

Scientist's Spreadsheet

User's Manual

Macintosh Version 2.17

for the 512K Macintosh[®], Macintosh Plus and Macintosh XL

March 1, 1987

by William Menke

Assistant Professor of Geological Science

Lamont-Doherty Geological Observatory of Columbia University

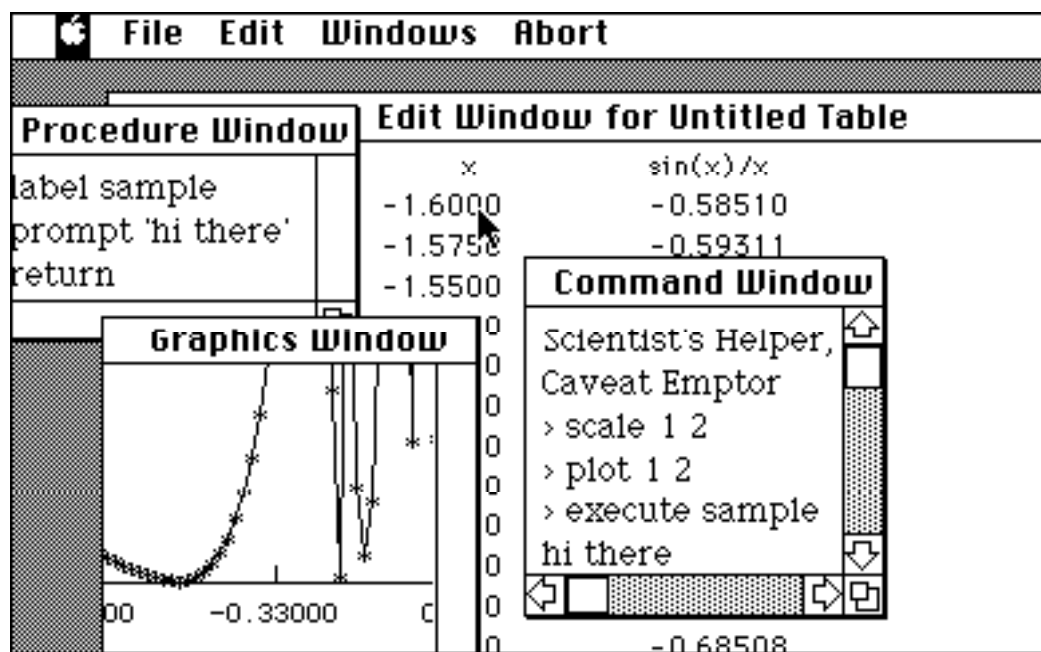
Palisades, New York 10964

(914) 359-2900

x 438

*A Data Manipulation program designed especially for
Scientists and Engineers*

*for performing
arithmetic, curve fitting and time series analysis
on tabular data*



Note: a completely compatible version of Scientist's Spreadsheet is now available for Sun 3[®] Workstations under the SunView environment.

This software and manual are public domain.

The author requests that any parties distributing modified versions of this software include a notification of modification that is visible in the Command Window when the program is launched, change the version number to x.y where x>10, and include their name and address in the manual.

Disclaimer

William Menke makes no warranty, expressed or implied, and there are expressly excluded all warranties of merchantability and fitness for a particular purpose. William Menke shall have no liability with respect to consequential, exemplary, or incidental damages arising out of or in connection with the use or performance of this software. It is the user's responsibility to assure that this software is adequate for the user's purposes.

Table of Contents

1. Introduction	4
1.1 The Scientist's Spreadsheet Table	5
1.2 Memory allocation	6
1.3 Windows	7
1.4 Menu Items	7
1.5 Commands	8
2.0 Tutorial	8
2.1 A Typical Scientist's Spreadsheet Session	8
2.2 Cookbook	9
2.3 Scientist's Spreadsheet Variables	10
2.4 Procedures	10
2.4.1 Executing Demo Procedure	10
2.4.2 Writing Procedures	10
3.0 Reference Information	12
3.1 Command Syntax:	12
3.1.1 Essential Commands	12
3.1.2 Commands that set the header	12
3.1.3 Commands for using variables	13
3.1.4 Graphics Commands	14
3.1.5 Commands for using procedures	14
3.1.6 Control commands to be used within procedures	15
3.1.7 Commands for column arithmetic and statistics	16
3.1.8 Time series analysis commands	18
3.1.9 Commands for table manipulations	20
4.0 Helpful Hints	21
5.0 Contents of Procedure Files on distribution disk	22
6.0 Known Anomalies and Caveats	24
6.1 Caveats	24
6.2 Known Anomalies	24
7.0 Technical information	25
7.1 Source code	25
7.2 Format of Ascii files	25
7.3 Format of procedure files	25
7.4 Format of binary files	25

1. Introduction. Scientist's Spreadsheet is an interactive data manipulator designed for the kind of tabular data commonly used by scientists and engineers. The fundamental data structure in Scientist's Spreadsheet is a table of numbers. These data can be input, saved as files, viewed, plotted, and operated upon mathematically.

- Scientist's Spreadsheet is basically a command string oriented program. The command set includes graphics, arithmetic, time-series analysis, curve fitting and table management operations.
- Scientist's Spreadsheet makes full use of the Macintosh windowing and menu selection functions. For instance, it contains a mouse-driven table editor to facilitate inputting, viewing, and manipulating the table.
- The contents of the table can be plotted on the Macintosh screen and then printed or saved as a MacPaint® file.
- Procedures (programs) can be written in Scientist's Spreadsheet command language and then executed as if they were Scientist's Spreadsheet commands. These procedures can use string variables and a variety of control structures.

1.1 The Scientist's Spreadsheet Table. The fundamental data structure used by Scientist's Spreadsheet is a rectangular table of numbers. Only one table can be accessed by the user at any one time.

