix, MacAnova has pre-defined macros `tek` and `vt` that emit these sequences for switching to Tektronix mode and VT100 mode, respectively, when using a particular terminal emulator (Versaterm which runs on a Macintosh). For another terminal emulator, it may be possible to redefine `vt` and `tek` to emit the appropriate codes.

When using NSCA Telnet 2.6 on a Macintosh to connect to a Unix machine, the following are suitable definitions of `tek` and `vt`:

```
tek <- macro("putascii(27,12)");vt <- macro("putascii(27,24)")
```

Only `vt` is really needed, since Telnet 2.6 automatically switches to Tektronix mode provided it is enabled. Thus,

```
plot(x,y); vt()
```

will draw a plot and automatically switch back to VT100 mode.

When using the freeware Kermit program for PC compatibles to connect to Unix, redefine `tek` and `vt` as follows:

```
tek <- macro("putascii(cat(27,91,63,51,56,104))")
vt <- macro("putascii(cat(27,91,63,51,56,108))")
```

Then the sequence, say,

```
tek();plot(x,y);vt()
```

will switch to Tektronix mode, draw the plot, and then switch back after you hit Return. You might even write a macro that you could use as a substitute for `plot()`:

```
tekplot <- macro("tek();plot($0);vt()")
```

to be used the same way as `plot()` but which automatically switches modes. A suitable place for defining such macros is in the initialization file (named `.macanova.ini in your home directory` on Unix). Briefly, this is a text file which contains MacAnova commands which are executed before the first `Cmd>` prompt appears. For more details see help topic `customize`.

6 Examples of statistical analyses

6.1 Introduction

This section includes some examples of statistical analyses using MacAnova. There is very little discussion of the analyses or explanation of the commands illustrated. Use `help()` or see the Users' Guide to get details of the general use of these commands. See a statistics textbook for information about the statistical techniques illustrated.

6.2 Histogram and pseudo-random number generation

In statistics courses we usually learn about normal or Gaussian data and especially about the standard normal distribution with mean 0 and standard deviation 1. Although it has been said that no real data is 100% normal, in MacAnova you can generate artificial data which really comes from a normal distribution. Here's how you might generate 5 independent standard normal random variables:

```
Cmd>  rnorm(5) # rnorm(n) generates n N(0,1) random numbers
NOTE: random number seeds set to 1025450084 and 305694887
(1)     -0.45119      -1.6091      -1.8971      0.34239     -0.52302
```

The line after the command clues us in to an important feature of the way MacAnova actually generates random numbers. Under the hood, as it were, there are two "seeds", numbers that are used to generate random numbers and which are updated every time `rnorm()` is used. If they haven't been set before the first time you use `rnorm()`, they are set automatically using the time and date, as was the case here when the starting seeds were set to 1,025,450,084 and 305,694,887. You can retrieve the current values with `getseeds()`:

```
Cmd>  getseeds()
Seeds are 1095363896 and 358403176
(1)   1.0954e+09    3.584e+08
```

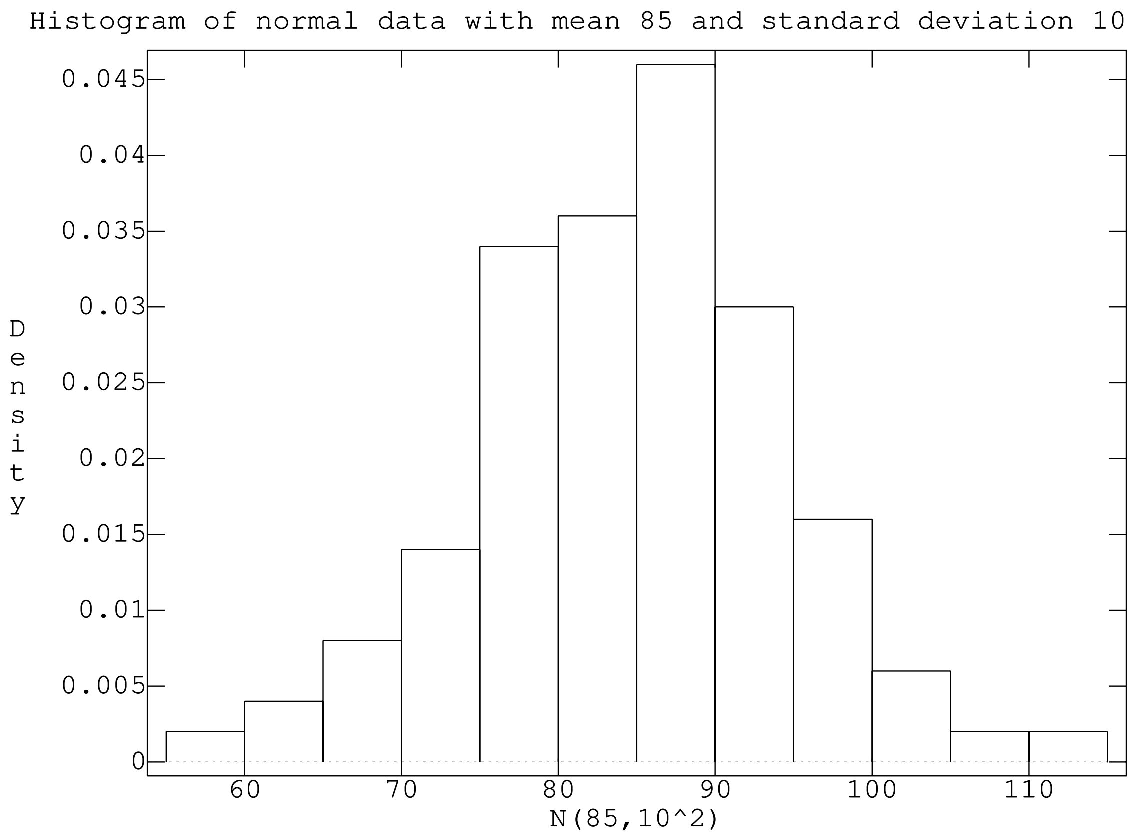
After generating 5 normals, the seeds have been updated to 1,095,363,896 and 358,403,176. You can set the seeds yourself using `setseeds()`:

```
Cmd>  setseeds(1025450084,305694887) # same as was previously chosen
Cmd>  rnorm(5) # the same random numbers were generated.
(1)     -0.45119      -1.6091      -1.8971      0.34239     -0.52302
```

This illustrates th important principle that when you start with the same seeds, you get the same random numbers.

We use `rnorm()` to generate artificial data to demonstrate how to use MacAnova to draw histograms. First the seeds are reset to specific values so that you can exactly reproduce the output by starting `rnorm()` at the same place.

```
Cmd>  setseeds(67871,32211) # reset seeds to some arbitrary values

Cmd>  x <- 85 + 10 * rnorm(100) # normal mu = 85, sigma = 10

Cmd>  describe(x,mean:T,var:T,min:T,max:T)
component: min
(1)       58.477
component: max
(1)       112.94
component: mean
(1)       84.776  Pretty close to 85
component: var
(1)       96.264  Pretty close to 10^2 = 100

Cmd>  hist(x,run(55,115,5),xlab:"N(85,10^2)",title:\
"Histogram of normal data with mean 85 and standard deviation 10")
```



The first argument to `hist` is the variable you are making a histogram of. The second specifies the class boundaries or limits. Since `run(55,115,5)` is the vector `cat(55,60,65,70,75,80,85,90,95,100,105,110,115)`, the limits are 55, 60, ..., 115.

The other arguments just affect the labelling. The heights of the bars are in the so called density scale with total area under the histogram equal to 1. Some statistics texts suggest making the height of the bar equal to the number of observations between the limits or equal to the relative frequency for the class. When the class limits are equally spaced as they are in the example, such histograms will look the same but will have different numbers on the vertical axis.

A simpler usage is `hist(x,10)`. This tells MacAnova you want 10 bars in the histogram, but leaves it up to MacAnova to pick the boundaries. This is good for a quick look at the data, but you probably won't like the boundaries MacAnova picks. For these data, the boundaries MacAnova selects are 55.754, 61.745, 67.736, ...,115.66, equally spaced with width 5.991.

