

**graph** Manual  
V2.00.000



# Contents

<b>Table of Contents</b>	<b>1</b>
<b>List of Figures</b>	<b>7</b>
<b>List of Tables</b>	<b>9</b>
<b>1 Introduction</b>	<b>11</b>
1.1 Goals . . . . .	11
1.2 Operation . . . . .	12
1.3 Plots . . . . .	12
1.3.1 Bar Graphs . . . . .	12
1.3.2 Box Graphs . . . . .	13
1.3.3 Line Graphs . . . . .	13
1.3.4 Circles . . . . .	13
1.3.5 Maps . . . . .	13
1.3.6 Pie Charts . . . . .	13
1.3.7 Scatter Plots . . . . .	13
1.3.8 Surfaces . . . . .	13
1.3.9 General Graphics . . . . .	13
1.4 Tasks . . . . .	13
1.4.1 Sorting . . . . .	14
1.4.2 Polynomial Curve Fitting . . . . .	14
1.4.3 Data Formats . . . . .	14
1.4.4 Output Formats . . . . .	14
<b>2 Parameters and Expressions</b>	<b>15</b>
2.1 Primitives . . . . .	15
2.2 Syntax . . . . .	16
2.3 Arithmetic . . . . .	16
2.3.1 Addition . . . . .	16
2.3.2 Other Operations . . . . .	17
2.4 Functions . . . . .	18
2.4.1 ABS . . . . .	18
2.4.2 AVERAGEDEVIATION . . . . .	18

2.4.3	INDEXED	18
2.4.4	LOG	19
2.4.5	LOG10	19
2.4.6	LOG2	19
2.4.7	MEAN	19
2.4.8	MEDIAN	20
2.4.9	MAX	20
2.4.10	MIN	20
2.4.11	ROUND	20
2.4.12	SIZE	20
2.4.13	SUM	20
2.4.14	STANDARDDEVIATION	20
2.4.15	VARIANCE	21
2.5	Variables	21
<b>3</b>	<b>Commands</b>	<b>23</b>
3.1	Axes	23
3.2	Background Grid	24
3.3	BAR	26
3.4	BAR3D	30
3.5	BOX	35
3.6	BMP	41
3.7	BUCKETS	42
3.8	CIRCLES	45
3.9	COLOR	50
3.10	COLORMAP	52
3.11	CONTOUR	53
3.12	DISPLAY	61
3.13	DRAW ARROW	62
3.14	DRAW BEZIER	64
3.15	DRAW LINE	66
3.16	DRAW TEXT	68
3.17	GIF	71
3.18	IMAGE	72
3.19	LEGEND	73
3.20	LINE	76
3.21	LLSQ	79
3.22	PIE	81
3.23	PLACE	85
3.24	PS	87
3.25	READBIN	88
3.26	READCSV	90
3.27	READONED	92
3.28	SCATTER	93
3.29	SHOW	100
3.30	SORT	102

<i>CONTENTS</i>	5
3.31 SURFACE . . . . .	104
3.32 SYMBOL . . . . .	110
3.33 TRANSFORM . . . . .	112
<b>4 Graphics</b>	<b>115</b>
4.1 Color Maps . . . . .	115
4.2 Line Styles . . . . .	116
<b>List of References</b>	<b>119</b>
<b>Index</b>	<b>119</b>



# List of Figures

3.1	Bar graph. . . . .	28
3.2	Multi-colored bar graph. . . . .	29
3.3	3D Bar graph. . . . .	32
3.4	Multiple bars on a single plot. . . . .	34
3.5	Multiple box plots. . . . .	39
3.6	Box plot with LLSQ line fit. . . . .	40
3.7	Results of bucketizing data . . . . .	44
3.8	CIRCLES command with sample output . . . . .	49
3.9	Bilinear Interpolation . . . . .	54
3.10	Basis computation. . . . .	54
3.11	Wave tank contour plot. . . . .	58
3.12	Contour plot with elevation coloring . . . . .	59
3.13	Contour plot with slope shading . . . . .	60
3.14	An arrow . . . . .	62
3.15	DRAW ARROW with curves . . . . .	63
3.16	Bezier curves making a callout cloud . . . . .	65
3.17	DRAW LINE example with square . . . . .	67
3.18	DRAW TEXT example . . . . .	70
3.19	Legend construction and display . . . . .	75
3.20	Multiple line graphs. . . . .	78
3.21	Linear Least Squares fit to partial COS curve . . . . .	80
3.22	Pie chart example . . . . .	84
3.23	Two graphs in one using PLACE . . . . .	86
3.24	Scatter plot. . . . .	96
3.25	Using symbol size to depict extra information . . . . .	97
3.26	Polynomial regression curve for random data. . . . .	98
3.27	Polynomial regression curve for random data, too high a degree. . . . .	99
3.28	Flat shading 3D surface plot . . . . .	107
3.29	Elevation color shading surface plot (flat, no boundaries) . . . . .	108
3.30	Diffuse color shading with light source pointing down . . . . .	109
3.31	Using the LOG transforms for line graphs. . . . .	113
4.1	Default color map . . . . .	115
4.2	Different linestyles and their STYLE values . . . . .	118





# List of Tables

2.1	Addition type coercion table . . . . .	17
2.2	Type coercions between other arithmetic operators. . . . .	18
3.1	Box plot whisker range values . . . . .	36
3.2	The default color map. . . . .	50
3.3	Sun angles for slope shading . . . . .	55
3.4	OS default display routines . . . . .	61
3.5	Defined fonts. . . . .	68
3.6	Binary File Types . . . . .	88
3.7	Scatter plot symbols . . . . .	94
3.8	Shading values for different surface displays . . . . .	106



# Chapter 1

## Introduction

The **graph** program implements the Tripline data mining effort visualization function. It accepts “programs” to generate one or more graphics as the result of an analysis. These might be better termed “scripts” since the commands are executed one at a time and there are no loops.

### 1.1 Goals

The design goals are simplistic:

1. **Functionality** The program provides general purpose visualization for two and three dimensional data.
2. **User Friendliness** None. Control is through other programs, perhaps browser based. Installation is extremely simple: there must be:
  - (a) No required libraries that aren't part of the basic system.
  - (b) Can be part of some other installation.
  - (c) No required changes to execution path, installation of configuration files or the like.
  - (d) Source code is complete. The only system requirements are the standard C libraries for I/O and basic mathematical functions. For obscure reasons Microsoft doesn't implement LOG2 so you have to do without on such machines.
  - (e) An error causes the program to exit without continuing.
3. **Documentation** All commands, options, and operations are documented. Program code documentation is mostly to NATO standards.
4. **Ease of Programming** Typically only one graph is generated. No data structures are ever freed, no garbage is ever collected. The program is not suited for embedded systems or those with limited memory.

5. **Portability** The program has been successfully tested under 32 and 64 bit Linux, Windows XP, Windows 7, and MacOSX.

## 1.2 Operation

The single program `graph` (or in Windows `graph.exe`) is placed on the standard system search path or you must provide its full path. The single command line argument is the name of a script file that contains commands, and perhaps data, to create a graph, map, or other visualization. If the command line argument is `-`, then input comes from `stdin`; it is not interactive. Additional arguments can be included and all are placed in the global variables `ARGVn` as strings.

```
graph test.graph
```

Typically, execution will be followed by echoes of the the input file with some rudimentary information about the line being executed. For example:

```
graph V1.00.009
1: READONED solitaire="solitaire.dat"
  Read 5858 numbers from solitaire.dat
2: BUCKETS solb TRUNCATE, DATA=solitaire
  Low computed = 0
  High computed = 52
  53 buckets created
  Finished buckets
3: IMAGE WIDTH=640, HEIGHT=480
  Image size is 640 x 480, set to 1
4: AXES_LABEL_FONT = "F8x13bold"
5: BAR solb XLABEL="Cards Up, mean " + mean(solitaire), YLABEL="Count"
  xll = 36, xur = 633
6: BMP "test.bmp"
  Wrote test.bmp at 640x480
```

Depending upon the output selected and system, there may also be a display of the generated image.

## 1.3 Plots

The following plots are implemented.

### 1.3.1 Bar Graphs

Counts in an array are displayed as solid bars off the horizontal axis in both 2 and 3 dimensions. There are also tools for computing histograms from data.

### **1.3.2 Box Graphs**

These are variations of the Tukey style plot that display 5 statistical values about data sets.

### **1.3.3 Line Graphs**

One or more sets of X-Y related data can be plotted on a single surface. Data can be sorted if it arrives unsorted. Line color and style can be varied. The data can be transformed for log graphs as well.

### **1.3.4 Circles**

Plot the data as a set of circles either at random locations or on indexed axes.

### **1.3.5 Maps**

The system can read some ESRI formatted shape files and has a limited set of cartography display tools and mapping transformations.

### **1.3.6 Pie Charts**

You can read a collection of data and plot them as a pie chart calling out a wedge to emphasize a particular value as well.

### **1.3.7 Scatter Plots**

One or more sets of X-Y related data can be plotted. Different symbols and different colors can distinguish data sets.

### **1.3.8 Surfaces**

Three dimensional datasets can be displayed on contour plots and as 3D surfaces. 3D viewing is restricted to certain angles.

### **1.3.9 General Graphics**

You can provide additional graphics with drawing of lines, areas, curves, text and other graphical constructs.

## **1.4 Tasks**

Various house keeping and other tasks can be performed.

### 1.4.1 Sorting

Data sets can be sorted into ascending/descending order with additional data following the sort. Thus it is possible to sort an X,Y dataset on either X or Y and have the other coordinate follow the sort.

### 1.4.2 Polynomial Curve Fitting

You can compute the linear least squares approximation to an X,Y dataset and plot this approximation as well. Polynomials can also be fit to data and the results plotted<sup>1</sup>.

### 1.4.3 Data Formats

Various data formats can be read including comma separated values (CSV) files, large sets of numbers in binary format, ESRI shape files and others.

### 1.4.4 Output Formats

You can generate your visualization into a Microsoft BMP format file or a Postscript image. Most of the images in this manual were written to Postscript and included in the  $\LaTeX$ document using the `graphicx` package.

---

<sup>1</sup>Not implemented yet

## Chapter 2

# Parameters and Expressions

Most commands accept lists of keywords and assigned expressions. These take the form of:

$$\langle \textit{keyword} \rangle = \langle \textit{expression} \rangle, \dots$$

repeated as needed. Each command accepts keywords and values specific to its function. Some accept groups of keywords such as those associated with axes display, grid markings, and so on. A keyword and expression can appear by itself as a way of modifying system global variables.

This chapter presents the syntax and semantics of these expressions and an expansion of some of the mathematical functionality it supports.

### 2.1 Primitives

The following data structures are supported:

*<integer>* 32 bit integers ranging from  $-2147483648 \rightarrow 2147483647$ . These integers can be included in the control file with the proviso that + and - when used for addition and subtraction must be separated from the integer value. Integers can also be in hexadecimal format by prefixing the digits with '0x' as in 0x1E0 (equivalent to 480).

*<float>* 64 bit double precision floating-point numbers. These numbers can be included in the control files with the usual E notation and the caveats about + and - for integers.

*<string>* Strings with up to 4096 characters in them. To include a double quote within a string you prefix it with a backslash. The back slash characters \n and \r generate codes 10 and 13 respectively. Strings can be included in input control files.

- <*list*> Lists of items which can be mixed integer and floating-point values. Lists cannot be read by the expression reader and cannot be included in the control file unless specified for the particular command.
- <*intvector*> A vector of integer values. These can only be created and cannot be included in the input file.
- <*floatvector*> A vector of double precision floating-point values. These can only be created and cannot be included in the input file.
- <*bytearray*> An *unsigned char* array with a width and height for storing the constructed image. These can only be created and cannot be included in the input file.
- <*intarray*> A 32 bit integer array with a width and height for contour plots and other purposes. These can only be created and cannot be included in the input file.
- <*floatarray*> A 64 bit double-precision floating-point array with a width and height. These can only be created and cannot be included in the input file.

## 2.2 Syntax

Addition, subtraction, multiplication and division are supported with left associativity and the usual operator precedence. Parentheses are used to temporarily override the default precedence. Parentheses enclose the arguments of a limited number of built-in functions. Constants include strings and numbers as above.

## 2.3 Arithmetic

Arithmetic operations between scalars, vectors and arrays are permitted. When an operation between an integer and floating-point value occurs, the value is always floating-point. The addition operator is overloaded for string concatenation.

### 2.3.1 Addition

Addition between numbers, strings, and vectors is supported. When adding a number to a string, the result is a new string with the number in ASCII. Strings up to 4096 characters in length are allowed. When adding vectors and lists the number of elements in each must be the same or an error is signalled. The possible combinations and their outcomes are shown in Table 2.1. Operations on vectors and arrays are always on an element by element basis.



$+$	<i>int</i>	<i>float</i>	<i>string</i>	<i>list</i>	<i>int vector</i>	<i>float vector</i>	<i>byte array</i>	<i>int array</i>	<i>float array</i>
<i>int</i>	int	float	string	list	int vector	float vector	byte array	int array	float array
<i>float</i>	float	float	string	list	float vector	float vector	float array	float array	float array
<i>string</i>	string	string	string	error	error	error	error	error	error
<i>list</i>	list	list	error	list	error	error	error	error	error
<i>int vector</i>	int vector	float vector	error	error	int vector	float vector	error	error	error
<i>float vector</i>	float vector	float vector	error	error	float vector	float vector	error	error	error
<i>byte array</i>	byte array	float array	error	error	error	error	byte array	int array	float array
<i>int array</i>	int array	float array	error	error	error	error	int array	int array	float array
<i>float array</i>	float array	float array	error	error	error	error	float array	float array	float array

Table 2.1: Addition type coercion table

### 2.3.2 Other Operations

Scalar subtraction, multiplication, and division between numbers and number structures is also implemented. In these cases strings are not allowed. Operations between vectors and arrays are on an element by element basis. Table 2.2 describes the permitted type coercions.

<i>-</i> <i>*/</i>	<i>int</i>	<i>float</i>	<i>list</i>	<i>int</i> <i>vector</i>	<i>float</i> <i>vector</i>	<i>byte</i> <i>array</i>	<i>int</i> <i>array</i>	<i>float</i> <i>array</i>
<i>int</i>	int	float	list	int vector	float vector	byte array	int array	float array
<i>float</i>	float	float	list	float vector	float vector	float array	float array	float array
<i>list</i>	list	list	list	error	error	error	error	error
<i>int</i> <i>vector</i>	int vector	float vector	error	int vector	float vector	error	error	error
<i>float</i> <i>vector</i>	float vector	float vector	error	float vector	float vector	error	error	error
<i>byte</i> <i>array</i>	byte array	float array	error	error	error	byte array	int array	float array
<i>int</i> <i>array</i>	int array	float array	error	error	error	int array	int array	float array
<i>float</i> <i>array</i>	float array	float array	error	error	error	float array	float array	float array

Table 2.2: Type coercions between other arithmetic operators.

## 2.4 Functions

Single argument functions are also implemented for simple statistical measures and common mathematical functions. Many of these work with lists, vectors and arrays producing a similar structure when appropriate.

### 2.4.1 ABS

For any scalar, list, vector or array, returns the absolute value of its elements or value in the same type. Byte arrays are returned as is since they have no signs.

### 2.4.2 AVERAGEDEVIATION

The **AVERAGEDEVIATION** function accepts lists and vectors with at least two elements. For the elements of these structures ( $M_0, \dots, M_n$ ), the average deviation is:

$$AVERAGEDEVIATION(M) = \frac{\sum_{i=0}^n |MEAN(M) - M_i|}{n}$$

### 2.4.3 INDEXED

The **INDEXED** function takes a vector as an argument and creates a string with all the values (up to 4k characters worth). Each entry is prefixed with its index, a colon and a blank. The vector element follows with floating-point

in %g format, integers %d format, and strings as they are. Each element is terminated by a newline character. The result can be used by the **DRAW TEXT** command to display the results of polynomial regression.

For example,

```
DRAW TEXT FONT=F6x10,X=0,Y=LASTY,MESSAGE="Degree: " + POLYNOMIAL_DEGREE
DRAW TEXT FONT=F6x10,X=0,Y=LASTY,MESSAGE="Coefficients:\n" +
INDEXED(POLYNOMIAL_COEFFICIENTS)
```

generates text showing the coefficients of a polynomial generated to fit a curve to a scatter plot.

#### 2.4.4 LOG

The **LOG** function computes the natural logarithm of its argument. This can be either an integer or floating-point value in which case a floating-point number is returned. If the argument is a list, the natural logarithm of each element is computed and a new list returned (in the same order as the argument). In the case of integer vectors, a new vector of floating-point values is returned. For floating-point vectors, a new vector is also returned.

#### 2.4.5 LOG10

The **LOG10** function computes the logarithm base 10 of its argument. This can be either an integer or floating-point value in which case a floating-point number is returned. If the argument is a list, the logarithm base 10 of each element is computed and a new list returned (in the same order as the argument). In the case of integer vectors, a new vector of floating-point values is returned. For floating-point vectors, a new vector is also returned.

#### 2.4.6 LOG2

The **LOG2** function computes the logarithm base 2 of its argument. This can be either an integer or floating-point value in which case a floating-point number is returned. If the argument is a list, the logarithm base 2 of each element is computed and a new list returned (in the same order as the argument). In the case of integer vectors, a new vector of floating-point values is returned. For floating-point vectors, a new vector is also returned.

Note that LOG2 is NOT implemented in windows - using it always results in 0.

#### 2.4.7 MEAN

The **MEAN** operation accepts lists and vectors with at least one element. For all elements of these structures ( $M_0, \dots M_n$ ) the mean is:

$$MEAN(M) = \frac{\sum_{i=0}^n M_i}{n + 1}$$

### 2.4.8 MEDIAN

Not implemented yet.

### 2.4.9 MAX

For any list, vector or array, this function returns its maximum value. For lists and floating-point structures, the result will be floating-point. For integer structures, the result will be an integer. Scalars and other types will cause an error to be signaled.

### 2.4.10 MIN

For any list, vector or array, this function returns its minimum value. For lists and floating-point vectors, this value will always be floating-point. For integer vectors and byte arrays, the result will be an integer. Other argument types such as scalars and strings will cause an error to be signaled.

### 2.4.11 ROUND

Floating-point values are converted to integers rounding up for positive values and rounding down for negative values. Integers are passed through unchanged. Rounding of a list of numbers is implemented - the integers remain the same and the floating-point values rounded accordingly. Integer vectors are passed through unchanged and floating-point vectors are converted to rounded integer vectors.

### 2.4.12 SIZE

Returns the number of elements in a list, the number of elements in a vector of any type, or 1 for scalar values. For arrays, the number of rows times the number of columns is returned.

### 2.4.13 SUM

Returns the sum of all elements in a list or vector. The value is returned as a floating point number whether or not the items summed are integers.

### 2.4.14 STANDARDDEVIATION

The **STANDARDDEVIATION** function computes the standard deviation of a list or vector. For all elements of these structures ( $M_0, \dots, M_n$ ) the standard deviation is:

$$STANDARDDEVIATION(M) = \sqrt{\frac{\sum_{i=0}^n (MEAN(M) - M_i)^2}{n}}$$

At least two elements are required. The function signals an error if presented with an array or scalar value.

### 2.4.15 VARIANCE

The **VARIANCE** function computes the variance of a list or vector. For all elements of these structures ( $M_0, \dots, M_n$ ) the variance is:

$$VARIANCE(M) = \frac{\sum_{i=0}^n (MEAN(M) - M_i)^2}{n}$$

## 2.5 Variables

There are many global variables defined as part of the set up routines. You can change these values at any time, but at your own risk. The following variables are not defined anywhere else.

**ARGV0 - ARGVn** These values are strings taken from the command line.

**ARGV0** will be the name used to execute this program, **ARGV1** is the command file being executed. Additional values can be included as well.

**NOISY** must have an integer value. If 0, no output should occur. If a value of 1, the control file will be echoed. Higher values will result in additional information.

**VERSION** Contains the program name and version as a string.



# Chapter 3

## Commands

**GRAPH** accepts single line commands that perform some action. We first present common attributes shared amongst the plotting commands. Individual commands are presented in alphabetical order by their keywords.

### 3.1 Axes

Commands that plot data value share code that draws the axes upon which the data is placed. A number of attribute-value items are shared amongst these commands.

The general syntax is:

$$\langle key \rangle = \langle expression \rangle$$

where the  $\langle key \rangle$  is a symbol and we give the value type required of the  $\langle expression \rangle$ .