Note about precision. All table entries are stored as single-precision (32 bit) floating point numbers in standard IEEE format. The special numbers NaN, Inf, and -Inf are supported.

In addition to the usual numbers, table entries can be set to NaN (not a number), Inf (for infinity) and -Inf. Improper mathematical operations such as division by zero will generate these entries. Scientist's Spreadsheet will not crash from an arithmetic error.

Note about entering data. Data can be entered in the table in one of three ways: typing it into the edit window, reading it in from a file previously created by Scientist's Spreadsheet, or reading it in from a TEXT file created by some other program (eg. MacWrite[®], Excel[®]).

Scientist's Spreadsheet tables come in two variations. One type is 'interpolated', meaning that the data in column 1 increase linearly with row number. Entries in column 1 of interpolated tables cannot be altered individually: only the starting value and sampling interval can be changed. The other type is 'uninterpolated', in which column 1 is no different than any other column. Interpolated tables require less disk storage than uninterpolated ones.

Note about interpolated tables Interpolated and uninterpolated tables generally act similarly, except:

- Entries in column 1 of an interpolated table cannot be changed.
- Some commands (mostly time-series analysis) work on only one kind of table.

Making interpolated and uninterpolated tables.

- By default, tables are uninterpolated.
- There are two ways to create an interpolated table:
 1. Setting the sampling interval and starting value, and then declaring the table interpolated. Column 1 of the table will immediately be overwritten to reflect the new settings, but the other columns will be unaffected. For example:

```
samp 1.0
start 0.0
interpolated true
```
 2. Interpolating a table to a given sampling interval with the interpolate or spline command. The number of rows in the table will change, and the data in all the columns will be interpolated. For example

```
interpolate 1.0
```
- An interpolated table can always be changed into an uninterpolated one with the command:

```
interpolated false
```

In addition to the table entries themselves, each table has a set of essential information, called a header, associated with it. The header is automatically updated by Scientist's Spreadsheet as the user manipulates the table, and is stored along with the table entries when a file containing a table is created.

Note about the header. The header contains the following information, referred to by standard names:

- rows:** the number of active rows in the table;
- cols:** the number of active columns in the table;
- title:** an 80 character string describing the data;
- interpolated:** a boolean flag (eg. is either 'true' or 'false') indicating whether or not the table is interpolated;
- colname:** a 10 character name for each column;
- samp, start:** the sampling interval and starting value of the table if interpolated. Col 1 is computed by the rule $\text{start} + \text{samp} \cdot (\text{row}-1)$.

The title and column names help to document the table, so the user should set them.

1.2 Memory allocation. The amount of the computer's memory allocated for the table can be varied. The default allocation when Scientist's Spreadsheet is launched is 128 rows and 2 columns. This size can be increased by the **allocate** command, up to 4096 rows and 32 columns. This should be done at the beginning of the session, since reallocating the table destroys the data in the old one (unless it is first saved in a file). Not all of the allocated rows and columns need be used at any one time. The user is free to reduce (and subsequently enlarge) the 'active size' table, using the **rows** and **cols** command (Figure 1).

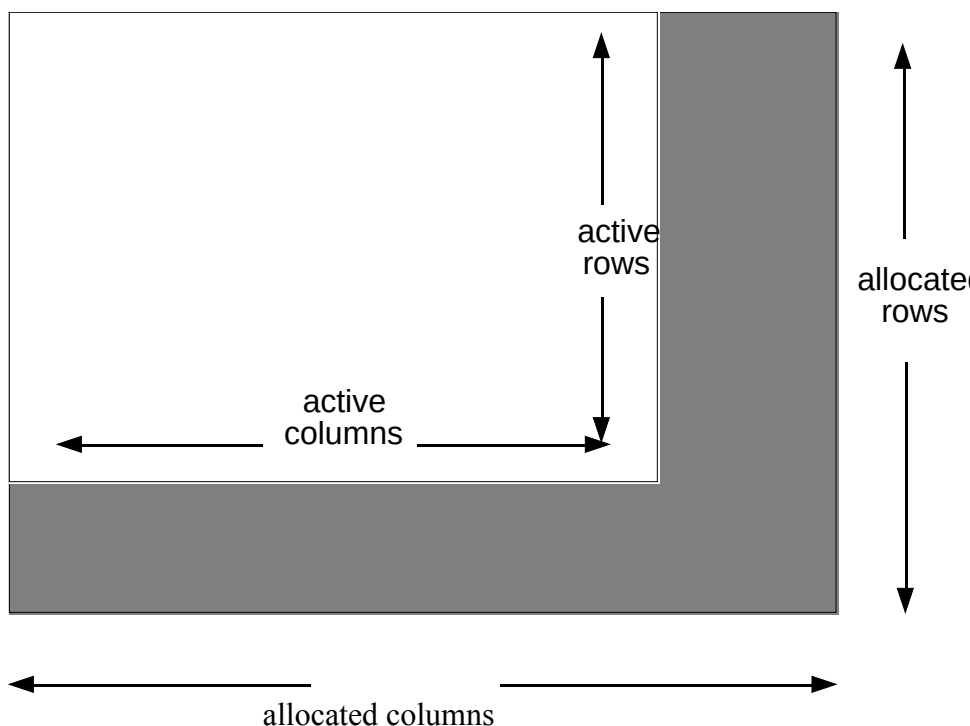


Figure 1. The active size of the table (white area) can be less than or equal to the allocated size (white and shaded area).

1.3 Windows. Scientist's Spreadsheet has four windows, named Command Window, Graphics Window, Edit Window, and Procedure Window:

The **Command Window** displays a history of what has been done during the session, and is where new commands are typed. The contents of the Command Window can be freely edited. Only text added to the last line and followed by a return is interpreted as a command.