6.3 Paired t analysis

Table 6.3.1 of Snedecor and Cochran (7th Edition, Iowa State Press 1980) gives the number of lesions on each of two halves of eight tobacco leaves. One half of each leaf was exposed to one preparation of a virus extract and the other half was exposed to another preparation. A paired analysis, based on the differences between the two halves of each leaf, is appropriate.

```
Cmd>  x1<-cat(31,20,18,17,9,8,10,7) # Data for preparation 1

Cmd>  x2<-cat(18,17,14,11,10,7,5,6) # Data for preparation 2

Cmd>  d <- x1 - x2; d ;n <- nrows(d) # differences
(1)          13           3           4           6          -1
(6)           1           5           1

Cmd>  stemleaf(d) # make stem and leaf display
    1   -0*|1
    3   +0*|11
    4   +0t|3
    4   +0f|45
    2   +0s|6
 High  13
        1*|1  represents 11  Leaf digit unit = 1

Cmd>  describe(d, n:T, mean:T, var:T)
component: n
(1)           8
component: mean
(1)           4
component: var
(1)      18.571
```

You could use MacAnova as a calculator to compute a paired t-test and a confidence interval for the mean difference from these values. Instead, you can use functions `tval()` and `tint()` which are specifically designed for this problem. You can use macro `twotailt` to compute P-values. Of course, they assume a random normal sample of differences for complete validity.

```
Cmd> tt <- tval(d-0)# (or tval(x1-x2)) sample size & t-statistic

Cmd> cat(tt, twotailt(tt,n-1))
(1)      2.6253    0.034144   t-statistic and 2 tail P-value

Cmd> tint(d,.95) # compute 95% confidence interval
(1)      0.3972      7.6028
```

The Student's t-statistic computed by `tval()` tests the null hypothesis H_0 that the expected difference is zero. Since the 0.034144 < .05 you can reject H_0 at $\alpha = .05$. With confidence 95% the expected difference is between .03982 and 7.6028.

6.4 Two-sample t-test

We analyze data on the weight gains in grams of 19 female rats, 12 on a high protein diet and 7 on a low protein diet, from Table 6.9.1 of Snedecor & Cochran:

```
Cmd> high <- cat(134,146,104,119,124,161,107,83,113,129,97,123)

Cmd> low <- cat(70,118,101,85,107,132,94)

Cmd> n1<- nrows(high); n2<-nrows(low); cat(n1,n2)
(1)         12          7


Cmd> describe(makestr(high,low), n:T, mean:T, var:T)
component: n
  component: high
(1)         12
  component: low
(1)          7
component: mean
  component: high
(1)        120
  component: low
(1)        101
component: var
  component: high
(1)     457.45
  component: low
(1)     425.33
```

Now you can use `t2val()` and `t2int()` to do a two-sample t-test and compute a two-sample confidence interval for the difference between the means.

```
Cmd>  tt <- t2val(high,low) # test statistic to test H0: μ1 = μ2

Cmd>  cat(tt, twotailt(tt,n1+n2-2)) # value of t and P-value
(1)       1.8914      0.07573   Not significant at 5% level

Cmd>  t2int(high,low,.95) # 95% confidence interval for μ1 - μ2
(1)     -2.1937       40.194    Includes 0
```

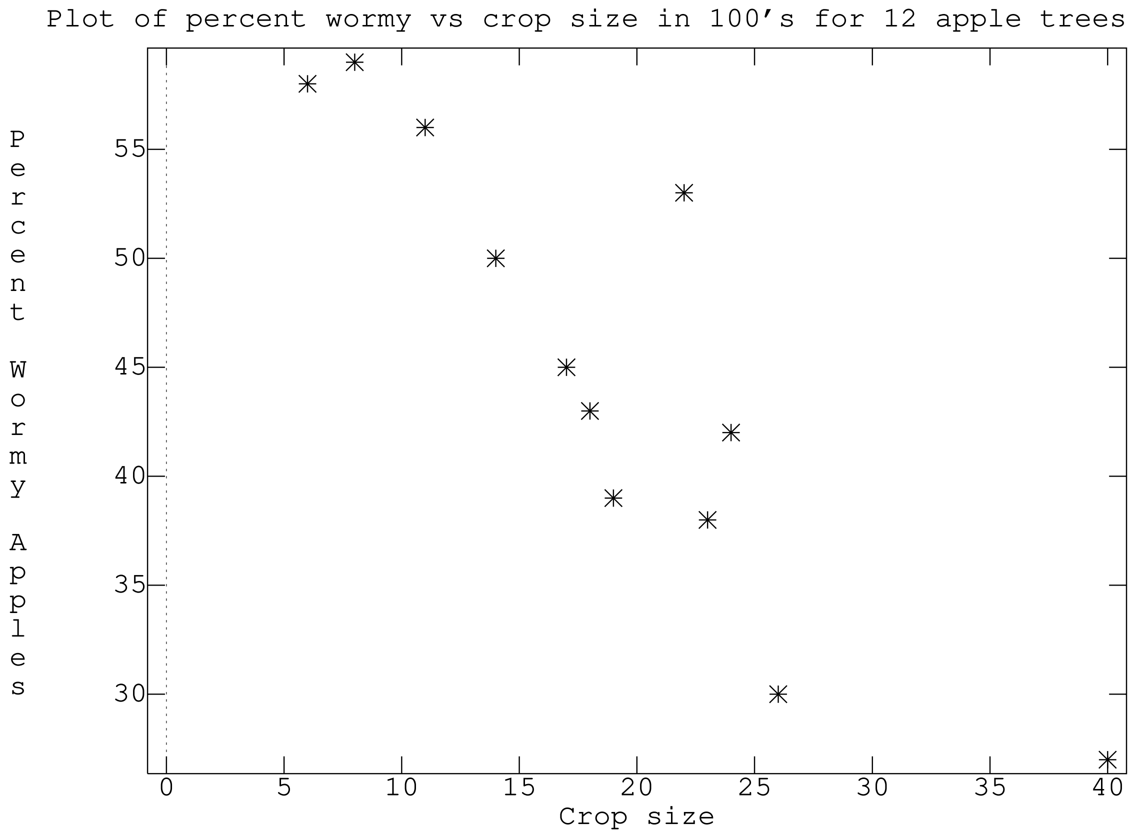
6.5 Simple linear regression

Here we analyze data from Table 9.7.1 of Snedecor and Cochran which gives the percentage of wormy fruit (`pcwormy`) and the number of apples harvested, in 100's, (`cropsize`) for 12 apple trees. The goal is to predict or explain the amount of wormy fruit in terms of the number of applies on the tree by a simple linear regression of `pcwormy` on `cropsize`.

```
Cmd>  cropsize<-cat(8,6,11,22,14,17,18,24,19,23,26,40)

Cmd>  pcwormy<-cat(59,58,56,53,50,45,43,42,39,38,30,27)
```

First we make a scatter plot of `pcwormy` against `cropsize`:

```
Cmd>  plot(cropsize,pcwormy,xmin:0\
 xlab:"Crop size",ylab:"Percent Wormy Apples", title:\
 "Plot of percent wormy vs crop size in 100's for 12 apple trees")
```



There is a fairly strong negative relationship between `pcwormy` and `cropsize`. Notice the use of keyword phrases with keywords `title`, `xlab`, `ylab` and `xmin` to control the appearance of a plot. These are not really necessary, but it is always a good idea to label your graphs informatively. Keyword phrase `lines:T` would connect successive points with lines, or you could use command `lineplot()`. You can select the plotting symbols used with command `chplot()`. See Sec. 5.

Now we do the actual regression analysis

```
Cmd>  regress("pcwormy=cropsize") # basic regression command
Model used is pcwormy=cropsize
                          Coef      StdErr           t
CONSTANT                 64.247      3.6029      17.832
cropsize                 -1.013     0.17215     -5.8842

N: 12   MSE:      27.384 DF: 10  R^2: 0.77590
Regression F(1,10):       34.624 Durbin-Watson: 1.68987
To see the ANOVA table type 'anova()'

Cmd>  twotailt(cat(17.832, -5.8842), 10)
(1)   6.5683e-09  0.00015431    P-values for t-stats
```

Both coefficients, the intercept or `CONSTANT` and the slope, the coefficient of `cropsize`, are significantly different from zero at the 5% level.

The general form of `regress()` for simple linear regression is `regress("y=x")`, where `x` and `y` are the names of vectors containing the independent and dependent variables, respectively.

Once the regression has been computed, you can retrieve the coefficients and their standard errors using function `secoefs()` which computes a structure. To get just the coefficients you can use `secoefs(se:F)` as an argument, while to get just the standard errors, you can use `secoefs(coefs:F)`:

```
Cmd>  secoefs() # this gets both coefficients and std. errors
component: CONSTANT
  component: coefs
(1)      64.247
  component: se
(1)      3.6029
component: cropsize
  component: coefs
(1)      -1.013
  component: se
(1)     0.17215

Cmd>  beta<-secoefs(se:F); ses<-secoefs(coef:F) #get separately

Cmd>  # beta contains coefficients, ses contains standard Errors

Cmd>  # Now compute lower and upper 95% confidence limits

Cmd>   n <- nrows(cropsize)# sample size

Cmd>  critval<-invstu(1-.025, n-2) # Student's t critical value

Cmd>  beta-critval*ses # lower confidence limits
component: CONSTANT
(1)      56.219
component: cropsize
(1)     -1.3966

Cmd> beta+critval*ses  # upper confidence limits
component: CONSTANT
(1)      72.275
component: cropsize
(1)    -0.62941
```

6.6 One-way Analysis of Variance

Here is a table of yields of four varieties of wheat, each grown on several plots with similar soils (Table 48 of *Biometricheskiye Metodi* of V. Yu. Urbakh, Science Press, Moscow 1964):