**AXES\_FILL** =  $\langle integer \rangle$  Gives the color index of a 3 color group used to fill the back, left, and bottom axes area in 3D plots. The three colors should range from dark to light. The globals **GREYS**, **REDS**, **YELLOW**S, **GREENS**, **CYANS**, **BLUES**, and **MAGENTAS** have already been assigned indices and color values. If your plot request contains no such option, the default is acquired from the **SURFACE\_AXES\_FILL** or **BAR3D\_AXES\_FILL** depending upon your plot request.

**AXES\_FONT** =  $\langle string \rangle$  This option describes the font used to display numeric values along the bottom and left hand axes. If this option isn't used, the default font is found in the global variable **AXES\_FONT** which normally defaults to the "F6x10" font. Table 3.5 on page 68 gives the currently available fonts.

**AXES\_LABEL\_FONT** =  $\langle string \rangle$  This option gives the font to use for displaying labels on each axis. Typically, this will be a description of

the data on that axis. The vertical axis is displayed at a 90 degree angle and the display area is automatically adjusted to account for the extra labels. If you don't include this option, the value of the global **AXES\_LABEL\_FONT** will be used instead. This defaults to the "F8x13" font.

**XHIGH** = *<number>* If present, overrides the high/low computation for the data and sets the number to this value.

**XLABEL** = *<string>* If present, this option gives text to display below the numbers on the X axis. It will use the **AXES\_LABEL\_FONT** for its display as described above. If this option does not appear, no extra label will appear, though the numbers and graticule will.

**XLOW** = *<number>* If present, overrides the high/low computation for the data and sets the lowest X value to this number.

**YHIGH** = *<number>* If present, overrides the high/low computation for the data and sets the number to this value.

**YLABEL** = *<string>* If present, indicates the text to display for the vertical axis. This will appear with the **AXES\_LABEL\_FONT** font as described above and will be displayed at a 90 degree angle. If this option is not present, no label will appear.

**YLOW** = *<number>* If present, overrides the high/low computation for the data and sets the lowest X value to this number.

In addition to these keyword parameters, there are two global variables for less common modifications.

**AXES\_HORIZONTAL\_CHARS\_SEPARATION** is the number of pixels between the graph and the top of the numbers marking the horizontal axis. It defaults to 2.

**AXES\_VERTICAL\_CHARS\_SEPARATION** is the number of pixels between the graph and the last pixel of the text markings. It defaults to 2.

## 3.2 Background Grid

A plot's background includes the filled area and an graticule extending the side markings. The following options are supported:

**BACKGROUND** = *<expression>* This option changes the default color of the grid area background. The default value (usually 1 for white) resides in the **GRAPH\_BACKGROUND** global variable.



**GRID\_COLOR** = *<expression>* The expression must evaluate to an integer value between 0 and 255 inclusive. This specifies the grid marking colors to use. The default color will be found in the **GRID\_COLOR** global and is usually 8 (for grey).

**GRID\_STYLE** = *<expression>* The expression must be an integer value the bits of which indicate the style to use when drawing the grid lines. In a 32 bit integer, a bit of 1 indicates that a pixel in the selected color should be drawn, a 0 indicates nothing should be drawn. The default value (typically -1 for drawing solid lines) can be found in the **GRID\_STYLE** global variable.

Setting a grid style of 0 will cause the grid to not be drawn. However, two options provide finer control.

**NOHORIZONTAL\_GRID** If this option is present, the grid will have only vertical lines (these can also be disabled).

**NOVERTICAL\_GRID** If this option is present, the system won't draw the vertical lines.

### 3.3 BAR

Create a bar graph.

---

```

BAR <name>
      BAR_COLOR = <integer>, <list>, <vector>,
      BAR_SEPARATION = <integer>,
      GRAPH_BACKGROUND = <integer>
      ... Axes keywords ...
      ... Grid keywords ...

```

---

**Description:** Plots a bar graph of X and Y data. Typically this is the result of a histogram computation. The first argument is a symbol name such as that used in the BUCKETS command. This symbol is automatically conjoined with `_X` and `_Y` to look for the X and Y values.

**Keywords:** See the keywords associated with the AXES commands that can be part of this command. In addition, the following are supported:

**BAR\_COLOR** = {<integer>, <list>, <vector>} Sets the color to display the bars in. If you don't provide this option, the global variable **BAR\_COLOR** is used which defaults to saturated red. If the value is a list, it must have the same number of entries as the plot data. Each entry is the color number of the bar associated with it. If the plot values are vectors, then the color value must also be a vector (or integer for a single color).

**BAR\_SEPARATION** = <integer> This value indicates the number of pixels to leave between each bar in the display. If you don't provide the keyword, the default of 1 is retrieved from the **BAR\_SEPARATION** global.

**Variables Set:** As a result of building the axes, the following variables are set:

**MINX** The minimum X value used to generate the axes.

**MINY** The minimum Y value used to generate the axes.

**SCALEX** The scale computed for the X axis. An X pixel value will be:

$$X_{pixel} = XLL + \frac{x - MINX}{SCALEX}$$

**SCALEY** The scale computed for the Y axis. A Y pixel value will be:

$$Y_{pixel} = YLL + \frac{y - MINY}{SCALEY}$$

**XLL** The lower left corner X pixel coordinate of the graph area.

**YLL** The lower left corner Y pixel coordinate of the graph area.

**XUR** The upper right corner X pixel coordinate of the graph area.

**YUR** The upper right corner Y pixel coordinate of the graph area.

**Errors Detected:** In addition to errors detected by the axis display code, the following errors are detected.

**At line *nn*, missing data symbol to plot** The symbol name containing the data to plot can't be found by the parser.

**At line *nn*, value of *name* is bad** The value of one of the axes (*name*) is not an integer vector or list.

**At line *nn*, no image size declared yet** You must create an image with the IMAGE command first.

**At line *nn*, lower left  $x=minx$ , and right  $x=maxx$  are probably bogus**  
The plot area generated by the axes commands generate a plot area that's too large.

**Known Deficiencies:** Doesn't work with lists and floating-point vectors yet.

**See Also:** SYMBOL TABLE on page 110, BUCKETS on page 42.

**Example:** The following example reads 1000 Gaussian distributed integers and displays their histogram.

```

READCSV
DEVIATE
84
67
-3
.
.
.
70
83
47
71
EOF
BUCKETS gauss DATA=DEVIATE,BUCKET_SIZE=10
IMAGE WIDTH=320,HEIGHT=200
BAR gauss BAR_COLOR=BLUE
bmp "bar.bmp"

```

This results in the following BMP image.

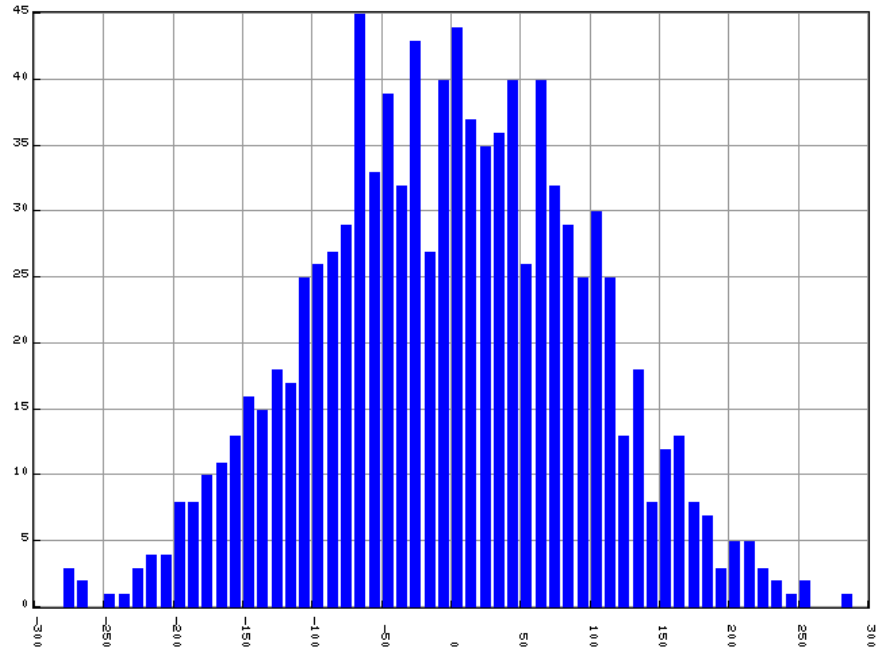


Figure 3.1: Bar graph.

The next example reads 256 histogram elements with the bottom 5% and top 5% displayed in red, the rest in blue. Many of the data lines have been removed. Note that the program generating the data must make the color assignment.

```
readcsv
pixel_x,pixel_y,bcolor
.
.
.
29,11685,2
30,13808,2
31,12516,2
32,13721,6
33,12732,6
.
.
.
215,15168,6
216,13565,6
```

```
217,14578,2
```

```
218,13740,2
```

```
.
```

```
.
```

```
.
```

```
EOF
```

```
IMAGE WIDTH=580,HEIGHT=480
```

```
BAR pixel BAR_COLOR=bcolor,XLABEL="Pixel Value",YLABEL="Count"
```

```
GIF "histo2.gif"
```

```
PS "histo2.ps"
```

```
DISPLAY file="histo2.gif"
```

The graph width was adjusted by hand until blank lines between bars disappeared.

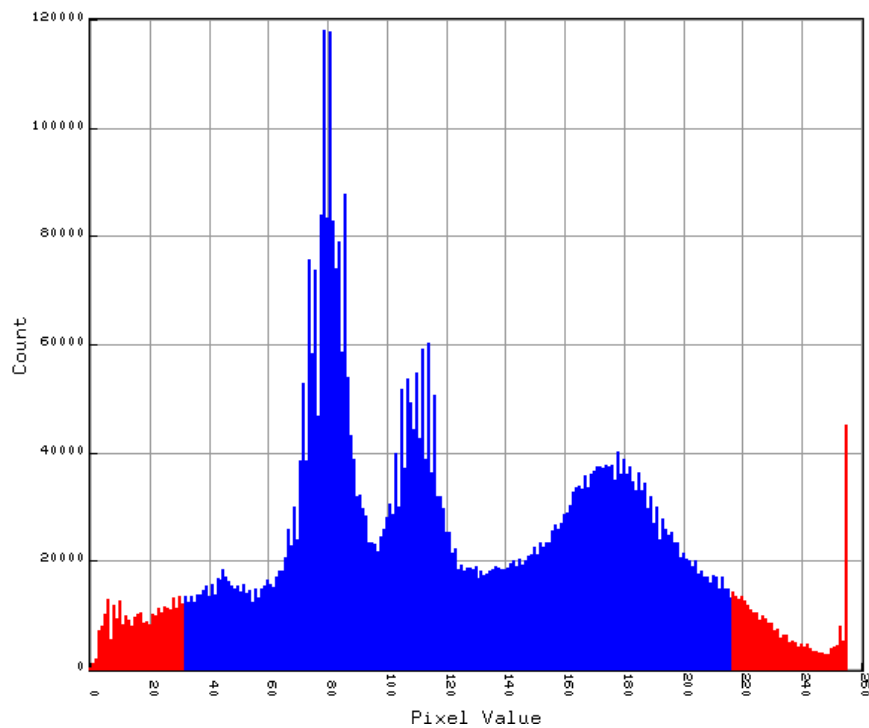


Figure 3.2: Multi-colored bar graph.

### 3.4 BAR3D

Create a 3D bar graph.

---

#### BAR3D

```

X = <list, vector>,
Y = <list, vector>,
AXES_FILL = <integer>,
COLOR = <integer>,
LABEL = <string>,
PERCENT = <number>,
ROTATEY = <number>,
XOFF = <number>,
YOFF = <number>,
ZOFF = <number>,
... Axes keywords ...,
... Grid keywords ...

```

---

**Description:** This display generates a 3 dimensional bar graph for one or more data sets. The bars can be different colors, the graph can be rotated and translated (within some fairly narrow bounds) and axes labeled and fiddled with in the usual means.

**Keywords:** Multiple data sets extend out along the Z axis with the vertical axis for showing value, the X axis for indices (typically the result of a histogram **BUCKETS** calculation). The following attribute-value pairs are recognized.

**AXES\_FILL** = *<integer>* This value specifies the first color index of 3 for filling the axes areas. The default value is found in the global variable **BAR3D\_AXES\_FILL**. The first color should be dark, the second lighter and the third lightest giving the appearance of the graph being lit from the top.

**COLOR** = *<integer>* This value specifies the color for the next set of bars. The color index refers to a dark value and the next two color indices should be lighter versions. The default color index is found in the **BAR3D\_COLOR** and defaults to 12, a set of 3 red colors. Globals with values include **GREYS**, **REDS**, **YELLOW**S, **GREENS**, **CYANS**, **BLUES**, and **MAGENTAS**.

This option can occur multiple times to provide different colors for different data sets.

**LABEL** = *<string>* This option provides a label for each bar to appear. This label will then appear in the legend if one is present.

**PERCENT** = *<number>* This value specifies how much of the space available for a bar is actually used. 100.0 results in run together bars, 0.0 makes them disappear. The default value is 80.0 and is found in the global **BAR3D\_PERCENT\_WIDTH**.

**ROTATEY** = *<number>* Sets the rotation around the Y (vertical) axis in degrees. The default value (-45.0) is found in the global **BAR3D\_ROTATEY** variable. Values should not exceed 0.0 and should not be less than -90.0 or the ordered hidden surface removal won't work.

**X** = *<list,vector>* This keyword and value must be present at least once. It specifies the X values for the corresponding Y values. The X and Y structures must be of the same type and size. When both an X and Y value are found in the list, a set of bars is created. The X and Y values are then removed and no plotting occurs until the next set is found.

**XOFF** = *<number>* Sets the X axis offset. The default value found in **BAR3D\_XOFF** is 0.0. Increasingly large negative values will shift the display area to the left. Larger positive values will go to the right.

**Y** = *<list,vector>* This keyword and value must be present at least once. It specifies the Y values for the corresponding X values. The X and Y structures must be of the same type and size. When both an X and Y value are found in the list, a set of bars is created. The X and Y values are then removed and no plotting occurs until the next set is found.

**YOFF** = *<number>* Sets the Y axis offset. The default value found in **BAR3D\_YOFF** is -.25. Increasingly large negative values will shift the display area down. For Larger positive values it will go up.

**ZOFF** = *<number>* Sets the Z axis offset. The default value found in **BAR3D\_ZOFF** is -3.0. Increasingly large negative values will shift the display area farther away mitigating some effects of the perspective transformation but will result in smaller images. The default viewing display area lies in the  $-2 \rightarrow -4$  Z area.

**Variables Set:** As a result of display the following variables are set:

**XLL** The lower left corner X pixel coordinate of the graph area.

**YLL** The lower left corner Y pixel coordinate of the graph area.

**XUR** The upper right corner X pixel coordinate of the graph area.

**YUR** The upper right corner Y pixel coordinate of the graph area.

**Errors Detected:** In addition to bad type errors and out of storage errors, the following are signaled.

At line *nn*, **COLOR value is not an integer** Color values must be integers.

**At line  $nn$ , X size  $xx$  and Y size  $yy$  are different** Vector and list sizes must be identical.

**At line  $nn$ , X and Y aren't same type or not supported types** You can't mix types. Scalars and arrays aren't supported in any case.

**Known Deficiencies:** Doesn't work with lists and floating-point vectors yet.

**See Also:** **BAR** on page 26, and **LEGEND** on page 73.

**Example:** The following example reads some data, computes a histogram, and displays a 3D bar graph.

```
image sol width=640, height=480
READONED solitaire "../solitaire.dat"
BUCKETS solb TRUNCATE, DATA=solitaire
bar3d X=solb_x,Y=solb_y,rotatey=-25, xoff=-.5,zoff=-2.0,PERCENT=50
ps "bar3d.ps"
```

This results in the Postscript image of Figure 3.3.

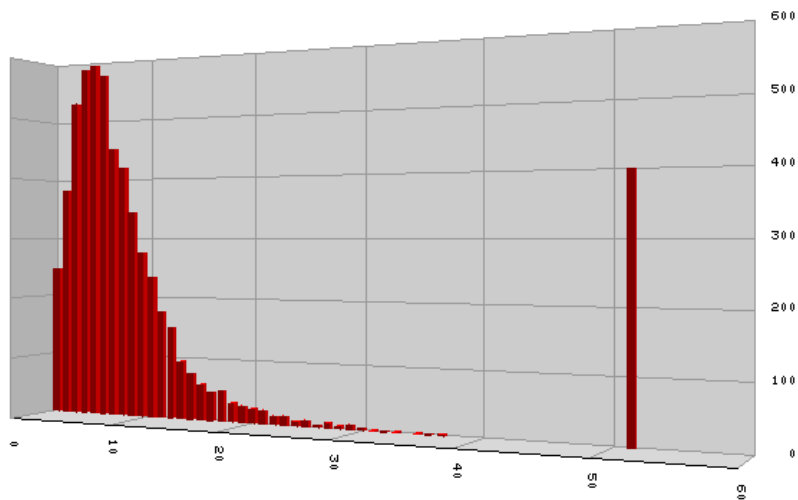


Figure 3.3: 3D Bar graph.

This next example reads a CSV file with some statistical data: each record is named with an alphabetic character as follows:



```

motor, April, June, January
"1/4A", 0, 3, 1
"1/2A", 0, 2, 2
"A", 9, 7, 3
"B", 6, 11, 4
"C", 25, 8, 6
"D", 10, 15, 7
"E", 3, 6, 2
"F", 11, 5, 1
"G", 8, 6, 1
"H", 1, 6, 3
"I", 4, 10, 5
"J", 2, 4, 4
"K", 1, 2, 1
"L", 0, 0, 2
"M", 0, 0, 1
"N", 0, 0, 1
"O", 0, 0, 0
"P", 0, 0, 0

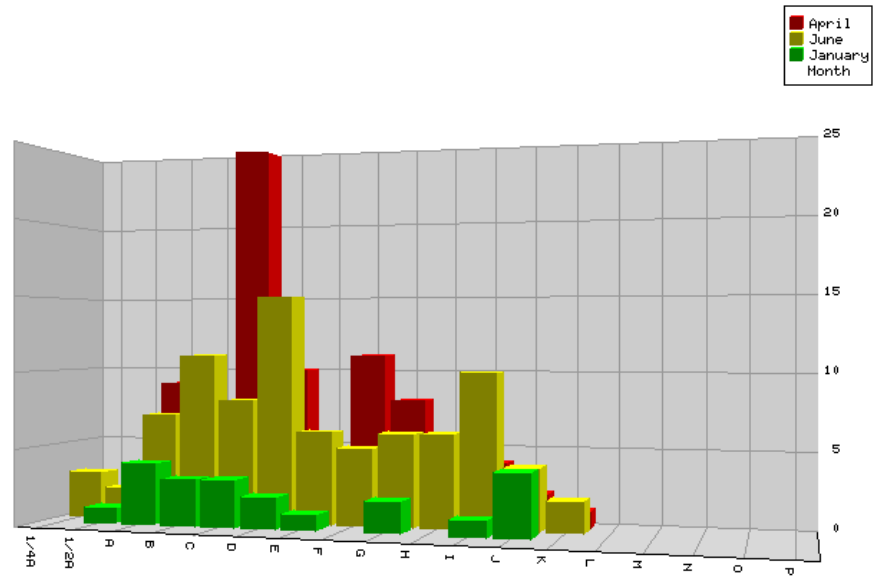
```

The script reads this data, sets the font and then draws all 3 bar sets. This line is too long and is split over several lines. A legend with the labels and colors is drawn.

```

// Demonstrate string labels.
IMAGE WIDTH=640,HEIGHT=480
READCSV "motors.csv"
AXES_LABEL_FONT = F12x24
BAR3D XOFF=-.475,ZOFF=-2,COLOR=REDS,LABEL="April",X=motor,Y=April,COLOR=YELLOWS,
      LABEL="June",X=motor,Y=June,COLOR=GREENS,LABEL="January",X=motor,Y=January,
      ROTATEY=-15,XLABEL="UROC Motor Totals"
LEGEND MESSAGE="Month"
PS "motors.ps"

```



### UROC Motor Totals

Figure 3.4: Multiple bars on a single plot.

### 3.5 BOX

Box (Tukey) plot.

---

#### BOX

**COLOR** = *<integer>*,  
**X** = *<list, vector>*,  
**Y** = *<list, vector>*,  
**LLSQ** = *<integer>*,  
**POLYNOMIAL\_COLOR** = *<integer>*,  
**POLYNOMIAL\_DEGREE** = *<integer>*,  
**SIZE** = *<number>*,  
**WHISKER\_STYLE** = *<integer>*,  
**RANGE** = *<integer>*,  
**PERCENTILE** = *<number>*,  
*... Axes keywords ...*  
*... Grid keywords ...*

---

**Description:** Generate a box plot in the current display area. Also called a Tukey plot and attempts to present 5 values in a limited space.