The **Graphics Window** contains a plot of columns of the table. Plots are made by typing the appropriate commands in the command window. Clicking the mouse when the graphics window is active causes the current position of the cursor to be printed in the command window.

The **Edit Window** displays the table in tabular form and allows you to edit its entries.

The **Procedure Window** contains currently defined procedures (programs). They can be freely edited and transferred between the window and files.

1.4 Menu Items:

The **Apple** menu, through which the desk accessories can be accessed.

The **File** menu, through which all disk I/O is accomplished.

Read Binary Table reads a previously saved table (and its header) from a file on disk into memory.

Save Binary Table as... saves a table in a file on a disk. This file is in a compact binary format that cannot be read by other applications. The table's header is stored along with the table.

Read Ascii Table... reads a table stored as an ascii text file into memory.

Save Ascii Table as... saves a table in a file on a disk. This file is in a standard format that can be read by some other applications, such as MacWrite, Excel, and Edit. These files contain only the column names and table values, with the items on each line separated by tabs.

Read Procedure reads a previously saved procedure into the procedure window, appending it to the bottom of whatever text is present in the window.

Save Procedure as... saves the entire contents of the procedure window in a file on disk.

Print Graph prints the graph on the printer (either the ImageWriter or the LaserWriter, whichever was selected by the Chooser).

Save graph as ... saves the contents of the graphics window as a MacPaint file.

Quit quits from Scientist's Spreadsheet. All data and procedures not previously saved to disk are lost.

Note that tables can be stored on disks in two forms: Binary and Ascii. You should normally use the Binary form, since it is faster and the entire table header is saved. MacWrite and Excel files containing a table can be read into Scientist's Spreadsheet if they are saved in the text-only mode and read as an ascii table.

The **Edit** menu:

Undo does nothing - it is provided only for use of the desk accessories.

Cut deletes the selected text from the command, procedure and edit window.

Copy copies the selected text from the command, procedure and edit window into the clipboard.

Paste transfers the contents of the clipboard to the selected position in the command, procedure and edit windows.

The **Windows** menu, which provides a way to select hidden windows.

The **Abort** menu, which allows procedures and some commands to be aborted.

The **Miscellaneous** menu:

Position Edit Window allows you to move display a given row and column.

1.5 Commands. Many Scientist's Spreadsheet operations are invoked by typing commands in the last line of the Command Window. In some instances a command consists of just a single word. For instance, the command **clear** erases the graphics window. In other instances, a command requires one or more arguments. For instance, in the command **plot 1 2**, the '1 2' are two additional parameters or command words that specify the two columns to be plotted. Scientist's Spreadsheet has about 75 commands, each of which is described later in the manual.

Note about command words. A command word is a sequence of characters containing no blanks or a quoted sequence of characters containing blanks.

2.0 Tutorial.

2.1 A Typical Scientist's Spreadsheet Session. In the following tutorial, material Scientist's Spreadsheet types is printed in **bold**, material the user types is printed in plain text, and comments are printed in *italic*. The session creates a table that is 128 rows by 3 columns, evaluates some trigonometric functions, and plots them.

Scientist's Spreadsheet, Version 2.17, by William Menke Caveat Emptor

New Table 128 by 2

>	allocate 128 3	<i>creates 128 by 3 table</i>
>	title 'my test dataset'	<i>sets title</i>
>	colname 1 time, t	<i>labels column 1</i>
>	colname 2 0.987sin(t)	<i>labels column 2</i>
>	colname 3 sin(t)/t	<i>labels column 3</i>
>	samp 0.1	<i>column 1 is interpolated column</i>
>	start 0.0	<i>with sampling interval 0.1</i>
>	interpolated true	<i>starting value 0.0</i>
>	cfunction sin 1 2	<i>put 0.987 times sine of column 1</i>
>	cmath 2 *# 0.987 = 2	<i>into column 2</i>
>	cfunction sin 1 3	<i>put sine of column 1 divided by</i>
>	cmath 3 / 1 = 3	<i>column 1 into column 3</i>
>	table 1 3 1.0	<i>sin(0)/0 now set to NaN</i>
		<i>reset to 1</i>
>	xaxis 0 12.8	<i>set abscissa of plotting screen</i>
>	yaxis -1 1	<i>set ordinate of plotting screen</i>
>	clear	<i>clear graph</i>
>	plot 1 2 solid	<i>plot column 2 against 1</i>
>	plot 1 3 dotted	<i>plot column 3 against 1</i>
>	quit	<i>quit from Scientist's Spreadsheet</i>

2.2 Cookbook.

2.2.1. Plot data in column 3 against data in column 4:

clear
scale 3 4
axes
plot 3 4

2.2.2. Create a table with time, t, sampled every 0.01 seconds in column 1, and sin(t) in column 2:

samp 0.01
start 0.0
interpolated true

cfunction sin 1 2

2.2.3. Evaluate the function $z=\sin^2(x+y)$ where x is in column 1, y is in column 2, and z is to be put in column 3.

```
cmath 1 + 2 = 3  
cfunction sin 3 3  
cmath 3 * 3 = 3
```

2.2.4. Create a table with time, t, sampled every 0.01 seconds in column 1, and a unit spike, $\delta(t-0.63)$, in column 2. Then bandpass the spike between 20 and 30 hertz.

```
samp 0.01  
start 0.0  
interpolated true  
constant 0 2  
table 64 2 1.0  
bandpass 20.0 30.0 2 2
```