Variety 1	Variety 2	Variety 3	Variety 4
17.0	15.8	17.4	15.7
17.2	17.0	16.6	16.8
16.1	16.4	16.2	15.1
17.0		15.6	15.2
16.8		15.5	
		17.2	

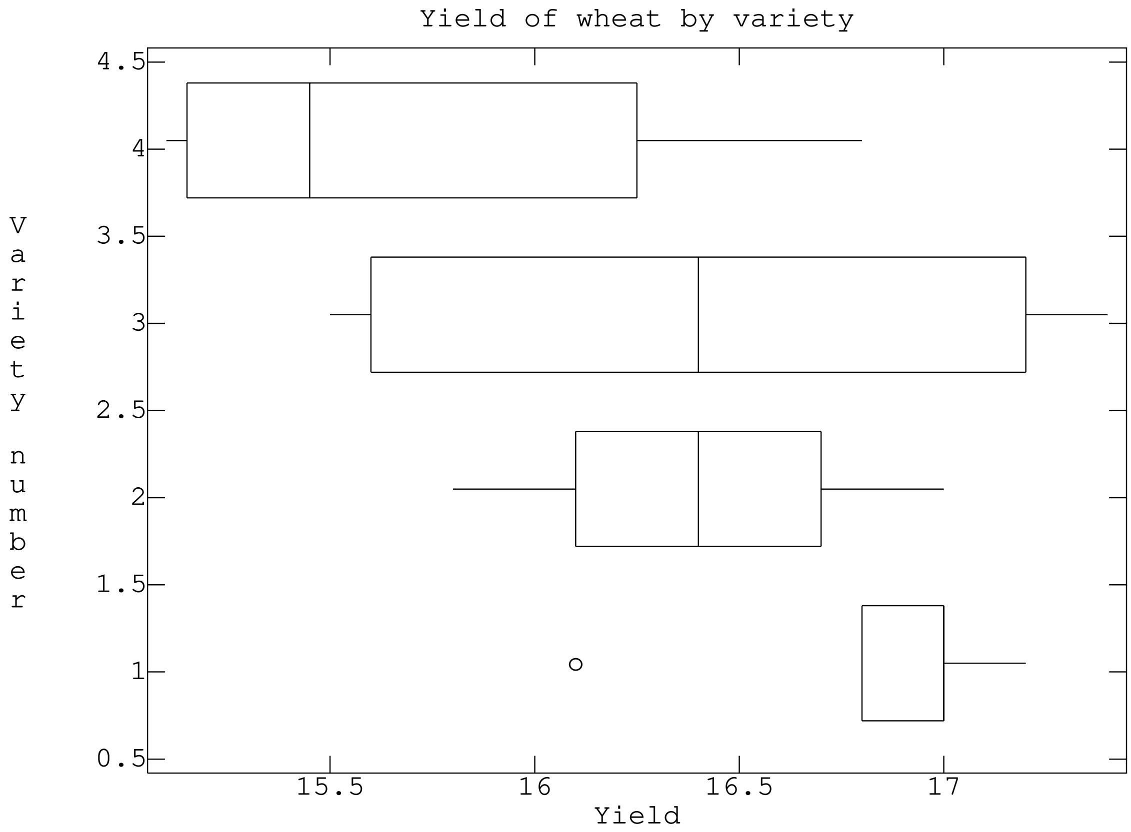
Here is a partial analysis of these data using the `tabs()`,`boxplot()` and `anova()` commands.

```
Cmd>  yield <- cat(17,17.2,16.1,17,16.8) # enter yield data,

Cmd>  yield <- cat(yield,15.8,17,16.4) # making one long vector

Cmd>  yield <- cat(yield,17.4,16.6,16.2,15.6,15.5,17.2)

Cmd>  yield <- cat(yield,15.7,16.8,15.1,15.2)

Cmd>  #Now enter variety numbers in another long vector

Cmd>  variety<-factor(cat(rep(1,5),rep(2,3),rep(3,6),rep(4,4)))

Cmd>  hconcat(variety, yield) # look at them together
 (1,1)          1         17
 (2,1)          1       17.2
 (3,1)          1       16.1
 (4,1)          1         17
 (5,1)          1       16.8
 (6,1)          2       15.8
 (7,1)          2         17
 (8,1)          2       16.4
 (9,1)          3       17.4
(10,1)          3       16.6
(11,1)          3       16.2
(12,1)          3       15.6
(13,1)          3       15.5
(14,1)          3       17.2
(15,1)          4       15.7
(16,1)          4       16.8
(17,1)          4       15.1
(18,1)          4       15.2

Cmd>  list(variety,yield) # variety is a factor
variety          REAL   18    FACTOR with 4 levels
yield            REAL   18
```

```
Cmd>  tabs(yield,variety) # compute variety means and variances
component: mean
(1)       16.82        16.4      16.417        15.7
component: var
(1)       0.182        0.36     0.63367     0.60667
component: count
(1)           5           3           6           4
```

Although the sample sizes are rather small, a box plot shows what is going on as well as the numbers do:

```
Cmd>  boxplot(split(yield,variety),\
 title:"Yield of wheat by variety",\
 xlab:"Yield",ylab:"Variety number")
```



The argument `split(yield,variety)` makes a structure with one component for each variety and then `boxplot()` draws a box plot for each component.

The box plots show some apparent difference between varieties. To test for its reality, you can compute an analysis of variance.

```
Cmd>  anova("yield=variety") # form of anova() for one-way ANOVA
Model used is yield=variety
```

```
WARNING: summaries are sequential
                                  DF          SS            MS
CONSTANT                           1      4821.6        4821.6
variety                            3      2.8237       0.94122
ERROR1                            14      6.4363       0.45974
```

For one-way ANOVA, the argument to `anova()` is a string of the form `"response = factor"`, where `factor` was created using function `factor()` as is the case here.

A side effect of `anova()` is the creation of two vectors, `SS` and `DF`, containing the sums of squares and their degrees of freedom, respectively.

```
Cmd>  hconcat(DF,SS,SS/DF) # compare with ANOVA table
(1,1)          1      4821.6       4821.6
(2,1)          3      2.8237      0.94122
(3,1)         14      6.4363      0.45974

Cmd>  f <- (SS[2]/DF[2])/(SS[3]/DF[3]) # compute F-statistics

Cmd>  cat(f,1-cumF(f,DF[2],DF[3])) # F statistic and P-value
(1)      2.0473     0.15348    Not significant at .10 level
```

6.7 Randomized Block (Two-way) Analysis of Variance

Here is a table of data from Table 14.2.1 of Snedecor and Cochran, from an experiment in which four seed treatments and a check (no treatment) were compared in a randomized block design with 4 replicates. The response is the percentage of seedlings in each plot that failed to emerge.

	Replicate #				
Treatment	1	2	3	4	5
Check	8	10	12	13	11
Arasan	2	6	7	11	5
Spergon	4	10	9	8	10
Semasan, Jr	3	5	9	10	6
Fermate	9	7	5	5	3

First, enter the data as one long vector, treatment by treatment, and then the replicate numbers and treatment numbers *as factors*:

```
Cmd>  failures<-cat(8,10,12,13,11, 2,6,7,11,5,\
 4,10,9,8,10, 3,5,9,10,6, 9,7,5,5,3)

Cmd>  reps<-factor(rep(run(5),5))
```

```
Cmd>  treatment<-factor(rep(run(5),rep(5,5)))

Cmd>  hconcat(reps, treatment, failures) # see them all
 (1,1)          1          1          8
 (2,1)          2          1         10
 (3,1)          3          1         12
 (4,1)          4          1         13
 (5,1)          5          1         11
 (6,1)          1          2          2
 (7,1)          2          2          6
 (8,1)          3          2          7
 (9,1)          4          2         11
(10,1)          5          2          5
(11,1)          1          3          4
(12,1)          2          3         10
(13,1)          3          3          9
(14,1)          4          3          8
(15,1)          5          3         10
(16,1)          1          4          3
(17,1)          2          4          5
(18,1)          3          4          9
(19,1)          4          4         10
(20,1)          5          4          6
(21,1)          1          5          9
(22,1)          2          5          7
(23,1)          3          5          5
(24,1)          4          5          5
(25,1)          5          5          3

Cmd>  tabs(failures,treatment,mean:T,count:T) # treatment means
component: mean
(1)       10.8         6.2         8.2         6.6         5.8
component: count
(1)          5           5           5           5           5

Cmd>  tabs(failures, reps, mean:T,count:T) # block means
component: mean
(1)        5.2         7.6         8.4         9.4           7
component: count
(1)          5           5           5           5           5

Cmd>  anova("failures=reps+treatment") # do analysis of variance
Model used is failures=reps+treatment
                                 DF          SS           MS
CONSTANT                          1      1413.8       1413.8
reps                              4       49.84        12.46
treatment                         4       83.84        20.96
ERROR1                           16       86.56         5.41

Cmd>  f <- (SS[3]/DF[3])/(SS[4]/DF[4]) # Compute F-statistic

Cmd>  cat(f,1-cumF(f,DF[3],DF[4])) # F-statistic and P-value
(1)      3.8743    0.021886     Significant at 5% level
```