**Keywords:** See the keywords associated with the axes and grid described earlier. The **X** and **Y** keywords must be present, the rest are optional. The following are supported.

**COLOR** = *<integer>* Indicates the color index to use for the next line display. If no color is given, the value in the global variable **BOX\_COLOR** (typically **BLACK**) is used.

**RANGE** = *<integer>* This option specifies how the whisker range is to be computed. The 4 possible options (and their values) are specified in Table 3.1. The default value is taken from the **BOX\_RANGE** global which is normally 2 for the IQR range.

Name	Value	Description
<b>MINMAX</b>	0	The top whisker marks the largest bucket value and the bottom the smallest. No outlier circles will appear.
<b>STANDARD_DEVIATION</b>	1	The top and bottom whiskers lie one standard deviation on either side of the bucket mean.
<b>IQR</b>	2	The top and bottom whiskers lie 1.5 times the interquartile range above and below the first and third quartiles.
<b>PERCENTILE</b>	3	The top and bottom whiskers lie at $100 - \textit{Percentile}$ and $\textit{Percentile}$ percent of each bucket's list. The value must range between 0 and 25.0.

Table 3.1: Box plot whisker range values

**LLSQ** =  $\langle integer \rangle$  Indicates that if X and Y have been selected, that a linear least squares computation should be performed on the data and its line drawn with the color given. If this occurs before the **SYMBOL** keyword, the symbols will be drawn on top the line if after, above the line. A legend entry is made with the line color and the **M** and **B** values as in:

$$y = Mx + B$$

Every time LLSQ is computed, the two parameters are stored in the global variables **M** and **B**. If multiple lines were fit in one graph, only the last is available.

**POLYNOMIALCOLOR** =  $\langle integer \rangle$  If you request a polynomial curve fit to the data by using the **POLYNOMIAL\_DEGREE** option, this color will be used to draw the line. If you don't specify a color, then the value of the global **POLYNOMIAL\_COLOR** will be used.

**POLYNOMIAL\_DEGREE** =  $\langle integer \rangle$  This option will fit a polynomial of the degree selected to the current data set. The color is selected by the **POLYNOMIAL\_COLOR** value as described above. This option does not add an entry for the legend as it's much too large. Setting **NOISY** greater than 1 will dump the computed coefficients to standard output.

The values of the polynomial coefficients are stored in the **POLYNOMIAL\_COEFFICIENTS** global variable and the degree selected is also stored in **POLYNOMIAL\_DEGREE**. If there are multiple polynomials fit in one graph, only the values for the last one are available.

**SIZE** =  $\langle number \rangle$  The size of each bucket. X values lie within  $\frac{1}{2}$  this value on either side of the bucket center point. If you don't include

the option, the default size is taken from the global **BOX\_SIZE**, normally 1.0. This can have harrowing results if your X range is large.

**WHISKER\_STYLE** = *<integer>* This option gives a 32 bit bit pattern for displaying the line connecting the quartile box to the whisker. The default is a dashed line (code 0xF0F0F0F0) taken from the global variable **BOX\_WHISKER\_STYLE**.

**X** = *<list, vector>* This required option must have a list or vector as its value. The Y values are put into sorted buckets based on their associated X value and the bucket size.

**Y** = *<list, vector>* This required option must have a list or vector as its value. The Y values are put into sorted buckets based on their associated X value and the bucket size.

**Variables Set:** As a result of building the axes, the following variables are set:

**MINX** The minimum X value used to generate the axes.

**MINY** The minimum Y value used to generate the axes.

**SCALEX** The scale computed for the X axis. An X pixel value will be:

$$X_{pixel} = XLL + \frac{x - MINX}{SCALEX}$$

**SCALEY** The scale computed for the Y axis. A Y pixel value will be:

$$Y_{pixel} = YLL + \frac{y - MINY}{SCALEY}$$

**XLL** The lower left corner X pixel coordinate of the graph area.

**YLL** The lower left corner Y pixel coordinate of the graph area.

**XUR** The upper right corner X pixel coordinate of the graph area.

**YUR** The upper right corner Y pixel coordinate of the graph area.

**Errors Detected:** In addition to errors detected by the axis display code, the following errors are detected.

**At line *nn*, missing X/Y values** Both X and Y values must be listed.

**At line *nn*, X and Y are different types** Their types must be the same.

**At line *nn*, WHISKER\_STYLE is not an integer** This must be a dashed line value style.

**At line *nn*, RANGE is not an integer** The range display type must be an integer between 0 and 3.

**At line *nn*, RANGE type *xx* is not known** The value must be between 0 and 3.

**At line *nn*, with RANGE=PERCENTILE, need a PERCENTILE value**  
You need the PERCENTILE value for PERCENTILE whiskers.

**At line *nn*, CIRCLE size is not an integer** The CIRCLE option needs an integer value.

**At line *nn*, CIRCLE size *cc* is illegal** The circle radius must be between 1 and some positive value.

**At line *nn*, size of X list and Y lists different** The lists for X and Y must be the same length.

**At line *nn*, X/Y value is not a number** Something in the list is not a number (string, symbol, etc??).

**At line *nn*, size of X and Y vectors different** The vectors must be the same size when used.

#### Known Deficiencies:

**See Also:** **COLOR** command on page 50 and the default color values listed there, the **BAR** command on page 26 and **BAR3D** command on page 30.

**Example:** The following example reads two files of 1000 pseudo-random numbers. The `experimentx.bin` file contains the uniform distribution X values as a binary integer vector. The `experimenty.bin` contains Gaussian distribution numbers times 100.0. Four different ranges are demonstrated and the result collected into a 2x2 display.

Note that two of the lines are too long to be displayed and are broken.

```
// Box plot.
READBIN xvals = "../regression/experimentx.bin"
READBIN yvals = "../regression/experimenty.bin"

IMAGE mnmx WIDTH=320,HEIGHT=200
BOX X=xvals, Y=yvals, SIZE=10, RANGE=MINMAX, XLABEL="MINMAX range", COLOR=RED

IMAGE stdev WIDTH=320,HEIGHT=200
BOX X=xvals, Y=yvals, SIZE=5, RANGE=STANDARD_DEVIATION, XLABEL="Standard Deviation Range",
CIRCLE=2

IMAGE iqr WIDTH=320, HEIGHT=200
BOX X=xvals, Y=yvals, SIZE=10, XLABEL="1.5 IQR range", COLOR=BLUE, WHISKER_STYLE=-1

IMAGE pct WIDTH=320, HEIGHT=200
BOX X=xvals, Y=yvals, SIZE=20, RANGE=PERCENTILE, PERCENTILE=20, XLABEL="20% range",
CIRCLE=1

IMAGE WIDTH=650,HEIGHT=410
```

```

PLACE mnmx, X=0, Y=210
PLACE stdev, X=330, Y=210
PLACE iqr, X=0, Y=0
PLACE pct, X=330, Y=0
BMP "box.bmp"
PS "box.ps"
DISPLAY file="box.bmp"
    
```

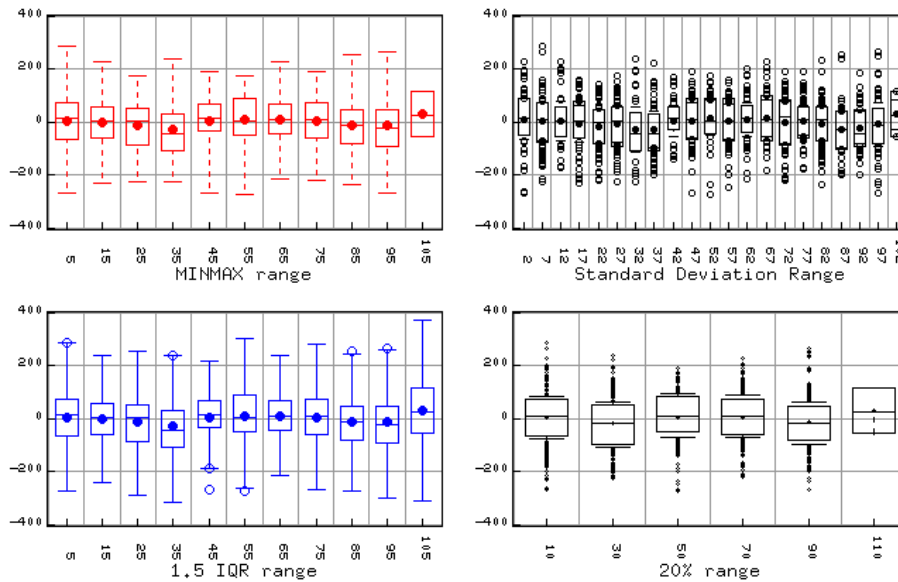


Figure 3.5: Multiple box plots.

A second example shows how LLSQ can be used to fit a straight line to some linear data and show its equation in the legend. There are 100 data points (not all are shown) randomly perturbed about a straight line in both X and Y. The graph shows these in bins 20 wide with a blue straight line over the data. The linear least squares equation values are shown in the legend (see Figure 3.6).

```

READCSV
XValue,YValue
0.489479,0
0.529269,1.42995
...
97.2071,48.9507
97.5944,106.579
99.2232,75.0649
EOF
    
```

```

IMAGE WIDTH=640,HEIGHT=480
BOX
RANGE=STANDARD_DEVIATION,COLOR=RED,X=XValue,Y=YValue,LLSQ=BLUE,SIZE=20,
  XLABEL="X",YLABEL="Y"
LEGEND X=50,MESSAGE="Some Data"
GIF "boxllsq.gif"
DISPLAY FILE="boxllsq.gif"

```

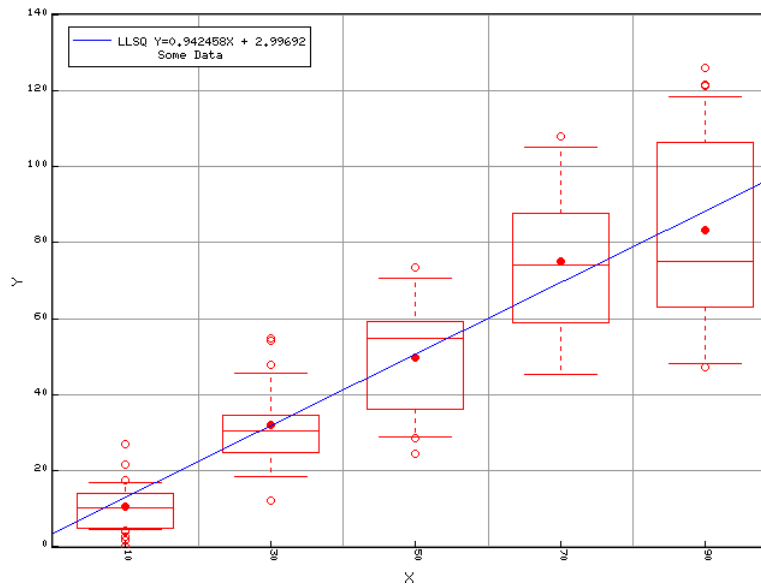


Figure 3.6: Box plot with LLSQ line fit.



## 3.6 BMP

Create a Microsoft BMP format image.

---

### BMP <string>

---

**Description:** This command takes the current state of the image (if there is one) and generates a Microsoft BMP format file using the string for a name. This will be an 8 bit color map image of the size set by the image command.

**Keywords:** None.

**Errors Detected:** The following errors are detected:

**At line *nn*, no image to write** No image is available. You must use an **IMAGE** command before this.

**At line *nn*, missing file name string** The command wasn't followed by a string file name.

**BMP\_WRITE: Couldn't open *xxx* for output because ...** The writing routine couldn't open the output file. The system reason is given. Typically the directory you want to write to is protected or doesn't exist.

**See Also:** **DISPLAY** on page 61, **GIF** on page 71, **IMAGE** on page 72, **PS** on page 87.

### 3.7 BUCKETS

Create a histogram of data.

---

```

BUCKETS <symbol>
  DATA = <list>,
  LOW = <integer>,
  HIGH = <integer>,
  BUCKET_SIZE = <number>,
  TRUNCATE,
  ROUND

```

---

**Description:** This simple histogram computation counts the number of entries that fall within buckets.

**Errors Detected:** The following errors are detected:

- At line *nn*, missing symbol for BUCKETS** The symbol with the data cannot be found by the parser.
- At line *nn*, symbol *sss* already defined** The X and Y associated symbols to be created already have values. Rather than destroy things, you should rename the input data.
- At line *nn*, no DATA field for BUCKETS** The DATA keyword is required. It should evaluate to a list.
- At line *nn*, DATA field is empty or wrong** The value put forward for DATA is not a list of numbers.
- At line *nn*, data file contains non-numbers** A list element is not a number.
- At line *nn*, value of LOW is not an integer** If the keyword LOW is present, its value must be an integer.
- At line *nn*, data is only value *ddd*** The data or the range had only one value *ddd* associated with it.
- At line *nn*, BUCKET\_SIZE=*bbb* > range=*rrr*** The bucket size computed is greater than the range - everything fits into one bucket.

**Keywords:** The first symbol is mandatory. This must be followed by the attribute-value keyword list separated by commas.

**BUCKET\_SIZE**=<number> Indicates the size of buckets to generate between the HIGH and LOW values (whether supplied or computed). If you don't supply this value, the value of the global **BUCKET\_SIZE** is used instead. This is normally 1.

**DATA** =<*list*> This keyword must be present. Its value must be a list of numbers. The values are counted in the appropriate buckets.

**HIGH**=<*integer*> If you provide this value, all values in DATA greater than this will be ignored. If you don't provide the value, the system computes the highest value provided by DATA and uses that.

**LOW**=<*integer*> If you provide this value, it marks the smallest value that will be counted. Values smaller than this are ignored. If you don't provide this value, it is automatically computed as the smallest value supplied by DATA.

**ROUND** If this keyword is present (by itself) values are rounded up when the bucket is computed.

**TRUNCATE** If this keyword is present (by itself) values are truncated to integer when the bucket number is computed.

**Known Deficiencies:**

**See Also:** BAR on page 26, BUCKETS on page 42.

**Example:** Reads a one-dimensional set of data and plots a bar graph of its histogram.

```
READONED solitaire="solitaire.dat"
BUCKETS solb truncate, data=solitaire
IMAGE width=640, height=480
AXES_LABEL_FONT = "F8x13bold"
BAR solb XLABEL="Cards Up, mean " + mean(solitaire), YLABEL="Count"
BMP "test.bmp"
```

This results in the following graph:

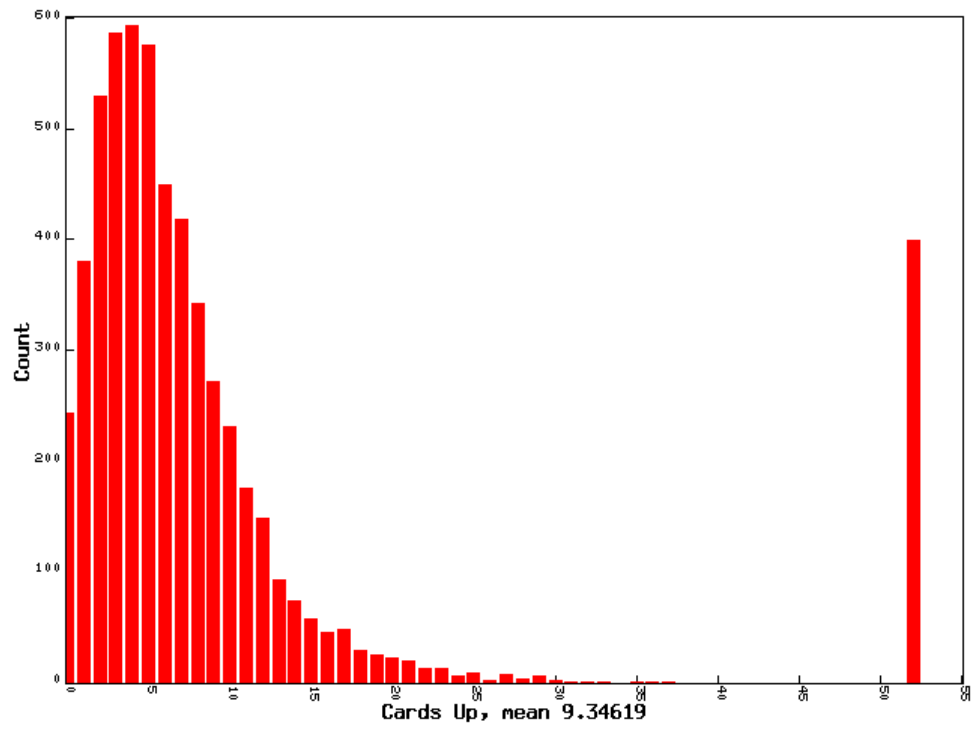


Figure 3.7: Results of bucketizing data

## 3.8 CIRCLES

Display X, Y, Z data as filled circles. X and Y values are indexed through lists and have names.

---

### CIRCLES

```

COLOR = <integer>,
COLORS = <list>,
COUNT = <integer>,
EQUALCOLOR = <integer>,
SIZES = <list>,
XINDEX = <list>,
XNAMES = <list>,
XVALUES = <list>,
YINDEX = <list>,
YNAMES = <list>,
YVALUES = <list>,
... Axes keywords ...
... Grid keywords ...

```

---

**Description** This command plots circles sized in relation to the Z value. Each X and Y value is indexed into a set of names in XNAMES and YNAMES whose position in the list becomes the actual X and Y value.

**Keywords:** See the keywords associated with the axes and grid options that can be part of this command. In addition, the following are supported:

**COLOR** = *<integer>* If this option is present, all circles are drawn with the indicated color value. If not present (and the **COLORS** keyword is also not present, then the circles are drawn in black (color 0). The **COLOR** option overrides the **COLORS** option if both are present.

**COLORS** = *<list>* If the **COLOR** option is not present, this specifies a list of colors to cycle through for each plotted circle. No attachment to names or indices are attempted and when the list runs out, assignment starts over with the first. This option is overridden by the **COLOR** option if both are present.

**COUNT** = *<integer>* If this option is present, this number of circle sizes will be added to the **LEGEND** display. The circles are displayed in black, and range from the smallest to largest. If no **COUNT** is present, then nothing is added to the legend.

**EQUALCOLOR** = *<integer>* If this option is present, circles with corresponding **XVALUES** and **YVALUES** that are equal will be displayed in this color. If the color is negative, an unfilled circle of the positive value will be drawn.

**SIZES** =  $\langle list \rangle$  This is a required list of circle radii in pixels. These are rounded up before doing the area fill. The minimum radius is 1 pixel and smaller values will be rounded up to this. The list length must be the same as that for the **XVALUES** and **YVALUES** lists.

**XINDEX** =  $\langle list \rangle$  This is a list of indices into the **XNAMES** list. The first list element corresponds to the zeroth element of **XNAMES**. For all  $j$  elements of **XVALUES**, there's an  $i$ th element of **XINDEX** with this value:

$$\forall j \exists i_j, XINDEX_{i_j} = XVALUES_j$$

The corresponding name (and horizontal position on the graph) is then:

$$name_j = XNAMES_{i_j}$$

**XNAMES** =  $\langle list \rangle$  This is a list of strings whose position in the list correspond to the values in the **XINDEX** list. Both these lists must have the same length. These names are shown on the horizontal axis with sufficient space left between entries to fill the available space. The labels are rotated right to 270 degrees.

**XVALUES** =  $\langle list \rangle$  This list gives the horizontal axis value for the displayed circles. These numeric values should have corresponding entries in **XINDEX** (see above) that are then converted into integer horizontal axis positions.

**YINDEX** =  $\langle list \rangle$  This required list is similar in function to the **XINDEX** list except that values are for the vertical axis.

**YNAMES** =  $\langle list \rangle$  This required list is similar in function to the **YNAMES** list except that values are labels for the vertical axis.

**YVALUES** =  $\langle list \rangle$  These are the vertical axis values. This list has the same properties and must have the same size as the **XVALUES** item.

**Variables Set:** As a result of building the axes, the following variables are set:

**MINX** The minimum X value used to generate the axes.

**MINY** The minimum Y value used to generate the axes.

**SCALEX** The scale computed for the X axis. An X pixel value will be:

$$X_{pixel} = XLL + \frac{x - MINX}{SCALEX}$$

**SCALEY** The scale computed for the Y axis. A Y pixel value will be:

$$Y_{pixel} = YLL + \frac{y - MINY}{SCALEY}$$

**XLL** The lower left corner X pixel coordinate of the graph area.

**YLL** The lower left corner Y pixel coordinate of the graph area.

**XUR** The upper right corner X pixel coordinate of the graph area.

**YUR** The upper right corner Y pixel coordinate of the graph area.

#### Errors Detected:

#### Known Deficiencies

**See Also:** **COLOR** command on page 50 and the default color values listed there. The **READCSV** command on page 90 is the preferred way of generating lists of values from data.