2.2.5. Convert an uninterpolated dataset into an interpolated one, remove a linear trend and then compute and plot its amplitude spectrum. Assume that the table is uninterpolated, column 1 is time sampled about once every second, and that the time series is in column 2.

```
interpolate 1.0  
trend remove 1 2 2  
spectrum amplitude  
scale 1 2  
clear  
axes  
plot 1 2
```

2.3 Scientist's Spreadsheet Variables. Variables contain character strings of up to 80 characters in length. A variable's name can be any character string that does not contain a blank. Once defined (eg. by the **setvar** command), the value of a variable can be used as a command word by including its name in the command string prefaced by the **@** symbol. For example, the commands:

```
setvar date 'November 1, 1755'  
prompt @date
```

create a variable named 'date' that is set to the value 'November 1, 1755', and then types the value of the variable in the command window. Note that variables can contain numbers in string form. Variables are mainly useful in procedures. In addition to variables defined by the user, Scientist's Spreadsheet also defines and automatically updates variables set to commonly used parameters:

The header variables: **rows, cols, title, interpolated, samp, start**

The arguments of the last procedure to be executed: **arg1, arg2, arg3, arg4**

The endpoints of the graphics axes, **xmin, xmax, ymin, ymax**

The position of the cursor in user coordinated after the last cursor

command:

xpos, ypos

The minimum value in a column after the min command: **min**

The maximum value in a column after the max command: **max**

The mean and standard deviation after a mean command: **mean, stddev**

The slope, intercept, and one-standard-deviation errors after a trend

command: **slope, intercept, errslope, errintercept**

The number of non-NaN data used in computations during commands such as **min,**
max, mean, trend, polyfit, multifit, etc: counts

For example, in order to print out the current value of the graphics x-axes, type the commands:

```
prompt @xmin
prompt @xmax
```

2.4 Procedures. The user can write short programs, or 'procedures' consisting of sequences of Scientist's Spreadsheet commands, variable definitions and references, and control structures. Procedures are first written in the procedure window or read into Scientist's Spreadsheet using the **Read Procedure** item in the File menu. They can then be run using the **execute** command (abbreviated **x**). Three files of procedures, Demo Procedure, Useful Procedures and Ternary Diagram are included on the distribution disk. Useful Procedures and Ternary Diagram are described later in the manual.

2.4.1 Executing Demo Procedure:

1. Launch Scientist's Spreadsheet by double-clicking on its icon.
2. Select Read Procedure from the File Menu.
3. When the Read File dialog box appears, select and open Demo Procedure.
4. Activate the Procedure Window, and note that the Demo procedure is present.
5. Activate the Command Window and type **x demo** in the bottom line, and follow the

directions that will appear. The procedure will create a table of data and plot it, and then allow you to measure points from the graphics window by clicking the cursor.

2.4.2 Writing Procedures. The following sample procedure squares column 1, adds it to column 2, and puts the results in column 3:

```
label add
cmath 1 * 1 = 1
cmath 1 + 2 = 3
return
```

This procedure is executed by typing on the last line in the Command Window:

```
x add
```

Procedures can get input from the keyboard. The above example can be modified to ask for a result column:

```
label add
input result 'enter result column'
cmath 1 * 1 = 1
cmath 1 + 2 = @result
return
```

This procedure is executed by typing

```
x add
```

Another way for a procedure to get information is through arguments entered in the command line.

```
label add
cmath 1 * 1 = 1
cmath 1 + 2 = @arg1
return
```

This procedure is executed by typing

```
x add 3
```

(where 3 is the result column).

Procedures can call other procedures:

```
label addAndSquare
```

```
x add
cmath 3 * 3 = 3
return
```

```
label add
cmath 1 * 1 = 1
cmath 1 + 2 = 3
return
```

Procedures can contain loops. The following procedure plots columns 2, 3, ... against column 1.

```
label plotAllColumns
for column 2 @cols
plot 1 @column
next column
return
```

Procedures can contain conditional statements. The following procedure plots columns 2, 3 ... against 1, querying each time whether to discontinue plotting.

```
label plot1
for column 2 @cols
plot 1 @column
input query 'continue? y or n'
if @query s= 'n'
    return
next column
return
```

3.0 Reference Information.

3.1 Command Syntax: Command keywords are printed in **bold**, arguments in plain text. *Italicized* command words may be omitted.

3.1.1 Essential Commands:

quit
allocate¹ **maximum_rows** **maximum_columns**

3.1.2 Commands that set the header:²

colname	column_number	any_string
cols	number_of_cols	
interpolated	(<i>true</i> false	
rows	number_of_rows	
samp	sampling_interval	
start	starting_value	
title	any_string	

- Notes.
- 1. Maximum table size is 4096 by 32, although in practice this is less and depends on the amount of memory on the system.
 - 2. Commands that change the header automatically update the header variables.

3.1.3 Commands for using variables:¹

delete	variable	variable_name
input	variable_name	prompt_string
prompt	any_string_1	... any_string_5
set	variable_name	$\begin{pmatrix} \text{table} & \text{row} & \text{col} \\ \text{colname} & \text{col} & \\ \text{coefficient}^2 & n & \end{pmatrix}$
setvar	variable_name	any_string
type	variables	
vfunction	$\begin{pmatrix} \sin \\ \cos \\ \tan \\ \text{asin} \\ \text{acos} \\ \text{atan} \\ \text{sqrt} \\ \ln \\ \exp \\ \text{erf} \\ \text{erfc} \\ \text{abs} \\ \text{int} \\ \text{frac} \\ \text{denan}^3 \\ \text{row}^3 \end{pmatrix}$	input_value variable_name
vmath	input_value_1	$\begin{pmatrix} + \\ - \\ * \\ / \end{pmatrix}$
concat	variable_name	any_string_1 any_string_2 any_string_3

Notes:

1. A variables name (any string containing no blanks) is distinct from its value, which is its name preceeded by the symbol @.
2. The coefficients are those determined by least-squares methods using the polyfit and multfit commands.
3. The **denan** function converts NaN's into zeroes. The **row** function returns the row number of a value in col 1 of an interpolated table, based on the current value of samp and start. The row is always in the range $1 \leq \text{row} \leq \text{rows}$.