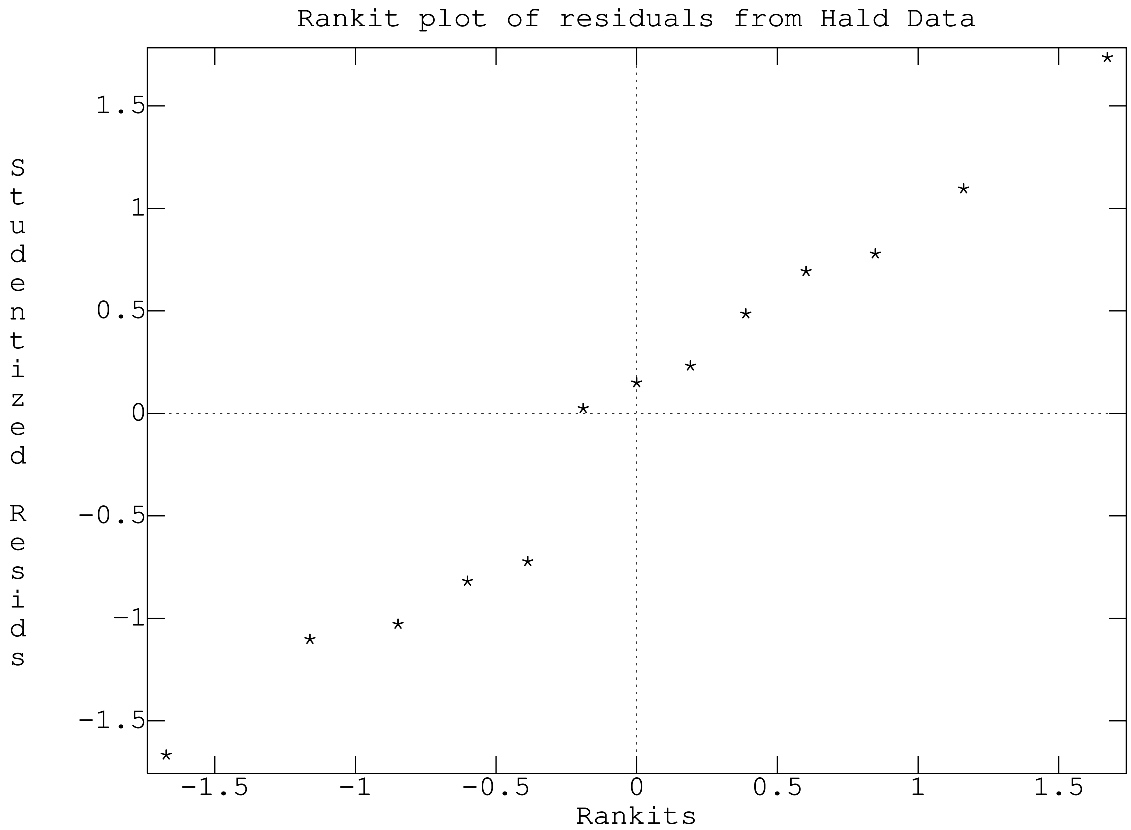
The form of `anova()` for a randomized block command is `anova("response = blocks + treatment")`, where `response` is the variable being analyzed, and `blocks` and `treatments` are vectors of block number and treatment number created by function `factor()`.

6.8 Multiple Regression

As an example we analyze some data in file `MacAnova.dat` distributed with MacAnova (it may not be in the file named `macanova.dat` on the computer you are using). It is a named data set with name `halddata` and hence we use `matread()` to read it. See Sec. 4.4.

```
Cmd>  hald <- matread("macanova.dat","halddata")
halddata     13      5 format
) Hald data from A. Hald, Statistical Theory with Engineering
) Applications
) Wiley, New York, 1960, p. 647
) Col. 1: X1 = percent tricalcium aluminate
) Col. 2: X2 = percent tricalcium silicate
) Col. 3: X3 = percent tetracalcium alumino ferrite
) Col. 4: X4 = percent dicalcium silicate
) Col. 5: Y  = cumulative heat of hardening after 180 days
)              (calories/gm)

Cmd>  makecols(hald,x1,x2,x3,x4,y); list(x1,x2,x3,x4,y)
x1              REAL   13
x2              REAL   13
x3              REAL   13
x4              REAL   13
y               REAL   13

Cmd>  regress("y=x1+x2+x3+x4")
Model used is y=x1+x2+x3+x4
                          Coef      StdErr           t
CONSTANT                 62.405      70.071      0.8906
x1                       1.5511     0.74477      2.0827
x2                      0.51017     0.72379     0.70486
x3                      0.10191     0.75471     0.13503
x4                     -0.14406     0.70905    -0.20317

N: 13   MSE:       5.983 DF: 8   R^2: 0.98238
Regression F(4,8):       111.48 Durbin-Watson: 2.05260
To see the ANOVA table type 'anova()'

Cmd>  anova() # also look at the sequential ANOVA table
Model used is y=x1+x2+x3+x4
WARNING: summaries are sequential
                                 DF          SS           MS
CONSTANT                          1  1.1837e+05  1.1837e+05
x1                                1      1450.1       1450.1
```

```
x2                                  1      1207.8      1207.8
x3                                  1      9.7939      9.7939
x4                                  1     0.24697     0.24697
ERROR1                              8      47.864       5.983

Cmd>  f<-(sum(SS[run(2,5)])/4)/(SS[6]/DF[6]) # regression F stat.

Cmd>  cat(f,1-cumF(f,4,DF[6])) # F and P-value; highly signif
(1)      111.48  4.7562e-07

Cmd>  beta<-secoefs(se:F);ses<-secoefs(coef:F)

Cmd>  critval<-invstu(1-.05/2,DF[6])

Cmd>  beta - critval*ses # lower limits
component: CONSTANT
(1)     -99.179
component: x1
(1)    -0.16634
component: x2
(1)     -1.1589
component: x3
(1)     -1.6385
component: x4
(1)     -1.7791

Cmd>  beta + critval*ses # upper limits
component: CONSTANT
(1)      223.99
component: x1
(1)      3.2685
component: x2
(1)      2.1792
component: x3
(1)      1.8423
component: x4
(1)       1.491

Cmd>  resvsrankits(title:"Rankit plot of residuals from Hald data")
```



This shows no obvious signs of non-normality in the residuals.

A MacAnova on a Macintosh

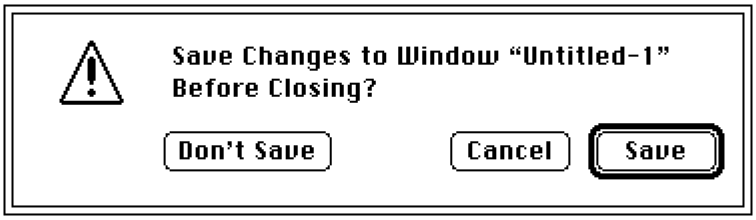
A.1 Command Windows

On the Macintosh, everything except high resolution graphing happens in a *command window*. You type commands and comments in the window and MacAnova prints all its non-graphics output it. A command window is editable and scrollable, much like a window in an editor or word processor. Although you seldom need more than one command window, you can have as many as nine, the one in front being the active one. When we talk about *the* command window below, we mean the one in front.

The *command line* starts one space after the last `Cmd>` prompt in the command window. It consists of everything you type after the prompt and may actually run on over several lines. You execute what you have typed on the command line by hitting Return when the *insertion point* (the “I-beam” cursor) is at the end. Alternatively, no matter where the insertion point is, you can execute the command line by hitting Enter. Up until you hit Return at the end of the line or Enter, you can go back and edit the line. When you use Enter, you don’t need to move back to the end of the line to execute it.

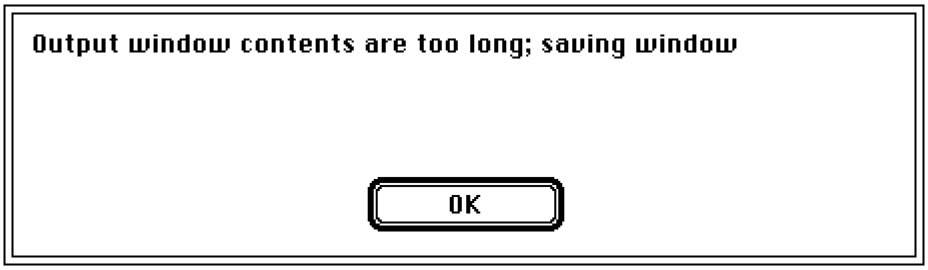
You can move around the command window using your mouse. If you prefer, you can use the arrow keys or certain command key combinations. See the **Windows** menu below. And you can edit anything in the command window, deleting, changing, or just adding new stuff, just as you might in a word processor. However, nothing that is typed before the last `Cmd>` prompt has any effect on what MacAnova does.