**Example:** The following example reads some X,Y data and counts and draws circles but of 2 times the natural log of the count.

The first data set creates the index/name lists for **XINDEX**, **XNAMES**, and **YINDEX**, **YNAMES**.

```
READCSV
Index,Name
0,"Human"
100,"Dog"
101,"Cat"
102,"Horse"
103,"Duck"
104,"Rat"
105,"Bug"
106,"Bird"
200,"Car"
201,"Truck"
202,"Van"
203,"Bicycle"
204,"Forklift"
205,"Motorcycle"
206,"Airplane"
207,"Train"
208,"Bus"
999,"Unknown"
EOF
```

The next section creates the data to plot as three lists for **XVALUES**, **YVALUES**, and **SIZES**.

```
READCSV
Video,Reality,Count
0,0,100
```

```

0,100,20
100,0,15
100,100,25
200,200,5
100,200,5
207,207,12
208,0,1
0,208,1
200,207,25
207,103,12
103,100,188
103,101,40
101,200,29
EOF

```

Next, we create a list of color values for display. These are simply numbers selected from the table in section 4.1 on page 115.

```

READCSV
cv
2
3
4
5
6
7
EOF

```

Finally, we create the graph using this data. Note that the command is too line to fit on one line and is split for demonstration.

```

image width=640,Height=480
CIRCLES XLABEL="Match",YLABEL="Reality",XINDEX=Index,YINDEX=Index,XNAMES=Name,YNAMES=N
XVALUES=Video,YVALUES=Reality,SIZE=2 * log(Count),COLORS=cv,COUNT=5
LEGEND
gif "circles.gif"
display file="circles.gif"

```

This results in the graph in Figure 3.8.



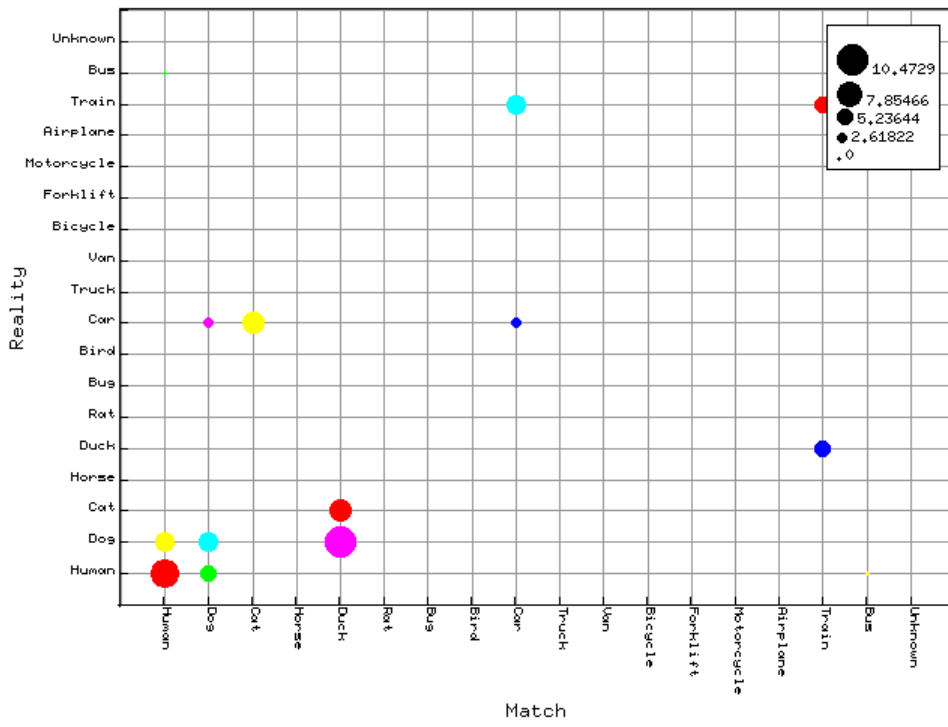


Figure 3.8: CIRCLES command with sample output

### 3.9 COLOR

Create or modify a single color.

---

```
COLOR <expression>,
      RED = <float>,
      GREEN = <float>,
      BLUE = <float>
```

---

**Description:** The default colormap is defined at startup. You can modify any of its 256 different colors.

Table 3.2 defines the default colormap - the first 105 values are predefined, the remaining 151 are available for use. The global variables can be used in place of numeric color values.

Global	Index	Description
<b>BLACK</b>	0	Saturated black. (red=0, green=0, blue=0).
<b>WHITE</b>	1	Saturated white. (red=1, green=1, blue=1).
<b>RED</b>	2	Saturated red. (red=1, green=0, blue=0).
<b>YELLOW</b>	3	Saturated yellow. (red=1, green=1, blue=0).
<b>GREEN</b>	4	Saturated green. (red=0, green=1, blue=0).
<b>CYAN</b>	5	Saturated cyan. (red=0, green=1, blue=1).
<b>BLUE</b>	6	Saturated blue. (red=0, green=0, blue=1).
<b>MAGENTA</b>	7	Saturated magenta. (red=1, green=0, blue=1).
<b>GREY60</b>	8	60% grey (red=0.6, green=0.6, blue=0.6).
<b>GREYS</b>	9	3 grey values (9, 10, 11) used for 3D displays.
<b>REDS</b>	12	3 saturated red shades (12, 13, 14) from 0.5 → 1 used for 3D displays.
<b>YELLOWS</b>	15	3 yellow shades (15, 16, 17) used for 3D displays.
<b>GREENS</b>	18	3 green shades (18, 19, 20) used for 3D displays.
<b>CYANS</b>	21	3 cyan shades (21, 22, 23) used for 3D displays.
<b>BLUES</b>	24	3 blue shades (24, 25, 26) used for 3D displays.
<b>MAGENTAS</b>	27	3 magenta shades (27, 28, 29) used for 3D displays.
<b>ELEVATION</b>	30	The next 96 colors are in 32 element blocks ranging from green → yellow → red → magenta.
<b>FIRSTFREE</b>	126	Colors from 126 → 255 are not assigned and default to saturated black (red=0, green=0, blue=0).

Table 3.2: The default color map.

**Keywords:** The following keywords are recognized. The default color values are 0. The values specified must range from 0.0 → 1.0 or an error is

signaled. At least one parameter must be present, those that aren't default to 0.

**RED** = *<float>* Specifies the red value for this color.

**GREEN** = *<float>* Specifies the green value for this color.

**BLUE** = *<float>* Specifies the blue value for this color.

**Errors Detected:** The following errors are detected.

**At line *nn*, color number *mm* is illegal** The color number shown is illegal. Only colors 0 → 255 can be set.

**At line *nn*, keyword *KEYWORD* must be numeric** One of the RED/GREEN, or BLUE keywords has a non-numeric value.

**At line *nn*, r=*rrr*,g=*ggg*,b=*bbb* one of which is illegal** One of the three listed values is outside of the legal range of 0 → 1 inclusive.

**See Also:** **COLORMAP** on page 52, **IMAGE** on page 72.

**Example:** Examples are scattered throughout the text.

### 3.10 COLORMAP

Create a set of colors.

---

```
COLORMAP
  SHADE = <integer>,
  COUNT = <integer>,
  START = <integer>
```

---

**Description:** The 3D surface routines need a colormap for generating highlights based on surface normals. The COLORMAP command creates single hue colormaps for such purposes.

**Keywords:** You specify an existing color as a basis (the shade) and the command creates colors darker than that and lighter than that.

**SHADE = <integer>** Specifies an existing color to serve as the medium display value. Typically this would be a saturated color such as RED. This option must be present or an error is signaled.

**COUNT = <integer>** Specifies how many colors to generate. The larger the number, the smoother the color transitions. If the option is not specified, the count is taken from the global variable **COLORMAP\_COUNT** which defaults to 64.

**START = <integer>** Specifies which color index to put the first new shade into (the darkest color sits here). Subsequent shades are written higher. This option must be present.

**Errors Detected:** The following errors are detected.

**At line *nn*, SHADE value must be an integer** The shade base (color index) must be an integer.

**At line *nn*, missing SHADE color index** The SHADE value must be present, there is no default global value.

**At line *nn*, COUNT value must be an integer** The COUNT value must be an integer. It need not be present though.

**At line *nn*, START value must be an integer** The starting color index must be an integer.

**At line *nn*, missing START value** The starting value must be present.

**See Also:** **COLOR** on page 50 and **SURFACE** on page 104.

**Example:** Examples are presented in the **SURFACE** section on page 104.

## 3.11 CONTOUR

Create a contour plot of X,Y,Z data.

---

### CONTOUR

```

ELEVATION = <array>,
BASIS = <number>,
ALTERNATE = <integer>,
COLOR1 = <integer>,
COLOR2 = <integer>,
XBASE = <number>,
XINC = <number>,
YBASE = <number>,
YINC = <number>,
SHADE = <integer>,
DIRECTION = <integer>,
COLORS = <integer>,
ELEVATION_COLOR = <integer>,
ZLOW = <number>,
ZHIGH = <number>,
... Axes keywords ...
... Grid keywords ...

```

---

**Description:** Display a contour plot (with lines) of an array of data. Use bi-linear interpolation if the data does not cover the plot area.

The command also allows for elevation coloring and slope shading backgrounds. These options are mutually exclusive - one, the other, neither, but not both.

Figure 3.9 shows how the elevation of the red dot is computed when we only have data for the 4 green dots. We first compute  $ea$  and  $eb$  by the triangles between the two top and two bottom points respectively. We then interpolate between these two values in the Y axis to get  $e$ .

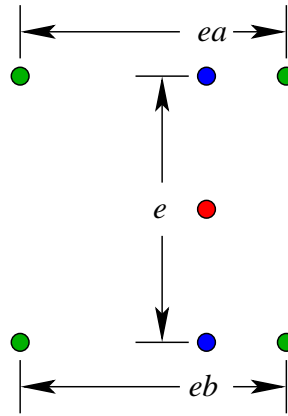


Figure 3.9: Bilinear Interpolation

**Keywords:** See the keywords associated with the axes and grid commands that can be part of this command. In addition, the following are supported:

**ALTERNATE** =  $\langle integer \rangle$  If present with a positive, non-zero value, then every alternate number of contour lines (starting at 0) is displayed with the second color (**COLOR2** keyword or **CONTOUR\_COLOR2** global).

**BASIS** =  $\langle number \rangle$  The basis number indicates where lines should be drawn. For a pixel in the center, if any of the three comparisons in Figure 3.10 shows a value less than and a value greater than a value modulo the basis, a dot is drawn with the appropriate color.

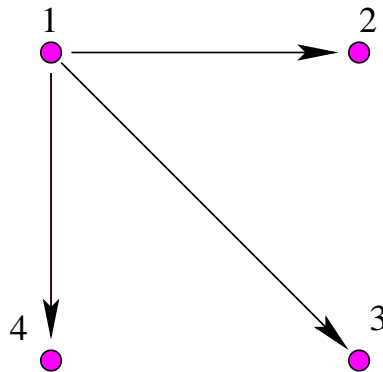


Figure 3.10: Basis computation.

**COLOR1** = *<integer>* Indicates the color index to use for for displaying lines indicating where the surface crosses a BASIS value. If no color is specified, then the global variable variable **CONTOUR\_COLOR1** (typically **BLACK**) is used.

**COLOR2** = *<integer>* This option gives the color index for alternate lines. For example, if your contour lines are every 10 vertical units you might wish to mark the 100 unit lines with a separate color as indicated. The default color is taken from **CONTOUR\_COLOR2** global and is by default **RED**.

**COLORS** = *<integer>* For the **SHADE** and **ELEVATION\_COLOR** options, this provides the number of colors used for shading or color display. If you don't sepcify the option and you do specify one of the other displays, the value is taken from the global **CONTOUR\_COLORS** which defaults to 16.

**DIRECTION** = *<integer>* For the **SHADE** option, this indicates the direction of the pseudo-sun used for plotting slope values. There are 8 possible directions with the values 0 → 7 with the names shown in Table 3.3. If you don't specify this option and do specify **SHADE**, the direction is taken from the default global variable **CONTOUR\_DIRECTION** that is normally set to **NW**.

Name	Value	Description
NW	0	Sun in the north west.
N	1	Sun straight north.
NE	2	Sun north east.
E	3	Sun straight east.
SE	4	Sun south east.
S	5	Sun straight south.
SW	6	Sun south west.
W	7	Sun straight west.

Table 3.3: Sun angles for slope shading

**ELEVATION\_COLOR** = *<integer>* Using this option specifies that elevation coloring should be used as a background for the contour lines. The integer value corresponds to the color of the lowest value. The colors are scaled to match the number of colors from the **COLORS** keyword or default.

**SHADE** = *<integer>* When this option is present, elevation shading is drawn before the contour lines. The slope is computed according to one of the 8 directions in Table 3.3, that is, in the direction of the sun. A positive slope results in a bright color and a negative slope in a darker color. The color values must be set by the **COLORMAP**

function (see page 52) and the first of these colors is the integer value of this keyword.

**XBASE** = *<number>* Displaying the X and Y values associated with Z values (the array values) can be modified with a linear equation. The value displayed with tic marks *X* is based on the the array index *i* as:

$$X = iXINC + XBASE$$

If the keyword does not occur in the list, then the global default **CONTOUR\_XBASE** (normally 0.0) is used that when combined with the default **CONTOUR\_XINC** value of 1.0 gives the array index.

**XINC** = *<number>* This is the multiplying value in the linear equation used to compute *X* above.

**YBASE**= *<number>* Displaying the X and Y values associated with Z values (the array values) can be modified with a linear equation. The value displayed with tic marks *Y* is based on the the array index *i* as:

$$Y = iYINC + YBASE$$

If the keyword does not occur in the list, then the global default **CONTOUR\_YBASE** (normally 0.0) is used that when combined with the default **CONTOUR\_YINC** value of 1.0 gives the array index.

**YINC** = *<number>* This is the multiplying value in the linear equation used to compute *Y* above.

**ZHIGH** = *<number>* Normally the low and high values are computed during initialization. This overrides the high value for the shade and color contours.

**ZLOW** = *<number>* Normally the low and high values are computed during initialization. This overrides the low value for the shade and color contours.

**Variables Set:** As a result of building the axes, the following variables are set:

**MINX** The minimum X value used to generate the axes.

**MINY** The minimum Y value used to generate the axes.

**SCALEX** The scale computed for the X axis. An X pixel value will be:

$$X_{pixel} = XLL + \frac{x - MINX}{SCALEX}$$



**SCALEY** The scale computed for the Y axis. A Y pixel value will be:

$$Y_{pixel} = YLL + \frac{y - MINY}{SCALEY}$$

**XLL** The lower left corner X pixel coordinate of the graph area.

**YLL** The lower left corner Y pixel coordinate of the graph area.

**XUR** The upper right corner X pixel coordinate of the graph area.

**YUR** The upper right corner Y pixel coordinate of the graph area.

**Errors Detected:** In addition to errors detected by the axis and grid display code, the following errors are detected.

**At line *nn*, lower left *x=xx*, and right *x=xx* are probably bogus**  
Modifications of LEFT\_PIXELS and RIGHT\_PIXELS for the axes value result in them being out of order (high less than low) which can't be.

**At line *nn*, X low (*xxx*) is greater than or equal to the high value (*xxx*)**  
The modifications XLOW and/or XHIGH result in the minimum and maximum values being out of order (high less than low).

**At line *nn*, missing ELEVATION for contour plot** You must specify an array for the routine to plot.

**At line *nn*, ELEVATION was not an array** The value you did give for the ELEVATION keyword was not an array.

**Known Deficiencies:**

1. Doesn't work with irregular spaced points.
2. Not much checking on good parameter values.
3. Can't associate UTM or australian coordinates very well yet.

**See Also:** COLOR command on page 50 and the default color values listed there, and READBIN on page 88 for reading binary arrays.

**Example:** The following example reads a binary byte array named `ba.bin` that has data for a simulated wave tank and plots its contours.

```
readbin cosv = "ba.bin" image width=640,height=480 contour
elevation=cosv,basis=25,xbase=0,xinc=1.0,ybase=0,yinc=1.0, alternate=5
ps "manual/contour.ps"
```

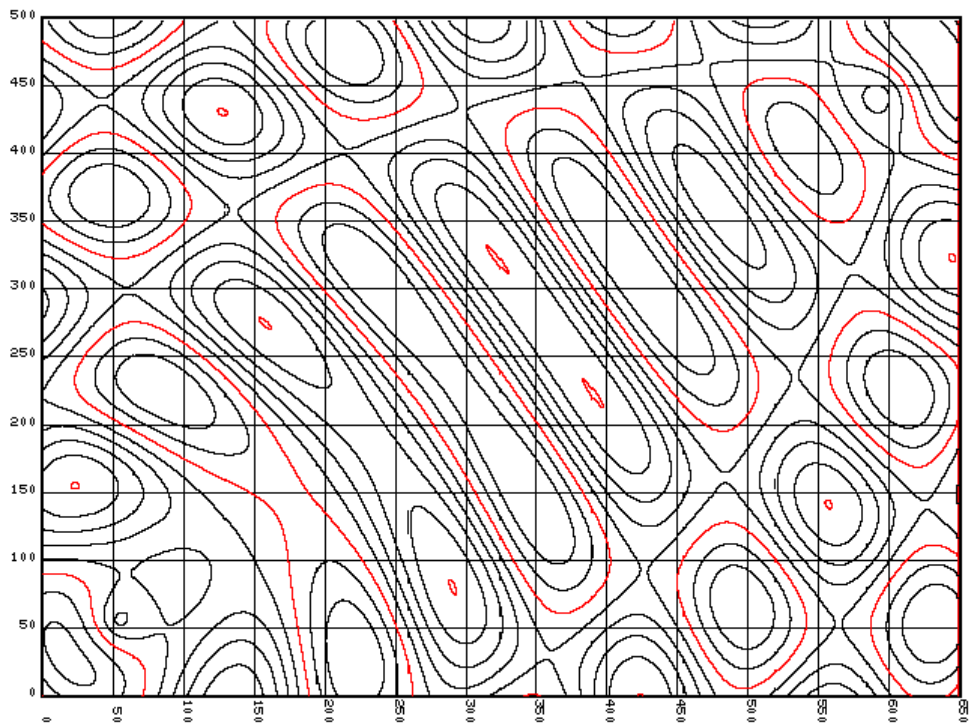


Figure 3.11: Wave tank contour plot.