3.1.4 Graphics Commands:

anotate¹ x y any_string
axes²
clear³
cursor⁴ any_prompt_string
hbars⁵ abccissa_col ordinate_col bar_half_width start_row end_row

plot abccissa_col ordinate_col $\left(\begin{array}{l} \text{solid} \\ \text{dotted} \\ \text{dashed} \\ \text{bold} \\ \text{dots} \\ \text{circles} \\ \text{crosses} \\ \text{stars} \\ \text{x} \end{array} \right)$ start_row end_row

scale⁶ abccissa_col ordinate_col
xaxis minimum_value maximum_value
yaxis minimum_value maximum_value
vbars⁵ abccissa_col ordinate_col bar_half_height start_row end_row

vector x₁ y₁ x₂ y₂ $\left(\begin{array}{l} \text{solid} \\ \text{dotted} \\ \text{dashed} \\ \text{bold} \\ \text{dots} \\ \text{circles} \\ \text{crosses} \\ \text{stars} \\ \text{x} \end{array} \right)$

3.1.5 Commands for using procedures:

beep
execute⁷ label
sleep number_of_seconds
refresh⁸ $\left(\begin{array}{l} \text{true} \\ \text{false} \end{array} \right)$

Notes.

1. Plots the given character string in the graphics window at position (x,y)
2. Clears (erases) the graphics screen.
3. Used to pick values from a plot using the mouse. The variables xpos and ypos are set to the cursor position when the mouse is clicked. If the prompt string is not given, no prompt is printed.
4. Sets the xaxis and yaxis values on the basis of the data in two columns.
5. Horizontal and vertical error bars.
6. Plots axes.
7. Automatically updates variables arg1, arg2, arg3, arg4.
8. Suspends or restarts refreshing of the edit window during the execution of a procedure (so the procedure will run faster). This command can only be used within a procedure. A return to the command window always restores the refresh.

3.1.6 Control commands to be used within procedures:

label		any_string
for ¹	variable_name	starting_integer ending_integer <i>increment</i>
next ¹		variable_name
goto		label
return		

if ²	any_string_1	$\left(\begin{array}{c} s= \\ s<> \\ = \\ < \\ > \\ <= \\ >= \end{array} \right)$	any_string_2 <i>num_of_lines</i>
------------------------	--------------	--	----------------------------------

Notes.

1. The **for-next** statement pair are used for loops, the **for** statement being placed at the beginning of the loop and the **next** being placed at the end. Loops can be nested.
2. The **if** statement permits the following *num_of_lines* to be executed only if the comparison is true. The comparisons `s=` and `s<>` test the ascii representation of strings. The comparison `=`, `<>`, `<`, `>`, `<=`, and `>=` assume that the strings contain floating point numbers and tests the value of these numbers. When testing for the presence of a NaN, the string comparisons should be used.

3.1.7 Commands for column arithmetic and statistics:

cmath	input_col_1	$\begin{pmatrix} + \\ - \\ * \\ / \end{pmatrix}$	input_col_2 = output_col
cmath	constant	$\begin{pmatrix} \# + \\ \# - \\ \# * \\ \# / \end{pmatrix}$	input_col = output_col
cmath	input_col	$\begin{pmatrix} + \# \\ - \# \\ * \# \\ / \# \end{pmatrix}$	constant = output_col
cfunction		$\begin{pmatrix} \sin \\ \cos \\ \tan \\ \operatorname{asin} \\ \operatorname{acos} \\ \operatorname{atan} \\ \operatorname{sqrt} \\ \ln \\ \exp \\ \operatorname{erf} \\ \operatorname{erfc} \\ \operatorname{abs} \\ \operatorname{int} \\ \operatorname{frac} \\ \operatorname{denan}^1 \\ \operatorname{row}^1 \end{pmatrix}$	input_col output_col
constant ²	value	<i>col_number</i>	<i>first_row</i> <i>last_row</i>
mean ³	$\begin{pmatrix} \text{type} \\ \text{keep} \\ \text{remove} \\ \text{compute} \end{pmatrix}$	$\begin{pmatrix} \text{input_col} \\ \text{input_col} \\ \text{input_col} \\ \text{input_col} \end{pmatrix}$	$\begin{pmatrix} \\ \text{output_col} \\ \text{output_col} \\ \end{pmatrix}$
trend ⁴	$\begin{pmatrix} \text{type} \\ \text{keep} \\ \text{remove} \\ \text{compute} \end{pmatrix}$	$\begin{pmatrix} \text{input_col_1} \\ \text{input_col_1} \\ \text{input_col_1} \\ \text{input_col_1} \end{pmatrix}$	$\begin{pmatrix} \text{input_col_2} \\ \text{input_col_2} \\ \text{input_col_2} \\ \text{input_col_2} \end{pmatrix}$ $\begin{pmatrix} \\ \text{output_col} \\ \text{output_col} \\ \end{pmatrix}$
min ⁵ max ⁵ sort	col_number col_number col_number		

polyfit ⁶	order remove	$\left(\begin{array}{lll} \text{type} & \text{input_col_1} & \text{input_col_2} \\ \text{keep} & \text{input_col_1} & \text{input_col_2} \\ \text{input_col_1} & \text{input_col_2} & \text{output_col} \\ \text{compute} & \text{input_col_1} & \text{input_col_2} \end{array} \right)$
multifit ⁷		$\left(\begin{array}{lll} \text{type} & \text{independent_col_list} & \text{dependent_col} \\ \text{keep} & \text{independent_col_list} & \text{dependent_col} \\ \text{remove} & \text{independent_col_list} & \text{dependent_col} \\ \text{compute} & \text{independent_col_list} & \text{dependent_col} \end{array} \right)$

Notes.