All command windows have a close box in the upper left hand corner and a zoom box in the upper right hand corner. If you close a window or quit, a dialog box similar the following appears



If you click on **Save**, the usual scrolling dialog box appears allowing you to select a folder and provide a file name. If you click on **Don’t Save**, nothing is saved, while **Cancel** stops the closing or quitting.

A command window holds up to about 28,000 characters, but the larger the contents of active window the more slowly MacAnova will print. For this reason, you may wish periodically to clean up the command window by deleting unwanted material. Alternatively you can open additional windows; see the **Windows** menu below. When the active command window gets too full, the following alert box appears:



Then you are given an opportunity to save it. Whether or not you save the window that is full, a new one is opened to continue your session.

A.2 Files

All files distributed with MacAnova should be in the same folder as the MacAnova application itself. The most important of these are the help file `MacAnova.hlp` and the macro files `MacAnova.mac`, `Design.mac` and `Tser.mac`. If you are using an initialization file (always named `MacAnova.ini`), it should go here also. Type `help(customizing)` for more information. It is not necessary that your data files or the files created by `save()` or **Save Window** be in this folder.

In order to be able to use macros `getdata` and `getmacros` which read from files whose names are in variable `DATAFILE` and `MACROFILE`, respectively, it is a good idea to have lines like the following in `MacAnova.ini`.

```
DATAPATH <- "HardDisk:MacAnova Folder:"
DATAFILE  <- paste(DATAPATH,"MacAnova.dat",sep:"")
MACROFILE <- paste(DATAPATH,"MacAnova.mac",sep:"")
```

The value assigned to `DATAPATH` is the name of the folder where `MacAnova.dat` and `MacAnova.mac are`. The colons (“`:`”) in `DATAPATH` separate the names in a “chain” of folders, each one inside the preceding, starting with the name of the hard disk.

A.3 Launching MacAnova

If your Macintosh does not have a hard disk, you will need two 800K (double density) or 1.44M (high density) floppy disks, one for the MacAnova application, together with the help file and macro file, (`MacAnova.hlp` and `MacAnova.mac`), and if there is room a data file. On the other disk should be a System Folder and any other files you need. First, of course, you will have to turn on the computer, and then, if necessary, open the disk icon to find and open the folder containing the MacAnova application.

MacAnova may be started up in several ways.

1. Double clicking the MacAnova application icon .

2. Double clicking on a Save File icon created by `save()` or `asciisave()` or by **Save Workspace** on the **File** menu. This restores the workspace previously saved.

3. Double clicking on an Output Window icon , created when you save the command window using **Save Window** or **Save Window As...** on the **File** menu. This loads the command window with commands and output from a previous run.

4. Selecting both a Save File icon and an Output Window icon and then double clicking restores *both* the previous workspace and previous output. This means you can end a session and restore things exactly to how they were at the point you quit.

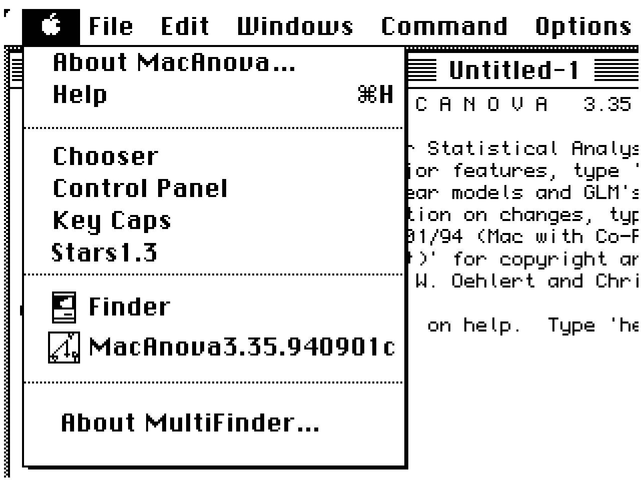
5. If you hold down ⌘ (the command key) while MacAnova is starting, MacAnova starts up in non-interactive mode. You will be asked to select a text file containing commands to be executed and a file in which the output is to be saved. No command window will be opened.

Under System 7, you may want to put an alias of MacAnova in the **Apple Menu Items** folder in the System folder.

A.4 Menus

There are 7 menus, the , **File**, **Edit**, **Windows**, **Command**, **Options** and **Font** menus.

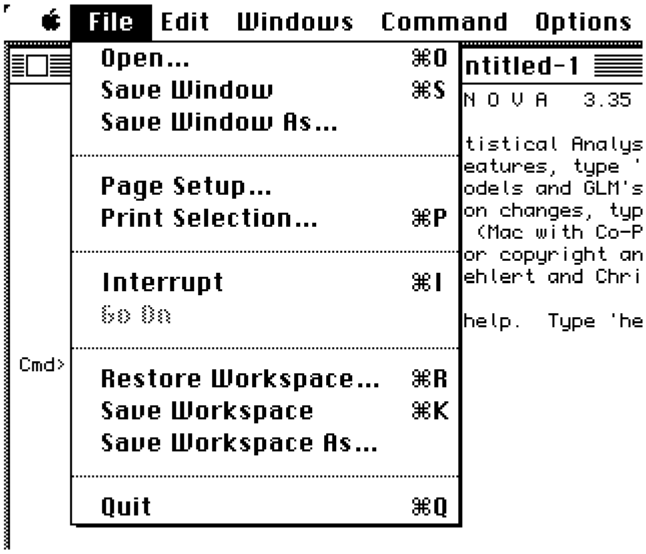
A.4.1 The Apple menu



About MacAnova displays a dialog box giving the version and copyright information. **Help** is roughly equivalent to typing `help()`. You will get a complete listing in the

Command Window of all topics available. If you have used the mouse to select a topic name in the window, **Help** gives you help on that topic. If your keyboard has a help key, hitting help does the same as selecting **Help**. See Sec. 3.6. The rest of the items are either Desk Accessories, or under System 7, applications. These have no effect on MacAnova, but launch other programs that can run at the same time. Note, however, at the current time, MacAnova does not background; when it is not the front most application, it does nothing.

A.4.2 The File menu



Open... allows you to load the contents of a previously saved command window into a new window. It becomes the active window. You use the usual scrolling dialog box to select the file.

Save Window writes the active command window to a file, overwriting anything already in the file if it exists. If there is no file already associated with the window, a scrolling dialog box will ask you to provide a name and select a folder. This file may be printed or edited using any word processor such as Microsoft Word, or Desk Accessory editor such as MockWrite or MiniEdit. See Sec. 4.2.

Save Window As... writes the active command window in a new file, overwriting anything already in the file if it exists.

Page Setup... brings up a standard page setup dialog box. If a command window is the front window, the options you choose will affect the printing of command windows. If a graphics window is in front, they will affect printing of any graphics window. If you don't select **Page Setup...** , the page setup dialog box will be displayed the first time you print a command or graphics window.

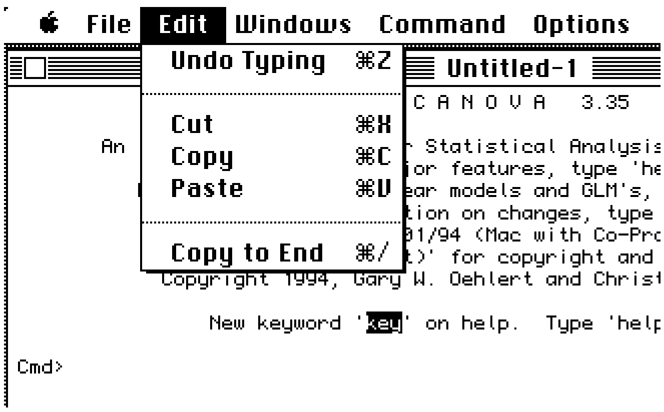
The **Print Selection** menu item can also be **Print Window** or **Print Graph**. You can use it to print text selected by the mouse in the active command window, the entire window, or the frontmost graphics window.

Interrupt breaks off output and most commands. `***** Interrupt *****` is printed followed by a new `Cmd>` prompt. It is usually selected by **⌘-.** or **⌘-I** . **Go On** is active only after a plotting command with `pause:T` as an argument, which causes MacAnova to pause, allowing you to **Copy** a plot to the clipboard or print it. When you select **Go On** (or press any key) MacAnova continues. **Restore Workspace...** allows you to restore a workspace previously saved by `save()` or `asciisave()`. Selecting **Restore Workspace** is equivalent to typing `restore("")`. It brings up a scrolling dialog box for you to select the file. See Sec. 4.3.

Save Workspace... saves the current workspace containing all your variables and macros. If the workspace has not been saved previously on the current run, **Save Workspace** is equivalent to typing `save("")` and you will be presented with a scrolling dialog box to select the save file. If the workspace has been saved previously on this run, it will be saved again to the same file, without asking you to select a file, as if you had simply typed `save()`. Since **Save Workspace** can be selected by **⌘-K**, this provides a quick and easy way to play it safe, saving your workspace every few minutes to minimize your losses in case of a crash. See Sec. 4.3. **Save Workspace As...** always asks you for a file name before saving your variables and macros.