You can add color values behind the contour lines. For example, you can use the elevation colormap to show the relative height values. The legend generated by this graph includes a description, range values, and a label. Note that the CONTOUR line is too long so it extends over two lines for expository purposes.

```
// Contour plot with color elevation.
READBIN cosv = "ba.bin"
IMAGE WIDTH=640,HEIGHT=480
CONTOUR ELEVATION=cosv,BASIS=25,XBASE=0,XINC=1.0,YBASE=0,YINC=1.0,ALTERNATE=5,
        COLOR2=WHITE ELEVATION_COLOR=ELEVATION_COLORS,COLORS=96
LEGEND
PS "contour_elev.ps"
```

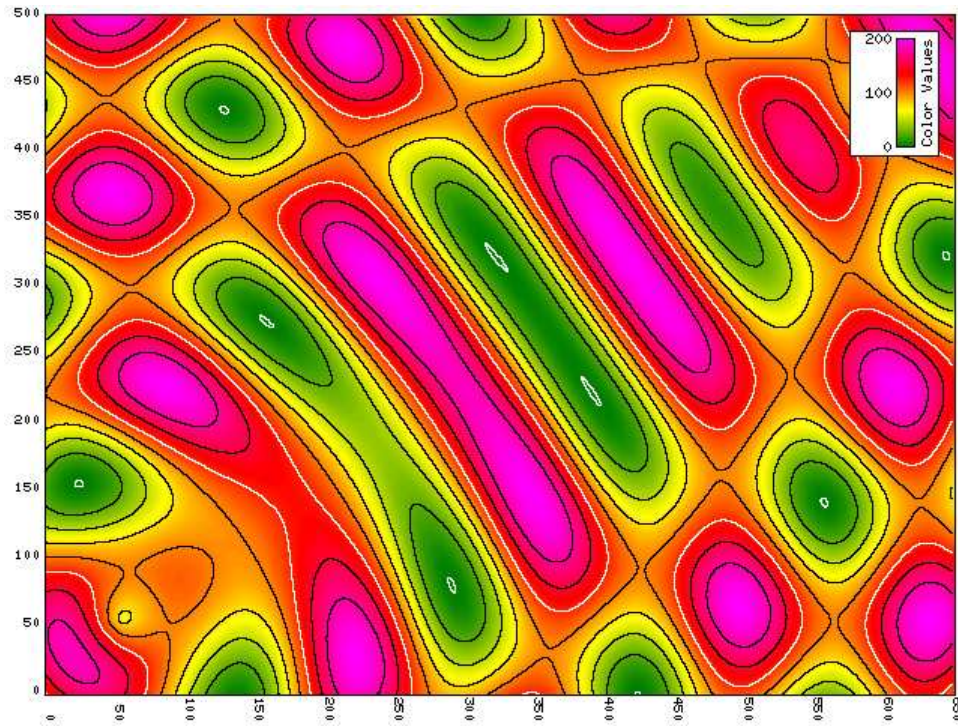


Figure 3.12: Contour plot with elevation coloring

The final example demonstrates the use of slope shading. We work with the same data but shade the slopes so that surface normals pointing to the north east are shaded more white and those to the south west are darker. Note that the plotting line is too long for the page so it is split to the next.

```
// Wave tank with shaded background.
READBIN cosv = "ba.bin"
IMAGE width=640,height=480
COLOR red,red=0.6, green=0.3,blue=0
COLOR green,red=.3,green=0.5,blue=.1
COLORMAP shade=1, count=16, start=30
CONTOUR ELEVATION=cosv,BASIS=25,XBASE=0,XINC=1.0,YBASE=0,YINC=1.0,COLOR1=green,COLOR2=RED,
        ALTERNATE=5,SHADE=30,DIRECTION=NE
PS "contour_shade.ps"
```

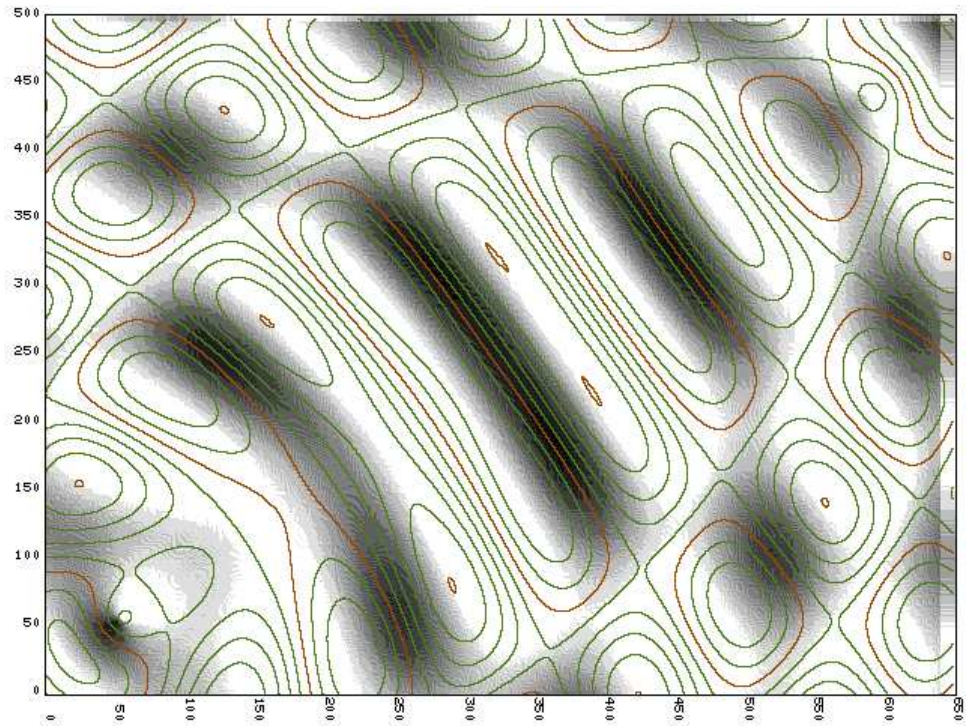


Figure 3.13: Contour plot with slope shading

## 3.12 DISPLAY

Display a graph on the screen.

---

### DISPLAY

**FILE** = *<string>*,  
**WITH** = *<string>*

---

**Description:** After a graph has been written to a file, this command can be used to call an operating system level display routine.

**Keywords:** The file name must be present but the display program is system dependent.

**FILE** = *<string>* This gives the name of a file to display. This key-value pair must be present or an error is signaled. The display program must be selected that can display the format of the file.

**WITH** = *<string>* If present, this names the program used to display the graph named. If not present, an operating system dependent routine is selected (see Table 3.4) that is stored in the **DISPLAY\_WITH** global variable.

OS	Program	Description
Linux	eog	Displays BMP, and GIF but not postscript images.
MacOS X	open	Displays all output types.
CYGWIN	explorer	Should work with most images but Postscript.
Windows	explorer	Should display just about everything.

Table 3.4: OS default display routines

**Errors Detected:** At line *nn*, **missing FILE to display** You must name the file to display.

At line *nn*, **FILE key is not a string** The name must be a string.

At line *nn*, **WITH key does not have a string value** The WITH parameter, when present, must have a string value.

**Known Deficiencies:** Doesn't check that the program will actually display the kind of file sent to it.

**See Also:** **BMP** on page 41, **GIF** on page 71, **PS** on page 87.

### 3.13 DRAW ARROW

Draw an arrow.

---

**DRAW ARROW**  $\langle \textit{number}_{x1} \rangle$ ,  $\langle \textit{number}_{y1} \rangle$ ,  $\langle \textit{number}_{x2} \rangle$ ,  $\langle \textit{number}_{y2} \rangle$ ,  
**COLOR** =  $\langle \textit{integer} \rangle$ ,  
**LENGTH** =  $\langle \textit{integer} \rangle$ ,  
**STYLE** =  $\langle \textit{integer} \rangle$ ,  
**WIDTH** =  $\langle \textit{integer} \rangle$

---

**Description:** Draw a line between two points with an arrow head on the end. The style and values are shown Figure 3.14.

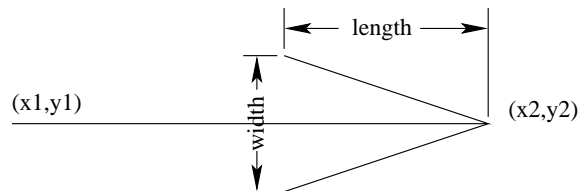


Figure 3.14: An arrow

**Keywords:** The keyword list, if present, must be preceded by the 4 numbers. If these numbers are integers, they are pixel positions. If they are floating-point values they are a percentage of the image area.

**COLOR** =  $\langle \textit{integer} \rangle$  Indicates what color the line should be drawn in. If this keyword isn't present, the value stored in the global **ARROW\_COLOR** is used (defaults to 0).

**LENGTH** =  $\langle \textit{integer} \rangle$  Sets the arrow length (see Figure 3.14) parameter - a value in pixels. If this keyword isn't present the value of the global **ARROW\_LENGTH** is used. Its default value is 10.

**STYLE** =  $\langle \textit{integer} \rangle$  Indicates the line style to use where the 32 bit integer value has a 1 if a COLOR pixel is to be drawn and 0 to not be drawn. If the option isn't present, the value is taken from the global variable **ARROW\_STYLE**. The default value is -1: all pixels on.

**WIDTH** =  $\langle \textit{integer} \rangle$  Sets the arrow width (see Figure 3.14) parameter to this number of pixels. If this parameter is not included, then the value of the **ARROW\_WIDTH** global is used. It defaults to a value of 4.

**Variables Set:** The location of the arrow's head is saved.

**LASTX** Set to the integer coordinate value of  $x_2$ .

**LASTY** Set to the integer coordinate value of  $y_2$ .

**Errors Detected:** The values for the four keywords must be integers or an error will be signaled.

**Known Deficiencies:**

**See Also:** DRAW BEZIER on page 64, DRAW LINE on page 62, and DRAW TEXT on page 68.

**Example:** The following example draws a few simple arrows and then shows up to put an arrow on the end of a Bezier curve.

```
image width=320,height=200
draw arrow 50, 50, 150, 50, color=red
draw arrow 50, 50, 125, 60, length=20
draw arrow 50, 50, 100, 70, width=10
draw bezier (50, 50) (75,90) (150,95) (220,30) (190, 100) color=blue, count=5, style=252645135
draw arrow prevx, prevy, lastx, lasty, color=blue, width=6
bmp "arrow.bmp"
```

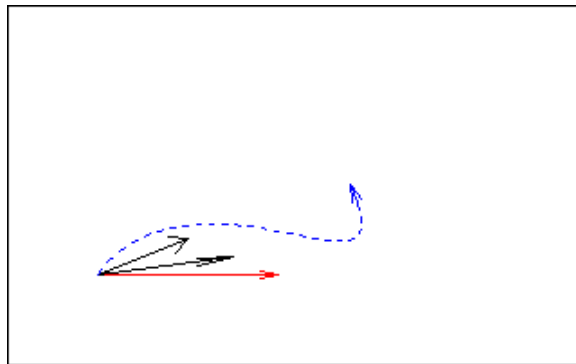


Figure 3.15: DRAW ARROW with curves





**Known Deficiencies:**

**See Also:** DRAW LINE on page 64, DRAW TEXT on page 68 and DRAW ARROW on page 62.

**Example:** The following egregious example draws a cloud with some text.

```
image width=320,height=200
draw bezier (50,50) (40,75) (65,75) color=blue
draw bezier (lastx,lasty) (60,96) (90,100) color=blue
draw bezier (80,98) (100,140) (120,110) color=blue
draw bezier (117,115) (140,105) (140,90) (130,85) color=blue
draw bezier (135,89) (160,80) (140,65) color=blue
draw bezier (144,67) (160,55) (120,40) (110,55) (90,40) (50,52) color=blue
draw text font=F8x13bold,x=75,y=80,MESSAGE="Cloud", color=6
draw text font=F8x13bold, x=65,y=58,MESSAGE="Computing", color=7
draw line 90, 45,110,0, color=blue
draw line 100,45,120,0, color=blue
draw line 110,45,130,0, color=blue
draw line 120,45,140,0, color=blue
bmp "bez.bmp"
```



Figure 3.16: Bezier curves making a callout cloud

### 3.15 DRAW LINE

Draw a straight line between two points.

---

**DRAW LINE**  $\langle number_{x1} \rangle, \langle number_{y1} \rangle, \langle number_{x2} \rangle,$   
 $\langle number_{y2} \rangle,$   
**COLOR** =  $\langle integer \rangle,$   
**STYLE** =  $\langle integer \rangle$

---

**Description:** Draws a line on the image between two points  $(x1, y1) \rightarrow (x2, y2)$ .

**Keywords:** The keyword list, if present, must be preceded by the 4 numbers. If these numbers are integers, they are pixel positions. If they are floating-point values they are a percentage of the image area.

**COLOR** =  $\langle integer \rangle$  Indicates what color the line should be drawn in. If this keyword isn't present, the value stored in the global **LINE\_COLOR** is used (defaults to 0). **LINE\_COLOR** is also shared with the LINE graph command.

**STYLE** =  $\langle integer \rangle$  Indicates the line style to use where the 32 bit integer value has a 1 if a COLOR pixel is to be drawn and 0 to not be drawn. The default value is taken from the **LINE\_STYLE** global variable which defaults to -1: all pixels on. This global is also shared with the LINE graph command.

**Variables Set:** The last location drawn to is saved.

**LASTX** Set to the integer coordinate value of  $x2$ .

**LASTY** Set to the integer coordinate value of  $y2$ .

**Errors Detected:** The values for the two keywords must be integers or an error will be signaled.

**Known Deficiencies:**

**See Also:** DRAW BEZIER on page 64, DRAW ARROW on page 62, and LINE on page 76.

**Example:** This example draws a square box.

```
image width=320,height=200
draw line 0, 0, 319, 0
draw line LASTX, LASTY, 319, 199
draw line LASTX, LASTY, 0, 199
draw line LASTX, LASTY, 0, 0
draw line 50, 50, 270, 50, color=red
```

```
draw line lastx, lasty, 270, 150, color=green  
draw line lastx, lasty, 50, 150, color=blue  
draw line lastx, lasty, 50, 50, color=magenta  
bmp "sq.bmp"
```



Figure 3.17: DRAW LINE example with square

### 3.16 DRAW TEXT

Draw text on an image.

---

#### DRAW TEXT

**COLOR** = *<integer>*,  
**FONT** = *<font>*,  
**X** = *<integer>*,  
**Y** = *<integer>*,  
**ANGLE** = *<number>*,  
**MESSAGE** = *<string>*

---

**Description:** Draw text somewhere on the image. This allows you to draw random text in any orientation, anywhere on the image using one of the 5 built-in fonts.

**Keywords:** The following keywords are defined:

**ANGLE** = *<number>* This the angle (in degrees) to draw the text at. As the fonts are pixels, the angles work best at multiples of 90 degrees. If you don't present this keyword, the system uses the value of the global **TEXT\_ANGLE** that normally has a default value of 0.0.

**COLOR** = *<integer>* Sets the color for drawing the text. If you don't include this option, the value of the global **TEXT\_COLOR** is used instead; the default value is usually 0.

**FONT** = *<imagefont>* Indicates which font to use. The argument must be one of those in Table 3.5. The width and height are in pixels and the zero value is the number of pixels from any descender to the bottom pixel.

Name	Width	Height	Zero
F5x7	6	7	1
F6x10	6	10	1
F8x13	8	13	2
F8x13bold	8	13	2
F9x15	9	15	3
F12x24	12	24	2

Table 3.5: Defined fonts.

**MESSAGE** = *<string>* This keyword must be present; there is no default value. The value must be a string to display. If the string

has  $\backslash n$  in it, the Y axis pixel will be reduced by the font height times the  $\cos(\frac{180angle}{\pi})$  and the X axis pixel by the font height times the  $\sin(\frac{180angle}{\pi})$ .

**X** = *<number>* This keyword must be present; there is no global substitute. If the value is an integer, it indicates the X pixel where the first text character begins. If it's a floating-point value, it becomes a percentage of the image width with 1.0 being 1 pixel beyond the far right border and 0 the leftmost pixel.

**Y** = *<number>* This keyword must be present; there is no global substitute. If the value is an integer, it indicates the X pixel where the first text character begins. If it's a floating-point value, it becomes a percentage of the image height with 1.0 being one pixel above the top border and 0.0 being the bottom border.

**Variables Set:** These are set to the next reasonable location to draw more text to.

**LASTX** Set to the integer coordinate as rotated where new text would start on the next line.

**LASTY** Set to the integer coordinate as rotated where new text would start on the next line.

**Errors Detected:** The following errors are detected:

**At line *nn*, value of FONT keyword is not an image** The value of the FONT keyword must be a known image font (see Table 3.5 for names).

**At line *nn*, fnt *fff* is unknown** The font listed is not known. The only defined fonts are those in Table 3.5.

**At line *nn*, missing [X/Y location parameter]** The X and Y parameters must be present.

**At line *nn*, missing text MESSAGE to draw** You must have a message to draw as well.

**Known Deficiencies:** As the fonts are stored as pixels rather than outlines, drawing off 90 degree increments is not recommended. The characters will have holes in them.

**See Also:** DRAW ARROW on a page 62, DRAW BEZIER on page 64 and DRAW LINE on page 66.

**Example:** The following example shows text of the various colors and sizes and rotates some about a common center.

```

image width=320,height=200, color=0
yt = 200 - 20
draw text X=100, Y=yt, MESSAGE="This is some 5x7 text", FONT=F5x7, color=1
yt = yt - F6x10_HEIGHT
draw text X=100, Y=yt, MESSAGE="Here is 6x10 text", FONT=F6x10, color = 2
yt = yt - F8x13_HEIGHT
draw text X=100, Y=yt, MESSAGE="And now for 8x13", FONT = F8x13, color=3
yt = yt - F8x13bold_HEIGHT
draw text X=95, Y = yt, MESSAGE = "This is bold 8x13", FONT = F8x13bold, color=4
yt = yt - F9x15_HEIGHT
draw text X=75, Y=yt, MESSAGE="This is the F9x15", FONT=F9x15, color=5
yt = yt - F12x24_HEIGHT
draw text X=75, Y=yt, MESSAGE="Here is F12x24", FONT=F12x24, color=7

draw text x=160, Y=60, MESSAGE=" 0 deg", ANGLE=0, color=1
draw text x=160, Y=60, MESSAGE=" 90 deg", ANGLE=90, color=1
draw text x=160, Y=60, MESSAGE=" 180 deg", ANGLE=180, color=1
draw text x=160, Y=60, MESSAGE=" 270 deg", ANGLE=270, color=1
ps "dt.ps"
display file="dt.bmp"

```



Figure 3.18: DRAW TEXT example

## 3.17 GIF

Create a GIF format image.

---

### GIF <string>

---

**Description:** This command takes the current state of the image (if there is one) and generates a GIF format file using the string for a name. This will be an 8 bit color map image of the size set by the image command. This will generate a considerably smaller size file than the BMP or PS commands..

**Keywords:** None.

**Errors Detected:** The following errors are detected:

**At line *nn*, no image to write** No image is available. You must use an **IMAGE** command before this.

**At line *nn*, missing file name string** The command wasn't followed by a string file name.

**At line *nn*, couldn't open *xxx* for output because ...** The writing routine couldn't open the output file. The system reason is given. Typically the directory you want to write to is protected or doesn't exist.

**See Also:** **DISPLAY** on page 61, **BMP** on page 41, **IMAGE** on page 72, **PS** on page 87.

### 3.18 IMAGE

Create a blank image to draw on.

---

```
IMAGE [ <name> ]
      WIDTH = <integer>,
      HEIGHT = <integer>,
      COLOR = <integer>
```

---

**Description:** Build a blank image to dump. You must execute this command before any that generate a graphical output. It creates a blank image of the requested size, and sets all its pixels to the specified background color.

If you include the optional name (a symbol), the image will be stored in this variable. This allows you to create multiple graphs and put them into a single image.

**Keywords:** The three optional keywords each have their own default global value of the same name. Using the keyword values overrides the global default values. However, the keyword values are not sticky, they apply to this command only.

**COLOR** = <integer> Sets the background color to use. Typically this is saturated white assigned to the global variable **WHITE** (by default 1). This is the default for the **BACKGROUND\_COLOR** global

**HEIGHT**=<integer> The image height must be an integer greater than or equal to 100. If this keyword is not present, the global value of **HEIGHT** will be used (defaults to 480).

**WIDTH**=<integer> The width must be an integer greater than or equal to 100. If this keyword is not present, the global value of **WIDTH** will be used (defaults to 640).

**Errors Detected:** The following errors can occur.

**At line *nn*, Symbol *KEYWORD* must have an integer value** The global value of one of the default keyword values is not an integer.

**At line *nn*, *ww* x *hh* is an unreasonable image size** One or other of the width *ww* or height *hh* is less than 100.

**At line *nn*, keyword *KEYWORD* must have an integer value** This occurs when a keyword value is not an integer (as opposed to the default global symbol value).

**See Also:** **BMP** on page 41, **PS** on page 87, and **COLOR** on page 50.

**Example:**



## 3.19 LEGEND

Add legend to graph.

---

### LEGEND

**BACKGROUND\_COLOR** = *<integer>*

**DECORATION\_COLOR** = *<integer>*

**DECORATION\_STYLE** = *<integer>*

**FONT** = *<font>*

**MESSAGE** = *<string>*

**X** = *<integer>*

**Y** = *<integer>*

---

**Description:** Most of the plotting commands generate additional data about their generated plot. For example, the **LINE** plot saves the line color, line style, and a label for each line plotted. These can be formatted and displayed anywhere on the plot with this command. In addition, you can add messages to this information, change its font, etc.

**Keywords:** The **LABEL** keyword on the various plots provides text for the legend display. Each plot type generates a stereotypical display within the legend: scatter plots show the symbol, size, and color, line plots show the line color and style, etc.

**BACKGROUND\_COLOR** = *<integer>* The legend image is written on top the plot image. The legend's area is normally set by the color in the global variable **LEGEND\_BACKGROUND\_COLOR** (typically white = 1). You can override this value with the **BACKGROUND\_COLOR** key.

**DECORATION\_COLOR** = *<integer>* The legend area is surrounded by a frame which uses this color as does the text collected from **LABEL** keys on the plots. If you don't specify this keyword, the decoration color is taken from the **LEGEND\_DECORATION\_COLOR** that normally has a value of 0 for black.

**DECORATION\_STYLE** = *<integer>* Line drawing for decorations uses this style value (see section 4.2 on page 118). If you don't specify this option, the value stored in the global **LEGEND\_DECORATION\_STYLE** is used (typically -1 for solid lines).

**FONT** = *<font>* This option specifies the font for displaying text in the legend. All legend text uses the same font. If you don't specify the font, the default in the global **LEGEND\_FONT** global that normally defaults to F6x10.

**MESSAGE** = *< string >* You can add text to the legend display as well.

This option can be repeated many times and the strings attached to them appear starting at the legend's bottom, one line per message.