1. The **denan** function converts NaN's to zero. The **row** function returns the row number of a value in col 1 of an interpolated table, based on the current value of samp and start. The row is always in the range $1 \leq \text{row} \leq \text{rows}$.
2. The constant command sets the rows of a column between starting_row and ending_row to the given constant value. If the row limits are omitted, they default to 1 and rows respectively.
3. The mean command computes the mean of a column and updates the variables mean, stddev and counts (the number of non-NaN column entries). 'type' types the result in the command window, 'keep' puts it in a column, 'remove' subtracts it from a column, and 'compute' has no action except updating the header variables.
4. Trend computes a least-squares fit between two columns and updates the header variables slope, intercept, errslope, errintercept, counts. See note 3 for explanation of keywords.
5. The commands min and max automatically update the variables min, max, counts.
6. Least-squares fit of polynomials of order 1-6. This command resets the variable counts. Variables can be set to the values of the coefficients with the set command. See note 3 for explanation of keywords.
7. Least-squares linear multivariate regression. See note 3 for explanation of keywords. This command resets the variable counts. Variables can be set to the values of the coefficients with the set command. The independent column list consists of column numbers separated by commas, spaces, or tabs. If spaces or tabs are used, the list must be surrounded by quotes. For example:

multifit type 1,2,3,4 5 6

3.1.8 Time series analysis commands:

bandpass ¹	low_frequency	high_frequency	input_col_num	output_col
circonv ²	input_cols_1	input_col_2	output_col	
coherence ³	frequency_smoothing_interval			
convolve ⁴	input_col#	operator_col_#	operator_length	output_col_#
differentiate ⁵	x_col_num	y_col_num	output_col_num	
histogram ⁶	minimum_value	maximum_value	number_of_bins	
integrate ⁷	x_col_num	y_col_num	output_col_num	

laplace⁸ $\left(\begin{array}{llll} \text{forward} & \text{starting_s} & \text{delta_s} & \text{new_rows} \\ \text{inverse} & \text{starting_t} & \text{delta_t} & \text{new_rows} \end{array} \right)$

noise ⁹	mean	std_deviation	output_col_num	
phaseshift ¹⁰	angle_in_radians	input_col_num	output_col_num	
reverse ¹¹	input_col_number	output_col_number		
sum ¹²	input_col_num	ouput_col_num		

spectrum¹³ $\left(\begin{array}{c} \text{amplitude} \\ \text{power} \\ \text{phase} \end{array} \right)$

taper¹⁴ $\left(\begin{array}{lllll} \text{ascending} & \text{x_col} & \text{output_col} & \text{start_row} & \text{end_row} \\ \text{descending} & \text{x_col} & \text{output_col} & \text{start_row} & \text{end_row} \end{array} \right)$

Notes.

1. Interpolated table only. Second order Chebyshev recursive filter. Will phase shift to larger times.
2. Circular convolution of two time series with Fast Fourier Transform. Input columns are padded with zeroes out to a power of two in length, FFT'ed, multiplied, and the result inverse FFT'ed. The result is truncated to the length of the table. Table must be interpolated.
3. Interpolated table only. Coherence between time series $x(t)$ and $y_i(t)$ where $x(t)$ is in column 2 and $y_i(t)$ is in cols 3, 4, ... cols. Column 1, t is replaced by frequency, f (Hertz), column 2 by $|x(f)|^2$, and the rest of the columns by coherence $|\langle x(f)y_i^*(f) \rangle|^2 / \langle |x(f)|^2 \rangle \langle |y_i(f)|^2 \rangle$, where $\langle \rangle$ means boxcar smoothing.
4. Interpolated table only. Brute-force convolution; operator length better be short. Points off the beginning of the input column are assumed to be zero.
5. Differentiation, dy/dx by first order finite differences $(dy/dx)_i = (y_{i+1} - y_i) / \Delta x$. The last row is set to NaN.
6. Interpolated table only. Columns 2 through cols are replaced with histograms of their previous values, and column 1 is replaced with the bin positions (centers).
7. Integration, $\int_0^x y(x') dx'$ by trapezoidal rule, with an integration constant of zero.
8. Interpolated table only. Laplace transform and its inverse. Forward transform does $f(s) = \int_0^\infty f(t) \exp(-st) dt$ by trapezoidal rule and is quite slow. Column one is assumed to be t , and columns 2...cols are several $f(t)$'s. The entire table is replaced with its transform, column 1 being replaced by s . Inverse transform is by Stehfest's (1969) Algorithm 368, and is rather unstable, so results must be treated with some skepticism. As a rule of thumb, $f(s)$ must be accurate between $0.6/t$ and $5.6/t$ to get sensible results.
9. Gaussian random noise.
10. Interpolated table only. Phase shift is defined so that the Hilbert transform is a phase shift of 90° . The data is padded with zeroes to a power of two ≥ 128 in length, so that the fast fourier transform algorithm can be used. The phase of the zero and nyquist frequencies are not changed.
11. Reverse the order of elements in a column.
12. Running sum of input column values.
13. Interpolated table only. Spectrum using the fast fourier transform. Columns 2 through cols are padded with zeroes to the nearest power of two ≥ 128 in length, then transformed. Column 1 is changed to frequency (Hertz). Table must be interpolated.
14. The column is multiplied by a half-wavelength cosine taper that rises from zero to one or falls from one to zero. Portions of the column outside of the ends of the taper are unaffected.