Quit terminates MacAnova. Alternatively you can type `quit`, `stop`, `bye` or `end`, or, if there is only a single command window, you can click in its close box. You will be asked whether you want to save the workspace or the command window. To bypass these requests, hold the option key down.

A.4.3 The **Edit** menu



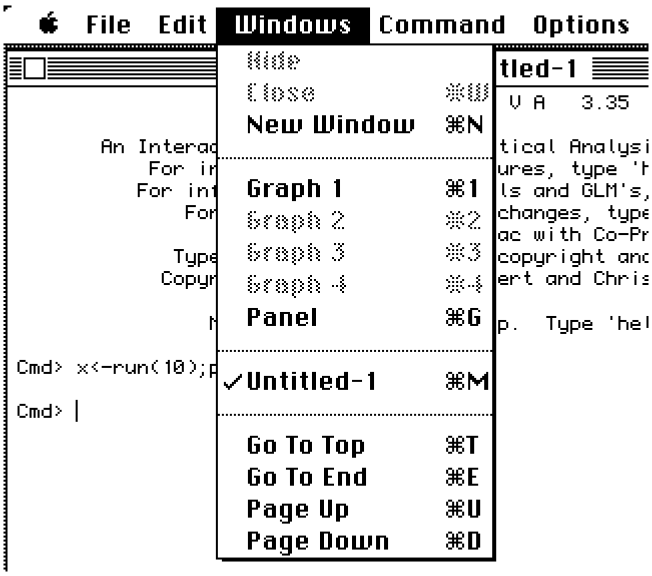
Instead of **Undo Typing**, the first item may be **Undo Paste**, **Undo Output**, **Redo Typing** or **Redo Output**. All of these let you undo or redo your previous action as long as you haven't done anything else to change the window. **Undo Output** is active immediately after the output from executing a command line. If you select it, output is deleted back to the end of the previous command line. This is useful if you made a typing error. Hit

⌘-Z, make the correction, and hit Enter to redo the command. If instead of changing the command, you immediately select **Redo Output**, the deleted output is restored.

When the Command Window is in front, **Cut**, **Copy**, and **Paste** do just what you expect on a Macintosh. When a graphics window is in front, only **Copy** is effective, copying the entire graph to the Clipboard.

Copy to End When you select text in the output window using the mouse, **Copy to End** copies the selected text to the end of the command line, leaving the insertion point after the copied text. You can then edit it, if necessary, before executing it. In the same circumstance, simply hitting Enter first copies the selected text to the command line and then adds a Return, thus executing the copied text immediately unless there is an unfinished quoted string or compound command line started with "`{`".

A.4.4 The **Windows** menu

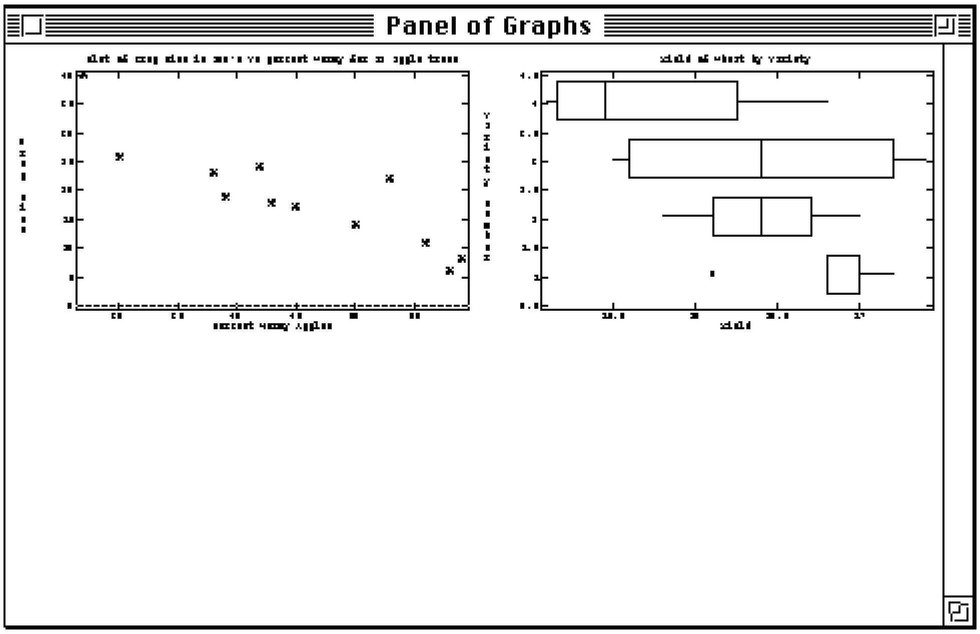


Hide and **Close** are operative when a graph window is in front or when there are two or more command windows. **Hide** hides the frontmost window, but the window can be made visible again by selecting its name on the **Windows** menu. Selecting **Close** is equivalent to clicking in the Close box of the window; it removes the window entirely. If it is a command window that has changed since it was opened, you are asked if you want to save it.

New Window opens a new command window with the name **Untitled-n** where n is an integer. A prompt `Cmd>` is printed and the window becomes the active command window. You can have no more than 9 command windows at any one time.

Graph 1, **Graph 2**, **Graph 3**, and **Graph 4**. Selecting any of these makes the indicated graphics window the front window. If a graph window does not exist, the corresponding item is not highlighted.

Panel. MacAnova maintains a fifth graphics window, the **Panel of Graphs** window, with miniature copies of the other four windows in its corners. Selecting **Panel** makes it the front window. Here is an example of a **Panel of Graphs** window when graphs have been drawn in **Graph 1** and **Graph 2**.



Clicking on any of the small copies brings the full sized window forward. The **Panel of Graphs** window can be copied to the clipboard or printed just like the other graphics windows.

Hitting Return when any graphics window is in front, brings forward the active command window.

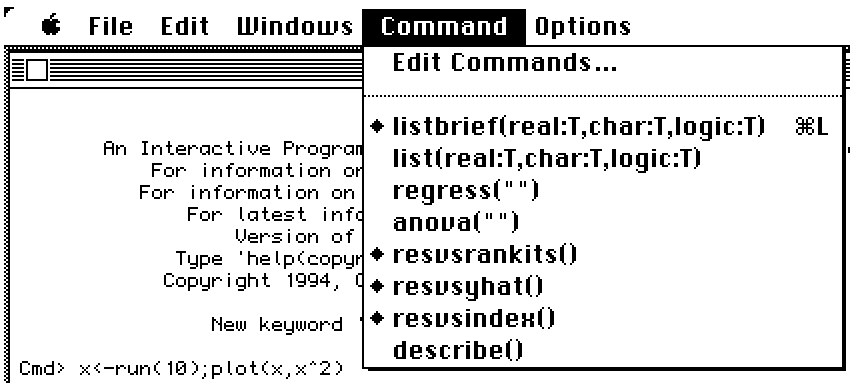
Untitled-1 There can be up to 9 command window names listed here. Selecting one makes it the active window. In the run from which this snapshot was taken there was only one command window named **Untitled-1**.

Go To Top scrolls to the start of the text in the active command window, but does not change the selection point. If there is a home key, it does the same.

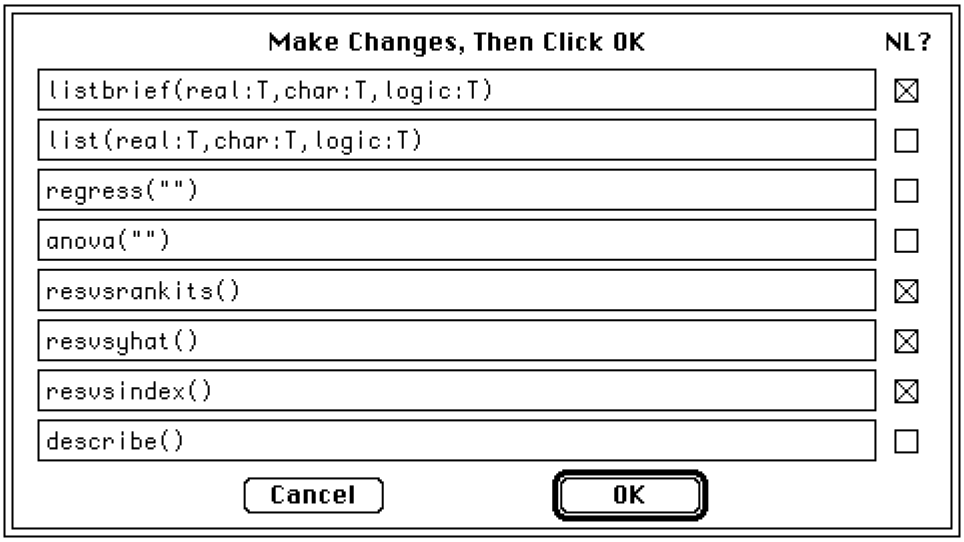
Go To End scrolls to the end of the text in the active command window and moves the selection point there. If there is a end key, it does the same.

Page Up and **Page Down** scroll the active command window up or down without moving the insertion point. If present, page up and page down keys do the same.