**X** = *< integer >* If you don't specify this option, the legend is automatically placed near the upper right hand corner of the plot display area (the global variables **XUR** and **YUR**). You can specify a better placement by including this option and its corresponding **Y** value.

**Y** = *< integer >* If you don't specify this option, the legend is automatically placed near the upper right hand corner of the plot display area (the global variables **XUR** and **YUR**). You can specify a better placement by including this option and its corresponding **X** value. You are not required to include both options.

**Errors Detected:** In addition to errors detected when values are not the right type, the following are detected:

**At line *nn*, FONT value is not a font** The value for the FONT option is not a known font. The font's are named and should not be entered as strings.

**At line *nn*, value of MESSAGE is not a string** The MESSAGE option must have a string value.

**At line *nn*, LEGEND entry label is not a string** This is an internal error and should not happen. If it does, something is misprogrammed.

**Example:** There are legend examples scattered throughout the text. The following shows how to generate a legend with extra statistics. Both the scatter plot and legend commands are too long for the text and have been extended over multiple lines (not syntactically correct).

```

READCSV "minesweeper.csv"
IMAGE WIDTH=640, HEIGHT=480
AXES_LABEL_FONT = F12x24
SCATTER SIZE=4,COLOR=RED,X=Count,Y=Time,LABEL="Count vs time",SYMBOL=TRIANGLE,
    XLABEL="Mean "+mean(Time),YLABEL="Seconds"
LEGEND MESSAGE="Count Mean = "+Mean(Count),
    MESSAGE="Count Standard Deviation "+standarddeviation(Count),
    MESSAGE="Time range "+min(Time)+"->"+max(Time)+" seconds",
    MESSAGE="MineSweeper",FONT=F8x13,Y=300
BMP "test3.bmp"
DISPLAY FILE="test3.bmp"

```

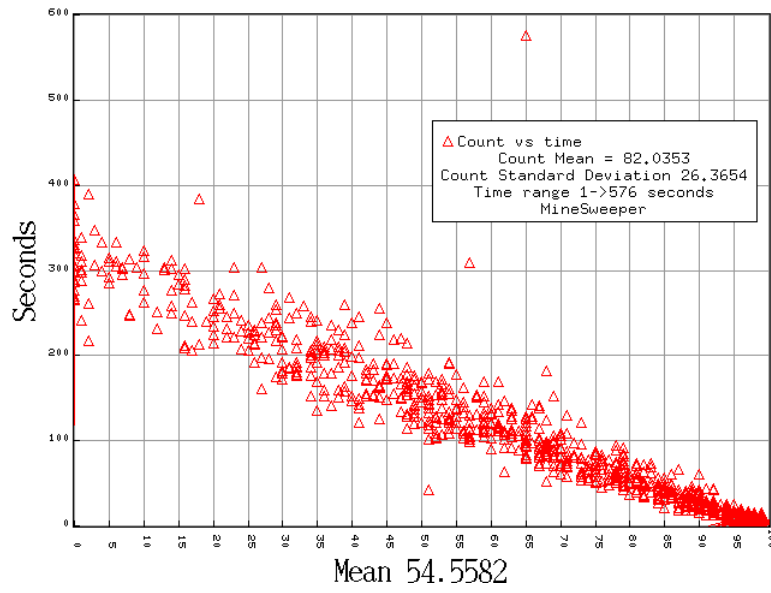


Figure 3.19: Legend construction and display

## 3.20 LINE

Line graph data.

---

### LINE

```

COLOR = <integer>,
STYLE = <integer>,
X = <list,vector>,
Y = <list,vector>
LABEL = <string>,
LINE = <any>,
MARKS = <integer>,
SIZE = <integer>,
MARK_COLOR = <integer>,
... Axes keywords ...
... Grid keywords ...

```

---

**Description:** Plots a line graph of X and Y data. Multiple data sets are permitted - they can be plotted with different colors and line styles.

**Keywords:** See the keywords associated with the axes and grid options that can be part of this command. In addition, the following are supported:

**COLOR** = <integer> Indicates the color index to use for the next line display. If no color is given, the value in the global variable **LINE\_COLOR** (typically **BLACK**) is used.

**LABEL** = <string> Causes an entry to be made in the legend for this plot. A short line of the currently selected color is drawn and the text of the string follows it. If you don't include a **LABEL**, no entry in the legend will be made. The label for a line should be included after the color and style are set, but before the actual plotting (**X** and **Y** values).

**STYLE** = <integer> This option gives a 32 bit bit pattern for displaying the line. The default is a solid line (code -1) taken from the global variable **LINE\_STYLE**.

**X** = <list,vector> This required option must have a list or vector as its value. If a Y keyword has already been found, the line is plotted and the X and Y values are erased to wait for more.

**Y** = <list,vector> This required option must have a list or vector as its value. If an X keyword has already been found, the line is plotted and the X and Y values are erased in preparation for the next line.

**LINE** = *<integer>* You can place marks on line points similar to the **SCATTER** plot. If you include this in the form **LINE** = **none**, the line will not appear but marks will. This is not sticky, the next data set will revert to normal line drawing.

**MARKS** = *<integer>* Marks will be placed on the line as in the **SCATTER** plotter. The integer value must be one of those in Table 3.7 on page 94. The value is not sticky and the next data values revert to a plain line.

**SIZE** = *<integer>* The drawn marks on top the line will have this size which works best if the value is a multiple of 2. If no value is given, the default size is 3.

**MARK\_COLOR** = *<integer>* Sets the mark color to this value if marks are present. The value defaults to 0 and is not sticky.

The order of the above keywords is important as they can occur multiple times for multiple lines. **COLOR** and **STYLE** are sticky between lines until changed by further occurrences. **X** and **Y** are not and must be specified for each line to be plotted.

**Variables Set:** As a result of building the axes, the following variables are set:

**MINX** The minimum X value used to generate the axes.

**MINY** The minimum Y value used to generate the axes.

**SCALEX** The scale computed for the X axis. An X pixel value will be:

$$X_{pixel} = XLL + \frac{x - MINX}{SCALEX}$$

**SCALEY** The scale computed for the Y axis. A Y pixel value will be:

$$Y_{pixel} = YLL + \frac{y - MINY}{SCALEY}$$

**XLL** The lower left corner X pixel coordinate of the graph area.

**YLL** The lower left corner Y pixel coordinate of the graph area.

**XUR** The upper right corner X pixel coordinate of the graph area.

**YUR** The upper right corner Y pixel coordinate of the graph area.

**Errors Detected:** In addition to errors detected by the axis display code, the following errors are detected.

**At line *nn*, COLOR is not an integer** The value of a **COLOR** keyword is not a valid integer.

**At line *nn*, STYLE is not an integer** The value of a **STYLE** keyword is not a valid integer.

**At line  $nn$ , mismatched structures or lengths are different** The X and Y values are not of the same length or type. Both must be lists or both must be identical vectors.

**Known Deficiencies:**

**See Also:** **COLOR** command on page 50 and the default color values listed there. The **SORT** command on page 102 must be used to assure that values are sorted along at least one axis.

**Example:** The following example reads cos and sin curves and plots them on the same surface.

```
readcsv "cos.csv"
readcsv "sin.csv"
image width=640, height=480
line color=red,x=X,y=cosX,color=blue,x=X,y=sinx,XLABEL="Plotted by "+VERSION
bmp "test4.bmp"
```

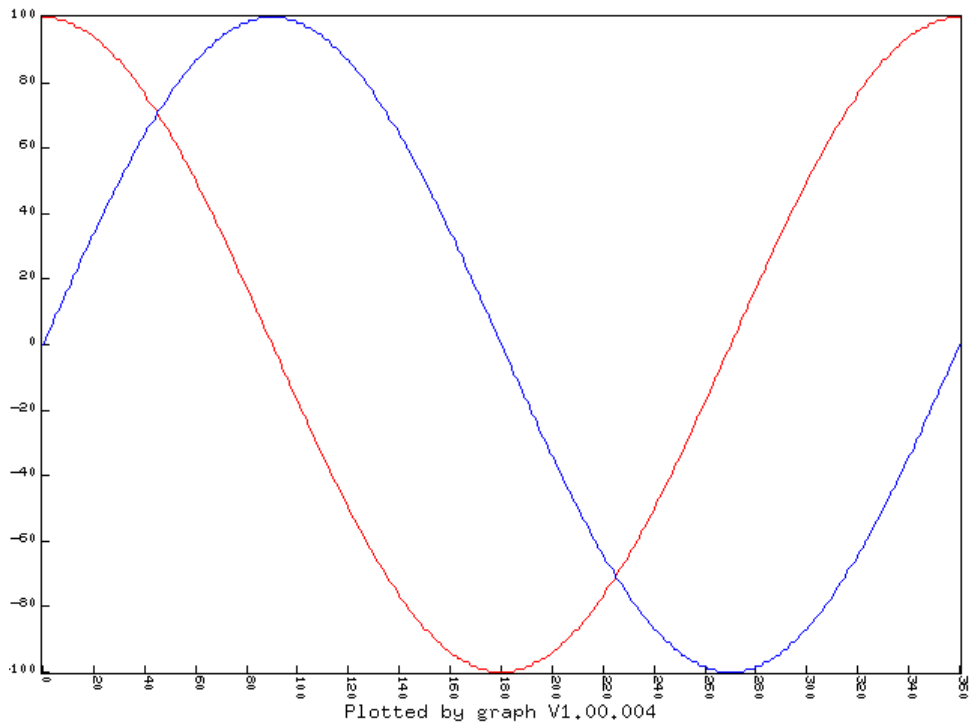


Figure 3.20: Multiple line graphs.

## 3.21 LLSQ

Generate data for linear least-squares fit to data.

---

**LLSQ**  $\langle X - symbol \rangle \langle Y - symbol \rangle$

---

**Description:** Compute the linear-least-squares fit equation for the set of X and Y points  $\langle X - symbol \rangle$  and  $\langle Y - symbol \rangle$ . These are lists or vectors of points from input or other calculations.

The symbols must have list or vector values. They must have the same number of elements and their contents must be numeric. The results are the coefficients of  $y = mx + b$  for a line that minimizes the error squared for each pair of X and Y points.

The command creates two new symbols for a two point line based on this equation. The first point is the point with X set to the minimum value of the original X values but with a Y value from the linear equation computed. The second is the same, but the X value is the largest of the original values. These are assigned to the names of the original symbols but are prefixed with **LLSQ\_...**. Thus, for two symbols **X**, and **COSX**, the new line created would have coordinates stored in **LLSQ\_X** and **LLSQ\_COSX**.

**Keywords:** None.

**Errors Detected:** At line *nn*, for linear least squares, **XY list has non-number in it**  
Something in a list wasn't a number. The either X or Y is specified.

**At line *nn*, need at least 2 entries for linear least squares** You need at least 2 points to fit the line to.

**At line *nn*, different length lists for linear least squares** You must have the same number of X and Y points - they're pairs.

**At line *nn*, vertical line for linear least squares** You can't fit a linear equation as above to a vertical line.

**At line *nn*, for linear least squares missing XY values** The parser couldn't find one or both of the data points to fit.

**At line *nn*, for linear least squares, 'sss' has no value** One of the variables *sss* has no value.

**At line *nn*, for linear least squares, 'xxx' is a list but 'yyy' is not**  
The Y value is not a list.

**See Also:**

**Example:** Fits a straight line to some in-line data. Not all the data is shown.  
The result of the computation is two new lists in the variables **llsq\_x** and **llsq\_cosx**.

```
readcsv
x,cosx
3,100
4,100
6,99
8,99
.
.
.
270,0
350,100
360,100
EOF
llsq x cosx
image width=640, height=480
line color=red,x=x,y=cosx,color=blue,x=llsq_x, y=llsq_cosx
bmp "test6.bmp"
```

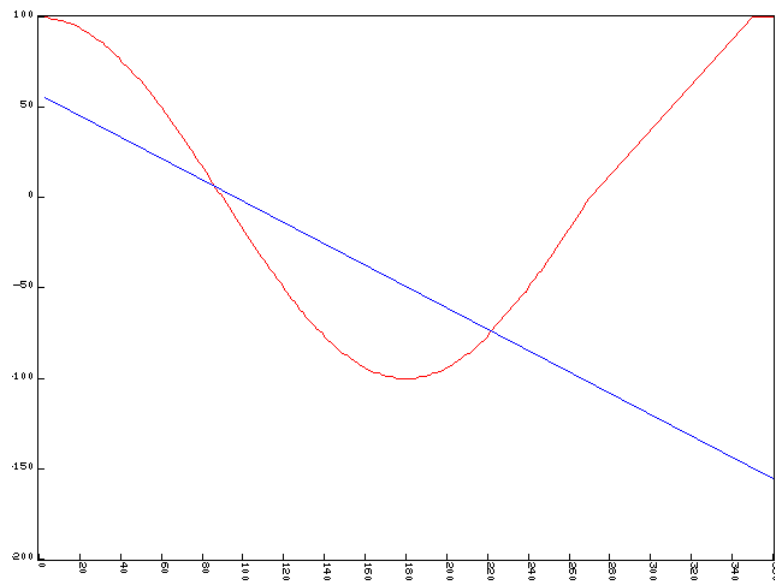


Figure 3.21: Linear Least Squares fit to partial COS curve



## 3.22 PIE

Pie chart.

---

### PIE

**ANGLE** = *<number>*,  
**COLORS** = *<list>*,  
**NAMES** = *<list, vector>*,  
**OUTLINE**,  
**OUTTAKE** = *<integer>*,  
**PIXELS** = *<integer>*,  
**RADIUS** = *<number>*,  
**START\_ANGLE** = *<number>*,  
**VALUES** = *<list, vector>*,  
*... Some axes keywords ...*  
**AXES\_LABEL\_FONT** = *<image font>*,  
**DECORATION\_COLOR** = *<integer>*,  
**XLABEL** = *<string>*,  
**YLABEL** = *<string>*

---

**Description:** Draw a pie chart from a list or vector of numbers. The values need not add up to 100%, but are normalized so that the total set encompasses 360 degrees. You can also provide a list of labels that match the values and a list of colors to cycle through. To emphasize a portion, a particular piece can be partially removed from the pie to highlight its contributions.

**Keywords:** In addition to a few of the axes control values, the following keywords are accepted by the pie charter.

**ANGLE** = *<number>* The pie slices are wedge shaped polygons with the distance between round points determined by this central angle (in degrees) and the radius. If the option is not present, the value is taken from the global **PIE\_ANGLE** that defaults to  $\frac{1}{2}$  degree.

**COLORS** = *<list>* Each pie slice is drawn with a color selected in sequence from a list either provided by the user with this keyword or taken from the default list **PIE\_COLORS**. The list need not have as many members as the list or vector of values. When the list runs out, color selection starts over. The default list is the 6 colors, **RED**, **YELLOW**, **GREEN**, **CYAN**, **BLUE**, and **MAGENTA**.

**NAMES** = *<list, vector>* This option provides a list or vector of tags for each pie slice. The type and number must correspond to the number of values.

**OUTLINE** If this keyword is present, Each pie wedge has an outline drawn around it using the decoration color. If the option is not present, no such outline is drawn.

**OUTTAKE** =  $\langle integer \rangle$  If this option is present, the  $n$ th (starting at 0) pie wedge and its title slides out from the center along a line running down its center angle. The number of pixels moved can be changed. Only one pie wedge can be highlighted in this manner.

**PIXELS** =  $\langle integer \rangle$  If the **OUTTAKE** option is present, this value indicates how many pixels the wedge is moved from the center. If this option isn't present, the value is taken from the **PIE\_OUTTAKE\_PIXELS** global variable that defaults to 20.

**RADIUS** =  $\langle number \rangle$  Normally the pie size is computed to fit the available space. This computation is not always successful so the user can override the default computation with this option. Note that the pie center position always falls in the plot areas center suitably modified for the presence of **XLABEL** and **YLABEL**.

**START\_ANGLE** =  $\langle number \rangle$  The first pie slice can start at any angle with this option. If it's not present, then the first angle is taken from the **PIE\_START\_ANGLE** global variable.

**VALUES** =  $\langle list, vector \rangle$  This key must be present and identifies the values for each pie wedge. A wedge  $v_i$ 's angle is:

$$W_i = 360 \frac{v_i}{\sum_{j=0}^{n-1} v_j}$$

If there are **NAMES** values, they must correspond in type and number or an error will be signaled.

The **AXES\_LABEL\_FONT**, **DECORATION\_COLOR**, **XLABEL**, and **YLABEL** keys can also be present and modify the display appropriately.

**Variables Set:** none.

**Errors Detected:** In addition to errors detected by the axis display code, the following errors are detected.

**At line  $nn$ , PIXELS value  $pp$  is  $< 1$  or  $> 100$**  The number of pixels offset must be greater than 0 (or why bother) and is reasonably less than 100.

**At line  $nn$ , missing required VALUES option** The **VALUES** key must be present or there is nothing to plot.

**At line  $nn$ , VALUES is not a list or vector** The **VALUES** key must be a list, integer vector, or floating-point vector or an error is signaled.

**At line *nn*, COLORS value must be a list** The COLORS key must be a list of small integers that match values in the current colormap.

**At line *nn*, ANGLE value *aaa* is bad** The angle value listed is less than or equal to 0 or greater than 45. Large angles (more than a few degrees) make for very choppy looking pies. Zero or negative values cause the plot algorithms to fail.

**At line *nn*, NAMES is not a vector like VALUES** For vectors both NAMES and VALUES must be vectors. You can't mix lists and vectors.

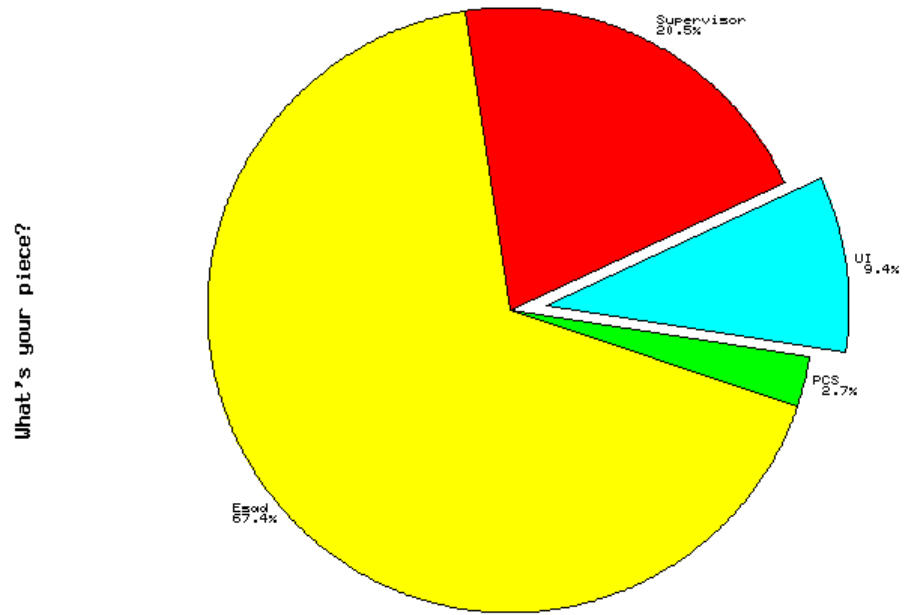
**At line *nn*, VALUES size *vv* not the same as NAMES *mm*** The vectors must be the same size.

**Known Deficiencies:** Doesn't work with vectors yet.

**See Also:** COLOR command on page 50 and the default color values listed there. COLORMAP on page 52 can also be used to create colors.

**Example:** The PIE command line is two long for one line and has been split to include all of it.

```
readcsv
system,colorindex,amperage
"Supervisor",2,0.759
"Esad",3,2.5
"PCS",4,0.1
"UI",5,0.35
EOF
image width=640,height=480
AXES_LABEL_FONT = F8x13bold
PIE NAMES=system,VALUES=amperage,COLORS=colorindex,OUTTAKE=3,PIXELS=25,OUTLINE,
    START_ANGLE=25,XLABEL="A PIE Chart",YLABEL="What's your piece?"
ps "../manual/pie.ps"
```



A PIE Chart

Figure 3.22: Pie chart example

### 3.23 PLACE

Copy one image onto another.

---

**PLACE** *<name>*,  
**X** = *<integer>*,  
**Y** = *<integer>*

---

**Description:** Take the image stored in the variable *<name>* and copy it to the current image with its lower left corner at pixel (X,Y).

**Keywords:** Both keys X and Y must be present.

**X** = *<integer>* Sets the X location where the lower left corner of the image to copy goes. Must be present.

**Y** = *<integer>* Sets the Y location where the lower left corner of the image to copy goes. Must be present.

**Errors Detected:** The following errors can occur.

**At line *nnn*, no image to place image into** You must have an image (created by the IMAGE command) before doing a place.