3.1.9 Commands for table manipulations:

copy	$\left(\begin{array}{lll} \text{col} & \text{input_col} & \text{output_col} \\ \text{row} & \text{input_row} & \text{output_row} \end{array} \right)$
swap	$\left(\begin{array}{lll} \text{col} & \text{input_col} & \text{output_col} \\ \text{row} & \text{input_row} & \text{output_row} \end{array} \right)$
insert	$\left(\begin{array}{lll} \text{col} & \text{input_col} & \text{number_of_cols} \\ \text{row} & \text{input_row} & \text{number_of_rows} \end{array} \right)$
delete	$\left(\begin{array}{lll} \text{col} & \text{input_col} & \text{number_of_cols} \\ \text{row} & \text{input_row} & \text{number_of_rows} \end{array} \right)$
type	col col_number
table	$\left(\begin{array}{ll} \text{row_number} & \text{col_number}^1 \\ \text{row_number} & \text{col_number value}^2 \end{array} \right)$
interpolate ³	sampling_interval
spline ⁴	sampling_interval

Notes:

1. Writes the current table value in the command window.
2. Sets the given table entry.
3. Linear interpolation of an uninterpolated table to an interpolated one. Table values that are set to NaN are ignored. The number of rows of the table is changed. If the row allocation is not large enough, the end of the data will be lost.
4. Natural cubic spline interpolation of an uninterpolated table to an interpolated one. Table values that are set to NaN are ignored.

4.0 Helpful Hints:

- 1) **Allocate** a table large enough for all your needs, right at the beginning of a Scientist's Spreadsheet session. Then, if you want to work with a smaller table, just declare it to be smaller using the **rows** and **cols** commands. Doing this usually speeds things up, since re-allocating space is time consuming.
- 2) Decreasing the active size of a table with the **rows** and **cols** command does not actually destroy any data. One can always recover the data by increasing the table size. This fact allows one to merge two tables stored in two different files: **allocate** a large table with enough columns to hold both tables, **read** in the first table, increase the number of active columns with the **cols** command and **copy** the columns to the right hand part of the table. Then **read** in the second table and increase the table size using the **cols** command, thus recovering the data from the first table.
- 3) Material from the Command Window can be **copied** and **pasted** into the procedure window, and quickly edited into a short procedure.
- 4) When reading an **ascii table** into MacWrite, choose the option whereby MacWrite interprets carriage returns as paragraphs. Then put enough tabs in the ruler so that all the columns line up properly.
- 5) If you transfer data to the Macintosh from another computer using MacTerminal[®], you will lose any tabs between the columns (MacTerminal converts them to spaces). Therefore, don't send tabs, send one space between each entry. Then use MacWrite to change each space to a tab. You can't type a tab into the change dialog box, but you can paste it in.
- 6) Remember that windows (and in particular the **graphics window**) can be plotted on the printer by typing Command-Shift-4.
- 7) If you're running on a system with very limited memory, occasionally clear the command window using the **clear** item in the **edit** menu.
- 8) A running procedure will update the edit window whenever one of its component commands alter the table. This can sometimes be helpful when debugging a procedure, since the user can watch the calculations as they proceed. However, the updating is quite time consuming. One way to speed things up is to close up the edit window to its minimum size before running the procedure. Another is to place the command **refresh false** within the procedure.

5.0 Contents of Procedure Files on distribution disk:

In Useful Procedures:

x arrow line_type

Plots an arrow in the graphics window with head and tail at cursor clicks. Line_type should be solid, bold, dotted, or dashed.

x autocorrelation

Autocorrelation of time series in column 2 (using the conv command). Table allocation must be at least 4 cols and at least twice the currently defined number of rows. The table is destroyed. Results returned in col 2.

x clip min_value max_value input_col output_col

Clips the data in a column.

x crosscorrelation

Crosscorrelation of time series in columns 2 and 3 (using the conv command). Table allocation must be at least 4 cols and at least twice the currently defined number of rows. The table is destroyed. Results returned in col 2.

x drawcurve

Lets the user construct a curve by clicking the mouse in the graphics window. The procedure loops until the column is filled or it is aborted from the abort menu.

x drawhist ordinate_col

Draws a histogram with vertical bars. To be used after histogram command.

x letter any_string

Writes the string in the graphics windows when the mouse is clicked.

x logplot

Log-log plot. Prompts for input.

x movingaverage

Moving average of a dataset with a triangular averaging function. Prompts for input. Requires one extra column for temporary results.

x naner ordinate-col

Lets the user click on a plot of a column to set its values to NaN. Good for throwing away unwanted data points. The procedure loops until it is aborted from the abort menu.

x numberplot abscissa_col ordinate_col plotting_symbol

Plots data with the points numbered sequentially.

x plot1 symbol second_symbol

Plots columns 2 ... cols vs. col 1, pausing and clearing the screen between each plot. Will plot a second symbol if given, eg. **x plot1 solid stars** makes plots with stars connected by straight lines.

x plotall symbol second_symbol

Plots all columns against one another, pausing and clearing the screen between each plot. Will plot a second symbol if given, eg. **x plotall solid stars** makes plots with stars connected by straight lines.

x pointer any_string line_type

Plots a labeled arrow in the graphics window with head and tail at cursor clicks. The arrow's tail is labeled with any_string. Line_type should be solid, bold, dotted, or dashed.

x pt abscissa_col_number ordinate_col_number symbol

clears, scales and plots

x rfunction function_type input_row output_row

Function applied to a row of the table. See cfunction command for function_types.

x rmath input_row_or_constant operator input_row_or_constant output_row

Arithmetic applied to a row. See cmath command for allowed operators. Note that equal sign is omitted, in contrast to vmath and cmath commands.

x scroller ordinate_col screen_width

The user can scroll the plot left and right by clicking to the left or right of center. The procedure loops until it is aborted from the abort menu.

x semilogplot

Semi-log plot. Prompts for input.

x separate input_col output_col of > values output_col of > values test_value

Divides the data in a column into two groups, depending on whether they are greater than or less than a given test value. Each group is put in a separate output column.