A.4.5 The Command menu



Edit Commands brings up the following dialog box so that you can change any or all of the commands in the last 8 lines of the **Command** menu.

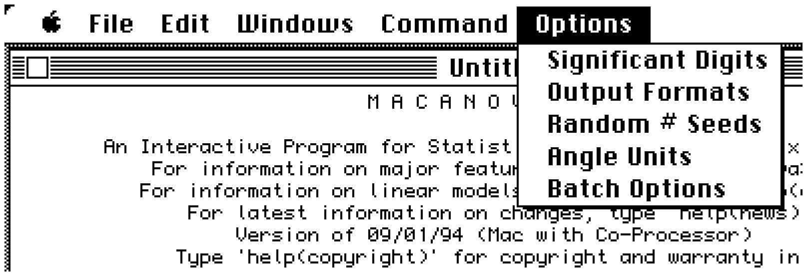


Check the box at the right if a Return should automatically be added to the command.

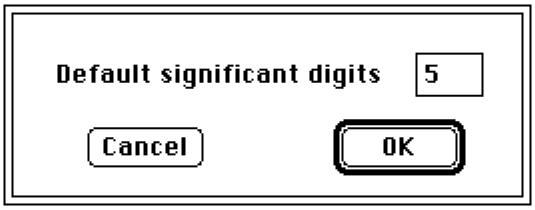
The last 8 items in the **Command** menu are predefined commands that are copied to the command line by selecting them with the mouse. A ◇ in the left margin indicates that the command will be followed by a Return, resulting in immediate execution. If ◇ is absent, no Return is added and you can edit the command before it is executed. The selection point will be inside a set of empty parenthesis (**describe()**) or empty quotation marks (**anova("")**). Thus selecting **resvsrankits()** results in an immediate residuals versus rankits plot, while selecting **anova("")** allows you to specify a model. You can replace or modify any of these commands using item **Edit Commands**. If you want to do anything complicated you should probably write a macro to do what you want and have only the macro invocation in the menu.

You can also select any of these commands by the key combinations ⌘**-Option-1**, ⌘**-Option-2**, etc.

A.4.6 The **Options** Menu

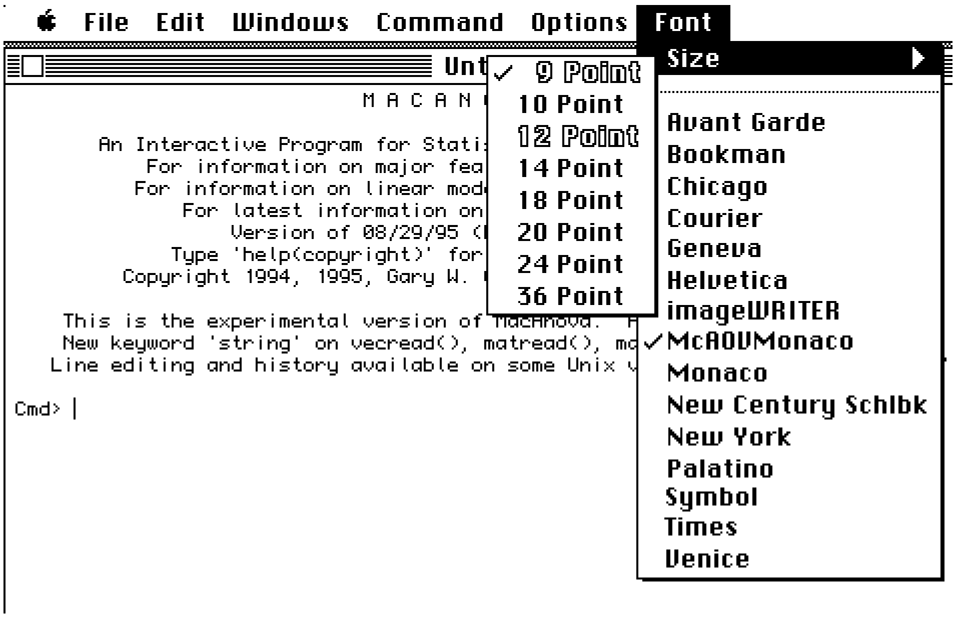


Each of the items on the **Options** menu opens a dialog box allowing you to set various options that would otherwise have to be set by command **setoptions()**. For example, if you select **Significant Digits**, the following dialog box appears:



Clicking **OK** executes the command **setoptions(nsig:5)**, specifying that the default output format will be floating point with 5 significant digits. **Output Formats** allows you more control over the default format. **Random # Seeds** allows you to set values to initialize the built in random number generator. **Angle Units** allows you to select which will be the default units of angles assumed by trigonometric functions and other commands – radians, degrees, or cycles (360° = 2π radians = 1 cycle). **Batch Options** allows you to set certain options pertaining to command **batch()**.

A.4.7 The **Font** Menu



You use this menu to change the font and/or font size used in the command window. The default font is **McAOVMonaco 9**, a special variant of **Monaco 9**. You probably will not want to change the font, but if you do, you should make sure the font is non-proportional, that is, all the letters and numbers have the same width. The only two standard fonts of this sort are **Monaco** and **Courier**. Selection of any other font almost certainly will mean that columns of numbers and other symbols will not line up and will be hard to understand.

A.5 Graphics windows

When one of the plotting commands is executed, a new graphics window is created, up to a maximum of four, not counting the **Panel of Graphs** window. If you try to make a graph when all the windows are in use, MacAnova prints

```
ERROR: graph windows full - delete a graph and type showplot()
```

Before you can make any more plots, you will have to close one or more graphics window. When you have freed up a slot, type `showplot()` and the graph you wanted will appear.

Alternatively, you can always specify a window number on any graphing command by keyword phrase `window:n`, where `n` is an integer from 1 to 4 or 0. `window:0` reuses the most recent used graphics window.

If `pause:T` is an argument to any plotting command, MacAnova pauses after the plot allowing you to **Copy** the graph to the clipboard or print it. When you select **Go On** on the **File** menu or press any key, MacAnova continues. Using `pause:T` together with `window:0` is particularly effective if you use a `for` or `while` loop to make a sequence of plots with differing data or parameters.

When a graphics window is the front window, selecting **Copy** on the **Edit** menu (**⌘-C**) copies the entire graph to the Clipboard. From there it can be pasted into the Scrapbook, or into a word processor or graphics editor window.

If you want to save a graph, **Copy** on the **Edit** menu is usually the best way. An alternative way is to save the graph in a file by using `file:fileName` as an argument to any plotting command. If `dumb:T` is also an argument, a “dumb” type plot that can be printed on a line printer is written to the file. Otherwise (no `dumb:T`) PostScript commands, recognized by many Apple LaserWriters and some other printers, are written to the named file. With the right programs, the file can be printed directly on a LaserWriter reproducing all the graphs saved in the file. You should use keyword phrase `new:T` the first time you write PostScript to a file but not subsequent times.

A.6 Other features

A.6.1 Interrupting MacAnova.

Some operations such as listing the values in a long vector or a large analysis of variance may take a long time to complete. If you wish to interrupt MacAnova during a long operation, select **Interrupt** from the **File** menu or press **⌘-.** or **⌘-I**. MacAnova may take a few moments to respond but will eventually stop what it is doing, print `***** Interrupt *****`, followed by a `Cmd>` prompt. Not all commands respond to **Interrupt**, but most do.

A.6.2 Executing other programs while in MacAnova

You can run other programs without quitting MacAnova if you are running under Multifinder or System 7. Just click on the desk top to return to the Finder or (under System 7) use the Applications menu at the far right end of the Menu bar, and then start another application up in the usual manner. If you are not using Multifinder or System 7, you are limited to executing Desk Accessories (DA’s) installed in your menu. One type of DA that is particularly useful is a DA text editor. Two of the most popular of these are MockWrite and MiniEdit.

A.6.3 Editing Macros

If you start to write your own macros, you will soon find you want to correct or improve them while you are running MacAnova. You can edit macros in at least two ways.

(a) You can write out the macro to a file using command `macrowrite()`. If you are running under Multifinder or System 7, you can then switch to an editor or word processor to edit the file without shutting down MacAnova; if not, shut down MacAnova and start the editor. When you are done editing, if the number of lines in the macro has changed you will need to update the number in first line Then you save the modified macro as a *text file*. After returning to MacAnova (or restarting it if necessary) read the macro back in using `macroread()`. It is a good idea to print it out to make sure it all got read. If some lines are missing from the end, it means the number of lines on the header was wrong.

(b) You can edit a macro right in the command window. First display the macro by typing its name. It will be printed, complete with enclosing quotation marks, and with any internal quotation marks preceded by "\". Make the necessary changes. Then move the insertion point to the `Cmd>` prompt and type

```
name <- macro(
```

without a closing parenthesis or Return, where `name` is the name of the macro or perhaps a new name. Now use your mouse to select the entire text of the edited macro, *including the enclosing quotation marks*, and then select **Copy to End** on the **Edit** menu or press ⌘-/. This will copy the macro to immediately after `macro(`. Finallt type the closing "`)`" and hit Return or Enter to re-create the edited macro.