**At line *nnn*, near column *mm*, missing symbol to place** The first item following PLACE must be a defined symbol.

**At line *nnn*, near column *mm*, missing ,** There must be a comma following the symbol to place.

**At line *nnn*, missing X/Y parameter** You must have both X and Y in the parameter list.

**At line *nnn*, value of X/Y is not an integer** The value of these parameters must be integers.

**See Also:** IMAGE on page 72.

**Example:**

```
image sol width=300, height=200
READONED solitaire="solitaire.dat"
BUCKETS solb TRUNCATE, DATA=solitaire
AXES_LABEL_FONT = "F8x13bold"
BAR solb XLABEL="Cards Up, mean " + mean(solitaire), YLABEL="Count"
READCSV "minesweeper.csv"
image mine width=300, height=200
scatter size=2, color=BLUE, X=Count,Y=Time, symbol=TRIANGLE
image width=600,height=200
place sol,x=0,y=0
```

```
place mine,x=300,y=0  
bmp "comp.bmp"
```

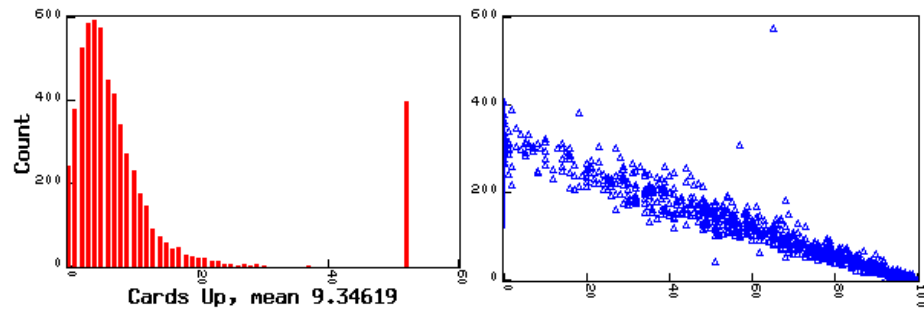


Figure 3.23: Two graphs in one using PLACE

## 3.24 PS

Generate a Postscript image.

---

**PS** <*string*>

---

**Description:** This command takes the current state of the image (if there is one) and generates a Postscript format file using the string for a name. This will be an 8 bit color map image of the size set by the image command.

**Errors Detected:** The following errors are detected:

- At line *nn*, must declare IMAGE first** No image is available. You must use an **IMAGE** command before this.
- At line *nn*, missing file name** The command wasn't followed by a string file name.
- At line *nn*, file '*fff*' couldn't be opened for output** The writing routine couldn't open the output file. Typically the directory you want to write to is protected or doesn't exist, or, some display program has it open.

**Keywords:** None.

**See Also:** **BMP** on page 41, **GIF** on page 71, **IMAGE** on page 72, and **DISPLAY** on page 61

### 3.25 READBIN

Read binary array data.

---

**READBIN**  $\langle symbol \rangle = \langle string \rangle$

---

**Description:** Read a binary file and store the value in the symbol listed. The binary files are stored as little-endian (least significant byte first) values. Unlike the ASCII file readers, input can not come directly from the control file.

Arrays are in storage order. The first element is [0,0], the next [0,1], and so on. The first subscript (the column) varies the least often.

The first byte of a binary file tells what type it is:

First Byte Value	File Type
0	The file is an integer vector (32 bit integers) of fixed size. The next 4 bytes indicate how many entries it has.
1	The file is a double-precision floating-point vector of fixed size. The next 4 bytes indicate how many entries it has.
2	The file is a byte array. The next 4 byte contain the number of lines (columns) and the subsequent 4 bytes the number of samples (rows).
3	The file is a 32 bit integer array. The next 4 byte contain the number of lines (columns) and the subsequent 4 bytes the number of samples (rows).
4	The file is a double-precision floating-point array. The next 4 byte contain the number of lines (columns) and the subsequent 4 bytes the number of samples (rows).

Table 3.6: Binary File Types

**Keywords:** None.

**Errors Detected:** The following errors are detected:

**At line  $nn$ , near column  $mm$ , missing symbol to assign to** The first token following the command must be a symbol.

**At line  $nn$ , near column  $mm$ , missing = sign** There must be an equal sign following the symbol.

**At line  $nn$ , near column  $mm$ , missing file name (string)** There must be a string following the = sign. Expressions are not permitted.



**At line nn, couldn't open 'fname' for input** - Couldn't open the input file named. It probably doesn't exist or the directory list is wrong. You can't include \$ variables in them.

**At line nn, file 'fname', binary file type dd unknown** - The file type (the first byte) is not 0 - 4 (the number is listed).

**At line nn, premature EOF on file 'fname'** The file wasn't long enough.

**See Also:** READCSV on page 90, READONED on page 92.

**Example:** Examples can be found in the **SURFACE** section on page 104 and the **CONTOUR** section on page 53.

## 3.26 READCSV

Read a CSV file.

---

**READCSV** [ *<string>* ]

---

**Description:** This command reads a simple CSV (comma separated values) file, a Microsoft format used by the EXCEL spreadsheet program among others. The first record of this file is a list of symbols separated by commas. These become variable names that get assigned a list of their corresponding column values. The file is read to the end and lists collected for each column. A missing value is treated as a floating-point 0. Random collections of integers, strings, and floating-point values are permitted.

If the command is followed by a string, then the input comes from a file of that name. If the command appears by itself, then input comes from the command file and is terminated by the symbol EOF.

**Keywords:** None.

**Errors Detected:** The following errors are detected:

**At line *nn*, missing string file name** The command must be either followed by a string or nothing. Numbers and symbols are not permitted.

**Premature EOF on file *fname*** The input file ended in the middle of a line.

**File *fname* has no header line** The first line of the input file is blank.

**File *fname* header line ends with comma** Just like it says. Empty column names are not permitted.

**File *fname* header has non symbol or string** Only symbols and strings are allowed on the header line.

**File *fname* has bad token in place of comma** Probably missing a comma.

**File *fname*, line *nnn*, too many values** There are more values on a data line than there were symbols on the header line. They must be the same.

**File *fname*, line *nnn*, non-numeric value** Only numbers and strings are allowed the the data fields.

**File *fname*, line *nnn*, invalid token after comma** Some other punctuation, string, or symbol appeared after a comma.

Note that if input is coming from the command line file, that file name will appear in error messages.

**See Also:** READONED on page 92.

**Example:** The following example reads a set of 4 data points in three columns.

```
READCSV
A, B, C, D
1,2,3,4
-2,3,5,22
-5,3,15,0
6,343,4,5
EOF
```

### 3.27 READONED

Read a file of ASCII numbers.

---

**READONED** < *symbol* > [ < *string* > ]

---

**Description:** Reads a file of numbers creating an ordered list that is stored in the symbol given. If a string follows this symbol, it is the name of a file to read. If no symbol follows, then input comes directly from the command file and is terminated by EOF.

**Keywords:** None.

**Errors Detected:** The following errors are detected:

**At line *nn*, missing symbol to assign list to** No symbol followed the command. Unlike CSV files, these files do not have their variable name embedded.

**At line *nn*, symbol '*xxx*' already defined** The symbol you listed already has a value. Use a different one.

**At line *nn*, missing file name to read** There needs to be a string or nothing following the symbol.

**At line *nn* in *fname*, non-numeric token** Only numbers are allowed in the file.

Note that if input is coming from the command line file, that file name will appear in error messages.

**See Also:** READCSV on page 90.

**Known Deficiencies:** Doesn't yet work with input from the command file.

**Example:** The following example reads a set of 10 data points from the command file. These are assigned (as a list) to the 'values' variable.

```
READONED values
1
2
0
9
22
34
3
0
23
10
EOF
```

## 3.28 SCATTER

Generate a scatter plot for one or more data sets.

---

### SCATTER

```

COLOR = <integer>,
LABEL = <string>,
LLSQ = <integer>,
POLYNOMIAL_COLOR = <integer>,
POLYNOMIAL_DEGREE = <integer>,
SIZE = <integer>,
SIZES = <list, vector>
SYMBOL = <integer>,
X = <list, vector>
Y = <list, vector>
... Axes keywords ...

```

---

**Description:** Plots a scatter graph of X and Y data. Multiple data sets are permitted - they can be plotted with different colors and symbols. Symbol sizes can be fixed or sized.

**Keywords:** See the keywords associated with the axes and grid commands that can be part of this command. In addition, the following are supported:

**COLOR** = *<integer>* Indicates the color index to use for the next plot data. If no color is given, the value in the global variable **SCATTER.COLOR** (typically **BLACK**) is used.

**LABEL** = *<string>* Indicates a label to attach to the current group that will be part of a legend. The current plot symbol will appear in the appropriate color followed by this string.

**LLSQ** = *<integer>* Indicates that if X and Y have been selected, that a linear least squares computation should be performed on the data and its line drawn with the color given. If this occurs before the **SYMBOL** keyword, the symbols will be drawn on top the line if after, above the line. A legend entry is made with the lines color and the **M** and **B** values as in:

$$y = Mx + B$$

At the completion of the **SCATTER** command, the global variables **M** and **B** are set to the two coefficients. If multiple **LLSQ** options are present, these values represent that last computed.

**POLYNOMIAL\_COLOR** = *<integer>* If you request a polynomial curve fit to the data by using the **POLYNOMIAL\_DEGREE** option, this color will be used to draw the line. If you don't specify a color, then the value of the global **POLYNOMIAL\_COLOR** will be used.

**POLYNOMIAL\_DEGREE** = *<integer>* This option will fit a polynomial of the degree selected to the current data set. The color is selected by the **POLYNOMIAL\_COLOR** value as described above. This option does not add an entry for the legend as it's much too large. Setting **NOISY** greater than 1 will dump the computed coefficients to standard output.

The coefficients will also be stored in a vector in the **POLYNOMIAL\_COEFFICIENTS** global variable. You can use the **INDEXED** function coupled with **DRAW TEXT** to display these values. A legend entry is not created.

**SIZE** = *<integer>* This option sets the radius of the scatter plot symbol. The size is sticky between plots. If no value is set, the default from the **SCATTER\_SIZE** global is used (usually 2).

**SIZES** = *<list, vector>* If this option is present, the next symbol specified takes its sizes from this list or vector. The type and size must be the same as that for **X** and **Y**. Unlike **X** and **Y**, the **SIZES** list is not sticky and disappears after a set is plotted.

**SYMBOL** = *<integer>* This option sets the symbol to use for the next data set. Table 3.7 defines the global variables and their values.

Global	Value	Description
CIRCLE	0	A circle (the default symbol).
SQUARE	1	A square centered on the point. Each side is $2SCATTER\_SIZE + 1$ .
TRIANGLE	2	An equilateral triangle.
PLUS	3	A centered plus sign. Each line is $2SCATTER\_SIZE + 1$ .
ASTERISK	4	A square asterisk.
FILLEDCIRCLE	5	A filled circle.
DIAMOND	6	A diamond.

Table 3.7: Scatter plot symbols

The plot only occurs when the **SYMBOL** keyword appears - there must be an **X** and **Y** value before it.

**X** = *<list, vector>* This required option must have a list or vector as its value.

**Y** = *<list, vector>* This required option must have a list or vector as its value.

The order of the above keywords is important as they can occur multiple times for multiple scatters. COLOR and STYLE are sticky between scatters until changed by further occurrences. X and Y are not and must be specified for each scatter to be plotted.

**Variables Set:** As a result of building the axes, the following variables are set:

**MINX** The minimum X value used to generate the axes.

**MINY** The minimum Y value used to generate the axes.

**SCALEX** The scale computed for the X axis. An X pixel value will be:

$$X_{pixel} = XLL + \frac{x - MINX}{SCALEX}$$

**SCALEY** The scale computed for the Y axis. A Y pixel value will be:

$$Y_{pixel} = YLL + \frac{y - MINY}{SCALEY}$$

**XLL** The lower left corner X pixel coordinate of the graph area.

**YLL** The lower left corner Y pixel coordinate of the graph area.

**XUR** The upper right corner X pixel coordinate of the graph area.

**YUR** The upper right corner Y pixel coordinate of the graph area.

**Errors Detected:** In addition to errors detected by the axis display code, the following errors are detected.

**At line *nn*, value of SIZE parameter not an integer** The symbol radius must be an integer.

**At line *nn*, value of COLOR parameter not an integer** The color parameter must be an integer ranging from 0 → 255.

**At line *nn*, value of SYMBOL is not an integer** The value for a symbol isn't an integer.

**At line *nn*, missing X or Y value to scatter plot** You must have an X and Y value assigned when the SYMBOL keyword appears.

**Known Deficiencies:** Needs clipping so symbols don't go outside of plot area. Only works with lists.

**See Also:** COLOR command on page 50 and the default color values listed there.

**Example:**

```

READCSV "minesweeper.csv"
BACKGROUND_COLOR = WHITE
image width=640,height=480
scatter size=4, color=RED, X=Count,Y=Time, symbol=TRIANGLE, llsq=BLUE,
    XLABEL = "Mean " + mean(Time), YLABEL = "Seconds"
legend MESSAGE="Count Mean = "+Mean(Count),
    MESSAGE="Count Standard Deviation "+standarddeviation(Count),
    MESSAGE="Time "+min(Time)+"->"+max(Time), MESSAGE="MineSweeper",
    FONT=F8x13, Y=300BMP "minesweeper.bmp"

```

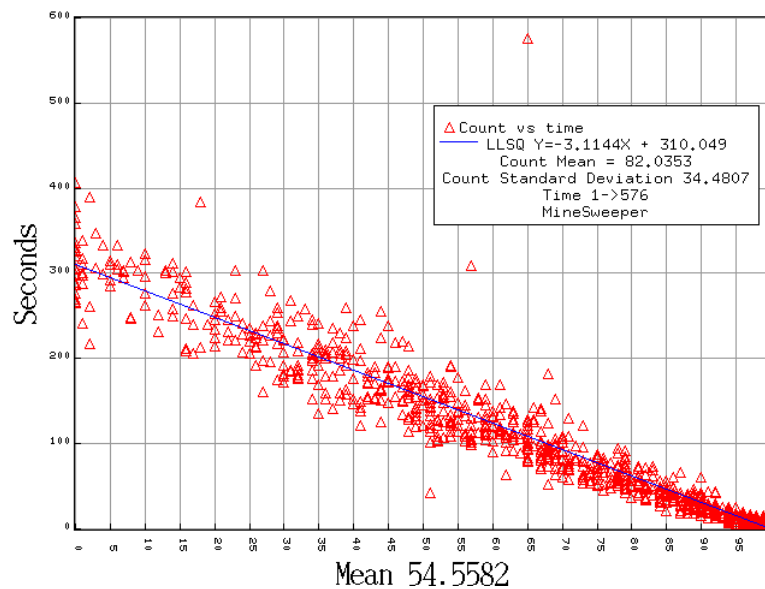


Figure 3.24: Scatter plot.

The next example demonstrates how symbol sizes can be modified. We read a small CSV file that looks like:

```

INDEX,COUNT,IMPULSE
1, 1, 1.25
4, 5, 10.0
6, 1, 40.0
7, 1, 80.0
9, 6, 320.0
10, 8, 640.0
11, 3, 1280.0
13, 1, 5120.0
14, 1, 10240.0

```



The script line for plotting is too long so is split across two lines. We take X from the INDEX value but Y is the log base two of the IMPULSE times the COUNT column. The SIZES are taken from the COUNT column but with a minimum size of 3. We set the high X value to 15 so that the final symbol is not truncated.

```
READCSV "scatter.csv"
IMAGE
SCATTER X=INDEX,Y=log2(IMPULSE*COUNT),SIZES=COUNT + 3,COLOR=RED,SYMBOL=TRIANGLE,
        XLABEL="MOTOR SIZE",XHIGH=15,YLABEL="TOTAL IMPULSE FLOWN"
PS "scatter2.ps"
```

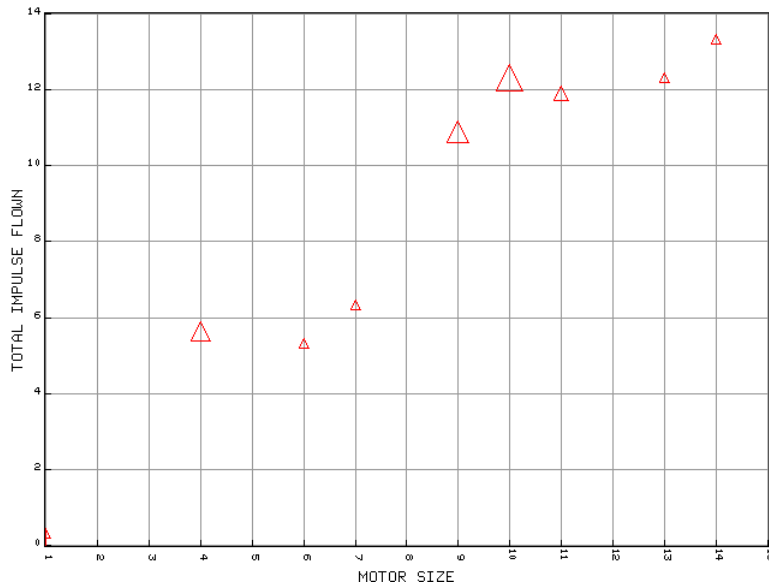


Figure 3.25: Using symbol size to depict extra information

The next example demonstrates how to use the polynomial regression to fit a curve to your data and then display the results. We first generated 10 data points randomly perturbed for interest. We plot these on a scatter plot with red triangles and then draw a 2nd degree polynomial to fit this data in blue. This is all drawn on an image named 'gr'.

The second task creates another image with just text drawn in it. This displays a heading, the polynomial degree taken from the **POLYNOMIAL\_DEGREE** parameter of the scatter plot. Finally, we use the **INDEXED** function to generate a multi-line string from the vector of parameters created when the last **POLYNOMIAL\_DEGREE** command was executed.

Finally, we combine these two images to make a third with the results shown in Figure 3.26.

```

READCSV
Xvalue, YValue
0.489479,0
0.529269,1.42995
2.23421,4.58672
3.47596,10.2601
4.46933,15.9008
5.08902,25.4647
6.15147,34.6383
6.78428,52.0333
8.03008,60.6
8.98747,84.4402
EOF
IMAGE gr WIDTH=320,HEIGHT=240
SCATTER
COLOR=RED,X=Xvalue,Y=YValue,SYMBOL=TRIANGLE,POLYNOMIAL_COLOR=BLUE,
POLYNOMIAL_DEGREE=2
IMAGE DEGS WIDTH=200,HEIGHT=240
LASTY = 220
DRAW TEXT FONT=F8x13Bold,X=0,Y=LASTY,MESSAGE="Polynomial Curve Fit\n"
DRAW TEXT FONT=F6x10,X=0,Y=LASTY,MESSAGE="Degree: " + POLYNOMIAL_DEGREE
DRAW TEXT FONT=F6x10,X=0,Y=LASTY,MESSAGE="Coefficients:\n" + indexed(POLYNOMIAL_COEFFI
IMAGE WIDTH=520,HEIGHT=240
PLACE gr,X=0,Y=0
PLACE degs,X=320,Y=0
GIF "pregr.gif"
DISPLAY FILE="pregr.gif"

```

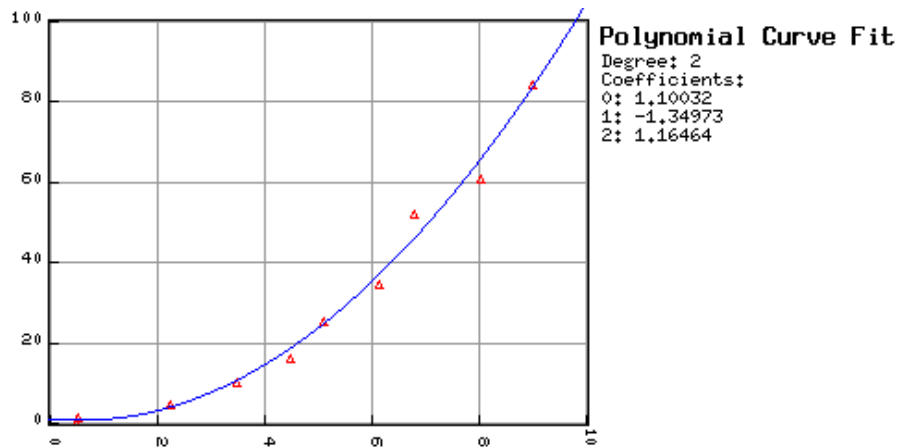


Figure 3.26: Polynomial regression curve for random data.

Figure 3.27 shows what happens if you use too high a polynomial degree for the data. The only change we made to get this graph is the **POLYNOMIAL\_DEGREE** value in the **SCATTER** plot line.