Output columns are padded with NaN's.

x stickplot abscissa_col ordinate_col

Plots a column with vertical bars and a zero baseline.

x transpose

Transposes the rows and columns of a table.

x zapper ordinate_col

Lets the user click on a plot of a column to set its values to the ordinate_value of the click. Good for editing data points. The procedure loops until it is aborted from the abort menu.

x zerophasebandpass low_frequency high_frequency input_col output_col

Zero phase bandpass with roll-off twice as fast as **bandpass** command.

In Ternary diagram: A set of procedures to plot ternary diagrams (used in geochemistry, etc.). The table must be at least 5 columns wide with the fractions of end members A, B, C, in columns 1, 2, 3 (although column 3 is computed automatically as 1-A-B.. Columns 4 and 5 are used for temporary results, so don't put data in them.

x plot-triangle

Draws the triangle on the graphics screen, with the vertices labeled with the column names.

x plot-ternary plotting-symbol

Recomputes C as 1-A-B and plots the data. The plotting symbol should be one of the usual ones for the plot command, and defaults to x if omitted. If you don't like the shape of the triangle, change the yaxis command in this procedure to suit your fancy.

x pick

Activates the cursor and prints out A,B,C values. The procedure loops until it is aborted from the abort menu.

6.0 Known Anomalies and Caveats:

6.1 Caveats.

1. Accuracy of calculations.

Any program that performs numerical calculations must trade off accuracy and speed of computation. While the many numerical algorithms that are incorporated in Scientist's Spreadsheet were chosen with care, and to the author's knowledge are sufficiently accurate for the needs of most scientists and engineers, instances will arise when a particular algorithm fails to give sufficiently accurate results for a particular use. For the most part these failures occur for pathological cases well known in numerical analysis: in curve-fitting when the normal equations are nearly underdetermined, in integral transforms when the integrand is too oscillatory or has singularities, *etc.* The user has the responsibility to check for these cases. A computer program is no substitute for knowledge about the techniques that one is using, and their limitations.

6.2 Known Anomalies.

1. Treatment of NaN's.

a) Keyboard entries and all transcendental functions generate NANDIV. This was done because the Manx floating point library generates error codes, not NaN's. Elementary arithmetic generate IEEE standard NaN's.

b) The **if** statement floating-point comparisons don't work well with NaN's. When testing for NaN's, use a string comparison.

2. Memory usage.

a) Scientist's Spreadsheet will crash if it runs out of memory. Some actions, such as **allocate**, check to see if enough memory is available, and return an error message if it isn't. But a system-generated memory error, such as from the Resource Manager, leads to a crash. This usually isn't a problem on systems with lots of RAM, except if you've reserved a lot of memory for a RamDisk.

b) The entire code for Scientist's Spreadsheet is locked into memory when the program is loaded.

3. Edit Window.

a) The edit window is crude, to say the least. The refresh is too slow, the scroll is always based on a 4096 by 32 table, etc.

b) If you scroll a newly input table entry out of the window, or execute a command, before hitting a return or clicking the mouse, you usually lose the value you just input.

4. Sort command. Nan's and Inf's are not discriminated during the sort.

7.0 Technical information.

7.1 Source code. Scientist's Spreadsheet is written in the C programming language, and compiled under Manx Aztec C version 1.06G. A complete copy of the source code can be obtained by sending an initialized diskette and a stamped return mailer to the author.

7.2 Format of Ascii files. Scientist's Spreadsheet Ascii files have type TEXT. The first line contains the column names, separated with tabs. Subsequent lines contain rows of the table, with entries separated by tabs. Each line is terminated with a return.

7.3 Format of procedure files. Scientist's Spreadsheet procedure files have type TEXT. Each line of the procedure is terminated with a return.

7.4 Format of binary files. The following C code demonstrates how a binary file is read:

```
#define cmdWordLen 80
typedef char cmdWS[cmdWordLen];
typedef char colStr[12];

struct headerRec {
    cmdWS title;
    int interpolated, rows, cols, maxRows, maxCols;
    float start, samp;
    colStr colName[32];
    char hEndArray[30]; /*fills out header to 512 bytes*/
};
typedef struct headerRec headerRec;

typedef float TArray[];
typedef TArray *TArrayPtr;
typedef TArrayPtr *TArrayHdl;

struct tableRec {
    headerRec header;
    TArrayHdl ptr[32];
};
typedef struct tableRec tableRec;

tableRec          table;
int               i, j, k, f, firstCol, oldMaxRows, oldMaxCols;
long              count, bytes;
Finfo             info;

/* note that a binary file has type RGTB */

oldMaxRows = table.header.maxRows;
oldMaxCols = table.header.maxCols;

count=512L;
FSRead(f,&count,&(table.header));

table.header.maxCols = oldMaxCols;
table.header.maxRows = oldMaxRows;

if (table.header.interpolated) {
```

```

    firstCol=1;
    /* and you must build column 0 yourself*/
    /* from the rule value=start+i*samp */
    }
else {
    firstCol=0;
    }

bytes = (long)table.header.rows*4L;
for( j=firstCol; j<table.header.cols; j++ ) {
    HLock(table.ptr[j]);
    count = bytes;
    k=FSRead(f,&count,*(table.ptr[j]));
    HUnlock(table.ptr[j]);
} /*end for*/

```