A.7 Miscellaneous information

MacAnova for the Macintosh was compiled using MPW C. It comes in two versions, one using a 68881 math coprocessor and 68020 instructions, and one that will run on a vanilla Macintosh without a coprocessor. They work identically, but the coprocessor version is much faster except on a Power Macintosh for which you should use the version without a coprocessor. See the file `Readme.txt` distributed with MacAnova. Both versions require System 4.2 or later and at least 1 megabyte of ram. Although you could run MacAnova on a two floppy Macintosh, a hard disk is *strongly* recommended. MacAnova works well under Multifinder and under System 7 but does not take advantage of all their features. In particular, at present MacAnova does no computations in the background.

As distributed, the co-processor version has name of the form `MacAnova3.35.YYMMDDc`, where `YYMMDD` is the date (for example `941118`) of compilation, and the non co-processor version has name `MacAnova3.35.YYMMDDn`.

B MacAnova on an IBM Compatible

B.1 Introduction

This appendix summarizes features special to the DOS versions of MacAnova. There is as yet no Windows version. There are two DOS versions, a *real mode* one that should run on any PC compatible with sufficient memory, and a *protected mode* version that requires at least a 386. See B.6 below for a list of differences between them. The most important difference is in the maximum allowed size of variables: The real mode version is limited to variables taking no more than 65,000 bytes of memory (8,125 `REAL` numbers), while the protected mode version has no such limit, and can use any available extended memory or even the hard disk to work with large data sets. Under Windows, both must be run from the DOS prompt.

There are no menus and once a line has scrolled off the screen it is lost unless you previously had used `spool()` (see Sec. 4.2).

The remainder of this Appendix below assumes that MacAnova and its associated files such as `MACANOVA.HLP` are installed in directory `C:\MACANOVA`. If that is not where MacAnova is located on your computer, make the necessary translation of `C:\MACANOVA` wherever it appears. Also the executable file is assumed to be `MACANOVA.EXE`.

B.2 Initialization file

The initialization file (see help topic `customizing`) is named `MACANOVA.INI` and should be in `C:\MACANOVA`. If the number of lines on your screen is not 25 but is, say, 24, you will want to include the line

```
setoptions(lines:24)
```

in `MACANOVA.INI`.

B.3 Launching MacAnova under DOS

MacAnova is started up by typing

```
macanova [-q] [file options] [screen options]
```

at the DOS prompt, where items in `[...]` are optional. If the current directory isn't `C:\MACANOVA`, you may need to type `C:\MACANOVA\macanova` instead of just `macanova`.

If `-q` is present, no start up message will be printed.

The possible **file options** are `-f initFile`, `-r saveFile`, `-b batchFile`, `-h helpFile`, `-m macroFile`, and `-d dataFile`, where each must be a valid file name.

If `-f initFile` is specified, file `initFile` is executed silently as a batch file at start up instead of `MACANOVA.INI` (see help topic `customizing`).

If `-r saveFile` is specified, the equivalent of `restore("saveFile")` is executed at start up and the initialization file is not read. See Sec. 4.3.

If `-b batchFile` is specified, the equivalent of `batch("batchFile")` is executed after initialization (see help topic `batch`).

If `-h helpFile` is specified, MacAnova will use `helpFile` as the default file of help documentation rather than the standard help file `C:\MACANOVA\MACANOVA.HLP`.

If `-d dataFile` is specified, MacAnova `CHARACTER` variable `DATAFILE` will have `"dataFile"` as value. `DATAFILE` is used by pre-defined macro `getdata` to make it easy to read data from a standard file of data sets. If you do not use `-d dataFile`, `DATAFILE` will have default value `"C:\MACANOVA\MACANOVA.DAT"`.

If `-m macroFile` is specified, MacAnova, `CHARACTER` variable `MACROFILE` will have `"macroFile"` as value. `MACROFILE` is used by pre-defined macro `getmacros` to make it easy to read macros from a standard library of macros. If `-m macroFile` is not specified, `MACROFILE` will have default value `"C:\MACANOVA\MACANOVA.MAC"`.

Even if `-d` or `-m` is not used, the default values of `DATAFILE` and `MACROFILE` can be changed in the initialization file `C:\MACANOVA\MACANOVA.INI`. The help file can be changed by `help(file:"fileName")` which command could also be in `C:\MACANOVA\MACANOVA.INI`.

The permissible **screen options** are `-l nlines` and `-w ncols`, where `nlines` and `ncols` are integers greater than 5 and 20, respectively. These set the default number of screen lines and columns, respectively. The default values are 25 and 80. If your monitor has a different size, say with 24 lines, you will want to use `-l 24` on the command line or put `setoptions(lines:24)` in your `MACANOVA.INI` file.

If `input.txt` is a file which contains MacAnova commands to do an entire analysis,

```
macanova < input.txt > output.txt
```

will run MacAnova and save all the results in file `output.txt`.

Just to repeat, if `MACANOVA.EXE` is not in `C:\MACANOVA`, you will need to replace `C:\MACANOVA` by the appropriate directory name in the above descriptions.

B.4 Graphics under DOS

The real mode version of MacAnova under DOS supports CGA, EGA, VGA, 8514, and Hercules graphics adaptors. Drivers for these are available in files `CGA.BGI`, EGAVGA.BGI, HERC.BGI, and IBM8514.BGI which are distributed with MacAnova. They are copyrighted by Borland and cannot be resold. They must be in C:\MACANOVA along with MACANOVA.EXE. After each plot, MacAnova pauses with the plot on the screen until you hit Return, at which point the plot is erased, the text window is refreshed, and you are returned to the Cmd> prompt.

The protected mode version is currently limited to VGA graphics. Moreover, it cannot make high resolution plots when Windows is active (we hope this restriction will soon disappear). Currently it also has fewer and less satisfactory line types available for high resolution plotting.

To be able to print a MacAnova high resolution plot when running the real mode version, you need a special TSR (Terminate and Stay Resident) program which must be executed prior to launching MacAnova. Then, when a MacAnova plot is on the screen, the PrintScreen button on the keyboard should print the plot. The standard one that comes with DOS is GRAPHICS.COM; it may work with your printer.

When you are running the protected mode version at the DOS prompt without Windows, you can use keyword phrase screendump:"filename" on any graphics command to save a copy of the graphics screen in PCX format on your hard disk. The PCX file can be used by various graphics and word processing programs. This option is not available in the real mode version, although it may be possible to capture graphic images in a disk file for incorporation in a word processor using a TSR program. The TSR GRAB.COM distributed with Word Perfect is one option you may already have available.

File SCNGRAB.EXE distributed with MacAnova is an executable archive that contains some shareware programs for printing and capturing high resolution graphics. Running this program produces several files including some documentation files whose names end in .TXT or .DOC. These describe what the remaining files do and how to use them.

The file:fileName option on the plotting commands by default writes PostScript to the named file. PostScript is a page description language recognized by some printers, including Apple LaserWriters. You should use keyword phrase new:T the first time you write PostScript to a file but not subsequent times. If both file:fileName and ps:F are arguments to a plotting command, the plot written to the file is a dumb type that can be printed on a line printer.

B.5 Other features

You can execute DOS commands directly from MacAnova by prefixing the line to be executed with "!" in the first position after the Cmd> prompt or by using the command shell(). Type help(shell). Under the real mode version, you are limited to DOS commands that require relatively little memory. These include most of the standard things like DIR, CD, TYPE, and MORE. Under the protected mode version, you can execute almost any program. In this version there is a pre-defined macro edit that uses shell() to invoke an editor (default is EDIT) allowing you to edit macros and data sets without quitting MacAnova. Type `help(edit)`. If you have a very small editor, you may be able to do the same in the real mode version. You will have to read in `edit` from the distributed macro file `MacAnova.mac` and probably need to edit it.

Command `putascii()` permits you to output arbitrary codes. For example, `putascii(7)` rings the bell or beeps.

B.6 Differences between versions

Here is a short table comparing the two versions:

	Real Mode Version	Protected Mode Version
Maximum size of object	65,000 bytes = 8,125 `REALs`	Limited only by memory and disk space
Interrupt key	Control C or Control Break	Control Break only
High resolution plots	White on black	Black on white
Graphics modes supported	CGA, EGA, VGA, 8514, and Hercules	VGA
High resolution plots when Windows is active?	Yes	No
Full range of high resolution line types?	Yes	Only two
`screendump:T` available on high resolution plots	No	Yes
Has pre-defined macro `edit`	No	Yes
Memory available for `shell()`	Very restricted	Quite large programs can be run

B.7 Other information

MacAnova is written in the C programming language. The real mode DOS version of MacAnova is compiled under version 4.5 of Borland C++. The protected mode version is compiled under a version of the Gnu C++ compiler, adapted for DOS by D. J. Delorie. Both are distributed as self extracting archives along with a self-extracting archive of auxiliary files such as `MACANOVA.HLP`..