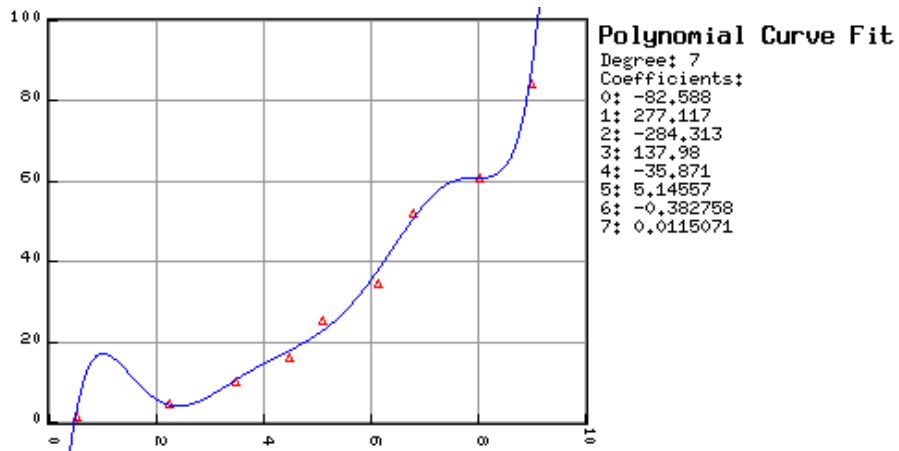


Figure 3.27: Polynomial regression curve for random data, too high a degree.

## 3.29 SHOW

Display a variable's value.

---

**SHOW** < *symbol* >

---

**Description:** Display the symbol's value. For lists, display all top level entities enclosed in parentheses. Same for all types of vectors but show the vector type as well. For image fonts, show the width, height, and zero offset.

**Errors Detected:** None. If you name a variable without a value, its value is NULL.

**Known Deficiencies:** Doesn't even try to do arrays.

**See Also:** SYMBOL on page 110.

**Example:** The following example reads a single column CSV file, computes a histogram, and shows some of the results.

```
// Test show command.
READCSV
value
5
5
...many more values ...

10
13
13
EOF
SHOW value
BUCKETS val TRUNCATE, DATA=value
SHOW val_x
SHOW FINEDOTDASH
SHOW VERSION
SHOW F6x10
```

This results in the following output:

```
2: READCSV
  File test18.graph had 47 entries
52: SHOW value
  value = ( 5
  5
...many more values ...
```

```
13
13
)
53: BUCKETS val TRUNCATE, DATA=value
    11 buckets created
    Finished buckets
54: SHOW val_x
    val_x = #vectorFloat[
    0: 5.000000
    1: 6.000000
    2: 7.000000
    3: 8.000000
    4: 9.000000
    5: 10.000000
    6: 11.000000
    7: 12.000000
    8: 13.000000
    9: 14.000000
    10: 15.000000
    ]
55: SHOW FINEDOTDASH
    FINEDOTDASH = -151587082
56: SHOW VERSION
    VERSION = "graph V1.01.004"
57: SHOW F6x10
    F6x10 = #imagefont [base=32,width=6,height=10,zero=1]
Toodles!
```

### 3.30 SORT

Sort data.

---

```
SORT [ ASCENDING | DESCENDING ] < sort > (
< symbol > [,]* )
```

---

**Description:** Sort a list or vector into ascending or descending order. Additional data items may also “tag along” with the sort - they follow the original values during the sort.

There are no attribute-value parameters. One of the two keywords **ASCENDING** or **DESCENDING** should appear after the **SORT** command. If it doesn't, then the first symbol is assumed to name the structure to sort into ascending order. One of the keywords indicates what order the values must be sorted into. < *sort* > must be a symbol with a list or vector value. It can be followed by one or more other symbols, separated by commas and enclosed in parentheses. These tag along with the structure being sorted.

**Keywords:** None.

**Errors Detected:** **At line *nn*, missing symbol to sort** There wasn't a symbol with a value to sort. This usually happens when **ASCENDING** or **DESCENDING** is spelt wrong.

**At line *nn*, symbol *sss* has no value** A symbol to sort has no value.

**At line *nn*, sort list is missing** The parser detected something following the symbol to sort but it wasn't a left parenthesis.

**At line *nn*, tag along sort value is not a list** All the structures being sorted must be the same type and have the same size.

**At line *nn*, tag along list isn't the same size as the sorted list** All the structures being sorted must be the same type and have the same size.

**At line *nn*, sorting value is not a number** Only numbers can be sorted.

**Known Deficiencies:**

**Example:** This example reads two data columns and sorts them into descending order based on the value of the first column 'rx'.

```
readcsv
rx,ry
1,34
99,12
33,67
```

66,12

2,997

EOF

sort descending rx (ry)

symbol table

### 3.31 SURFACE

Surface plot of three dimensional data.

---

#### SURFACE

```

AXES_FILL = <integer>,
BOUNDARY,
BOUNDARY_COLOR = <integer>,
COLOR = <integer>,
DECORATION_COLOR = <integer>,
GRID_COLOR = <integer>,
GRID_STYLE = <integer>,
HEIGHT = <array>,
LIGHT_U = <number>,
LIGHT_V = <number>,
LIGHT_W = <number>,
ROTATEY = <number>,
SHADING = <integer>,
SIZE = <integer>,
XOFF = <number>,
YOFF = <number>,
ZOFF = <number>,
... Axes keywords ...

```

---

**Description:** Display a 3D surface in front of some axes. Y is up, Z is front to back, and X is left to right. You can use flat shading with polygon outlines, elevation color shading or diffuse shading with a vector light source.

**Keywords:** In addition to some of the axes keyword-value pairs, the following are supported or required.

**AXES\_FILL** = *<integer>* When present, this keyword sets the area fill color value for the 3 planes defining the X,Y,Z axes. The default global value is taken from **SURFACE\_AXES\_FILL** that defaults to 9, a sort of light grey.

**BOUNDARY** When this keyword is present, the surface's quadrilaterals will be outlined with the **BOUNDARY\_COLOR** or its default **SURFACE\_BOUNDARY\_COLOR**. The default is to not outline the quadrilaterals unless the **SHADING** option is **FLAT**.

**BOUNDARY\_COLOR** = *<integer>* If the **BOUNDARY** option appears, then set the drawing color to this index. If this option doesn't appear use the default value in the global **SURFACE\_BOUNDARY\_COLOR** (typically 0 which is black).



**DECORATION\_COLOR** = *<integer>* Sets the color for the various decorations. This defaults to the global **DECORATION\_COLOR** which is typically 0 (black).

**GRID\_COLOR** = *<integer>* This option sets the grid color which defaults to black taken from the **GRID\_COLOR** global.

**GRID\_STYLE** = *<integer>* Set the grid style which is normally set by the **GRID\_STYLE** that is set to **SOLID**.

**HEIGHT** = *<array>* This required pair presents the surface data to be plotted. The column index (number of lines) becomes the X axis value. The value at each array point becomes the height, and the row becomes the Z axis value. If the key is not present, an error will be signalled.

**LIGHT\_U** = *<number>* Sets the X component of the lighting vector for diffuse shading. The vector must be normalized, i.e. for the 3 components:

$$\sqrt{u^2 + v^2 + w^2} = 1$$

The default value is taken from the **LIGHT\_U** global and for the X component is 0.

**LIGHT\_V** = *<number>* Sets the Y component of the lighting vector for diffuse shading. The vector must be normalized as indicated with **LIGHT\_U**. The default value is taken from the **LIGHT\_V** global and is normally -1 indicating a light pointing straight down.

**LIGHT\_W** = *<number>* Sets the Z component of the lighting vector for diffuse shading. The vector must be normalized as indicated with **LIGHT\_U**. The default value is taken from the **LIGHT\_W** global and is normally 0.

**ROTATEY** = *<number>* Sets the rotation around the Y (vertical) axis in degrees. The default value (-45.0) is found in the global **SURFACE\_ROTATEY** variable. Values should not exceed 0.0 and should not be less than -90.0 or the ordered hidden surface removal won't work.

**SHADING** = *<integer>* This option sets the type of surface shading to use. This integer value is one of the values in Table 3.8 and defaults to that in the **SURFACE\_SHADING** global that is normally set to **FLAT**.

Global	Value	Description
<b>FLAT</b>	0	Flat shading using <b>COLOR</b> value.
<b>ELEVATION</b>	1	Use elevation color shading with <b>SIZE</b> colors starting at <b>COLOR</b> value.
<b>DIFFUSE</b>	2	Use diffuse shading with <b>SIZE</b> colors starting at color index <b>COLOR</b> and a light source with a normalized vector pointing <b>LIGHT_U</b> , <b>LIGHT_V</b> , and <b>LIGHT_W</b> .

Table 3.8: Shading values for different surface displays

**SIZE** = *<integer>* Sets the number of colors to be used for the **ELEVATION** and **DIFFUSE** shading models. The default value is taken from the global **SURFACE\_SHADING\_SIZE** that is normally 16.

**XOFF** = *<number!>* Sets the X axis offset. The default value found in **SURFACE\_XOFF** is 0.0. Increasingly large negative values will shift the display area to the left. Larger positive values will go to the right.

**YOFF** = *<number!>* Sets the Y axis offset. The default value found in **SURFACE\_YOFF** is -.25. Increasingly large negative values will shift the display area down. For Larger positive values it will go up.

**ZOFF** = *<number!>* Sets the Z axis offset. The default value found in **SURFACE\_ZOFF** is -3.0. Increasingly large negative values will shift the display area farther away mitigating some effects of the perspective transformation but will result in smaller images. The default viewing display area lies in the  $-2 \rightarrow -4$  Z area.

**Variables Set:** none

**Errors Detected:** In addition to bad type errors and out of storage errors, the following are signaled.

**Known Deficiencies:** The “hidden surface” removal algorithm is non-existent and relies on us drawing stuff from back to front. There’s no connection between the axis size and good display angles.

**See Also:** **BAR3D** on page 30, **COLORMAP** on page 52.

**Examples:**

The first example demonstrates a flat surface plot of a 400 x 400 cosine surface. The control file resembles:

```
// Flat surface plot.
READBIN cosv = "../regression/surface.bin"
```

```

IMAGE WIDTH=640,HEIGHT=480
SURFACE HEIGHT=cosv,ROTATEY=-25,XOFF=-0.2,SHADING=FLAT
PS "surface_flat.ps"

```

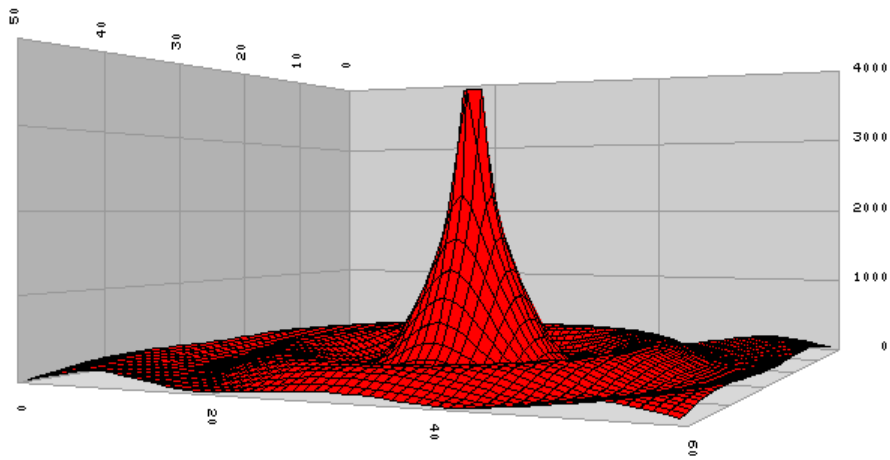


Figure 3.28: Flat shading 3D surface plot

The second example plots the same dataset but with elevation colors and no polygonal boundaries. The results are shown in Figure 3.29.

```

// Surface plot with elevation shading.
READBIN cosv = "../regression/surface.bin"
IMAGE
SURFACE HEIGHT=cosv,ROTATEY=-25,XOFF=-.2,SHADING=ELEVATION,COLOR=ELEVATION_COLORS,SIZE=96
PS "surface_elev.ps"

```

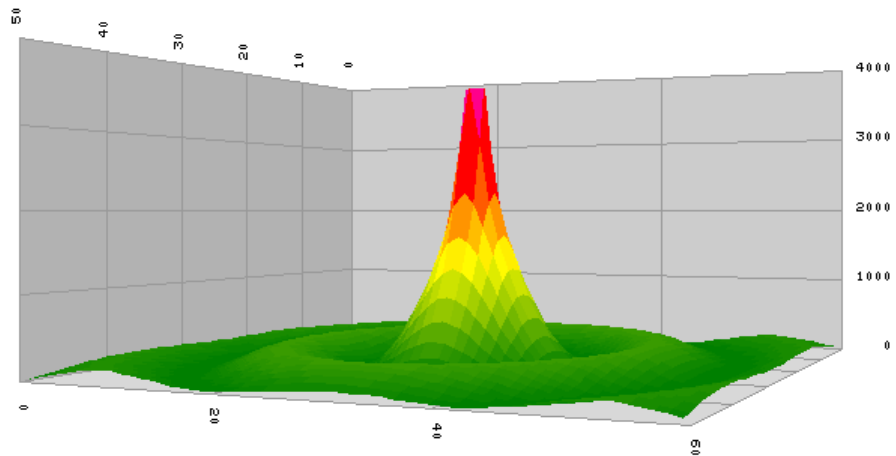


Figure 3.29: Elevation color shading surface plot (flat, no boundaries)

The final example shows a larger data set (640 x 480) of floating-point values displayed with **DIFFUSE** shading. Notice that we had to create the shades of red using the **COLORMAP** command as they are not part of the default colormap.

```
// Diffuse shading surface.
READBIN sinv = "../regression/surfaceb.bin"
IMAGE
COLORMAP SHADE=2,COUNT=32,START=30
SURFACE HEIGHT=sinv,ROTATEY=-25,XOFF=-.2,SHADING=DIFFUSE,COLOR=30,SIZE=32
PS "surface_diffuse.ps"
```

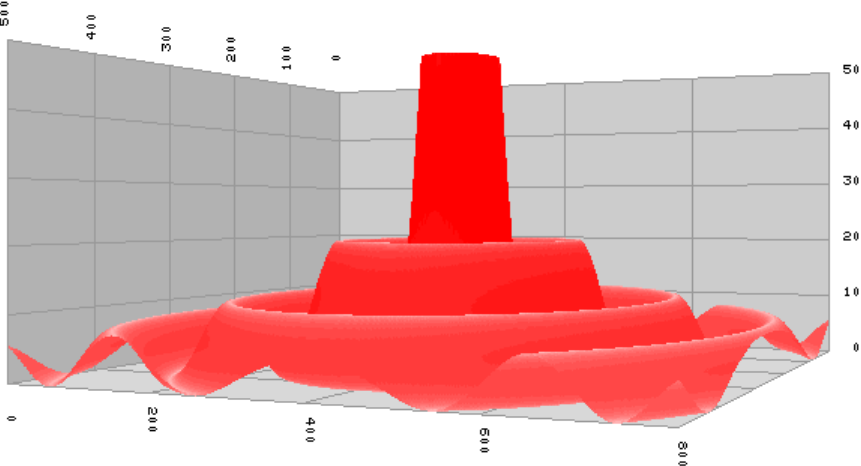


Figure 3.30: Diffuse color shading with light source pointing down

## 3.32 SYMBOL

Display the current symbol table.

---

### SYMBOL TABLE

---

**Description:** Dump the symbol table values to standard output.

**Keywords:** None (TABLE is ignored if present).

**Errors Detected:** None

**Known Deficiencies:** Lists and vectors aren't dumped very well. Symbols longer than 30 characters may be truncated or the columns won't match very well.

**Example:** The following shows the content of the symbol table at startup.

```
1: symbol table
      LINE_STYLE: -1
      SQUARE: 1
      GREEN: 4
      FIRSTFREE: 105
      AXES_FONT: "F5x7"
      AXES_LABEL_FONT: "F8x13"
      CIRCLE: 0
      ASTERISK: 4
      BUCKETSIZE: 1
      MAGENTA: 7
      HEIGHT: 480
      WIDTH: 640
      WHITE: 1
      RED: 2
      BLUE: 6
      SCATTER_COLOR: 0
      GRAPH_BACKGROUND: 1
      ELEVATION: 9
      SCATTER_SIZE: 2
      AXES_VERTICAL_CHARS_SEPARATION: 2
      AXES_HORIZONTAL_CHARS_SEPARATION: 2
      YELLOW: 3
      NOISY: 1
      GREY60: 8
      BLACK: 0
      CYAN: 5
      LINE_COLOR: 0
```

```
PLUS: 3
DECORATION_COLOR: 0
BACKGROUND_COLOR: 1
BAR_COLOR: 2
BAR_SEPARATION: 1
TRIANGLE: 2
VERSION: "graph V1.00.004"
SCATTER_SYMBOL: 0
```

### 3.33 TRANSFORM

Select transform for graphing.

---

```

TRANSFORMX [ NONE | LOG2 | LOG | LOG10
]
TRANSFORMY [ NONE | LOG2 | LOG | LOG10
]
TRANSFORMZ [ NONE | LOG2 | LOG | LOG10
]
TRANSFORMW [ NONE | LOG2 | LOG | LOG10
]

```

---

**Description:** Set the default transform to use on the axis named (these are 4 different commands). The **NONE** transform is straight Cartesian coordinates, the **LOG2** does  $\log_2$  of its axis, **LOG** is the natural logarithm, and **LOG10** is the  $\log_{10}$  transform. The default on all axes is **NONE**.

For 3D displays, Z is in and out of the page, positive Y up, and positive X to the right. The W transform will be used for additional values associated with arrays.

**Keywords:** None.

**Errors Detected:** At line *nn*, missing transform symbol The transform keyword (one of 4 above) must be followed by a symbol.

At line *nn*, symbol 'sss' is not a known transform The transform name you gave isn't known to the system.

**Known Deficiencies:** The axes drawing for the transforms isn't very good yet. Not all plot routines that could use the transforms actually do so.

**Example:** The following example reads a file with  $y = x^2$  in it. The first plot does not transform the values and the second uses a log scale for the X axis and a log2 scale for the Y axis (doesn't work under Windows so don't try it).

```

// Testing log graphs.
readcsv "lg2.csv"
image a width=320, height=240
line color=red, X=n, Y=n2, XLABEL="Y=X^2"
image b width=320, height=240
transformy log2
transformx log
line color=red,X=n,Y=n2, XLABEL="Y=X^2",YLABEL="log2(Y)"

```



```
image width=640,height=240
place a,x=0,y=0
place b,x=320,y=0
ps "../manual/transform.ps"
```

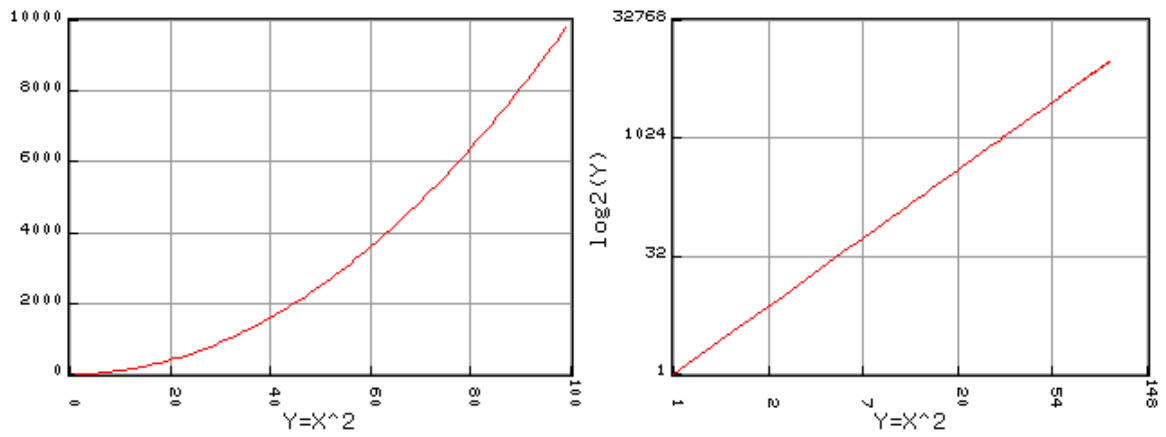


Figure 3.31: Using the LOG transforms for line graphs.



# Chapter 4

## Graphics

Some graphics implementation details need explanation. We examine the default color map and line styles.

### 4.1 Color Maps

The system generates up to 256 color bit maps. A default color map is created at startup along with global names for some of the values. The default is shown in Figure 4.1. Names and color indices are given next to each color.

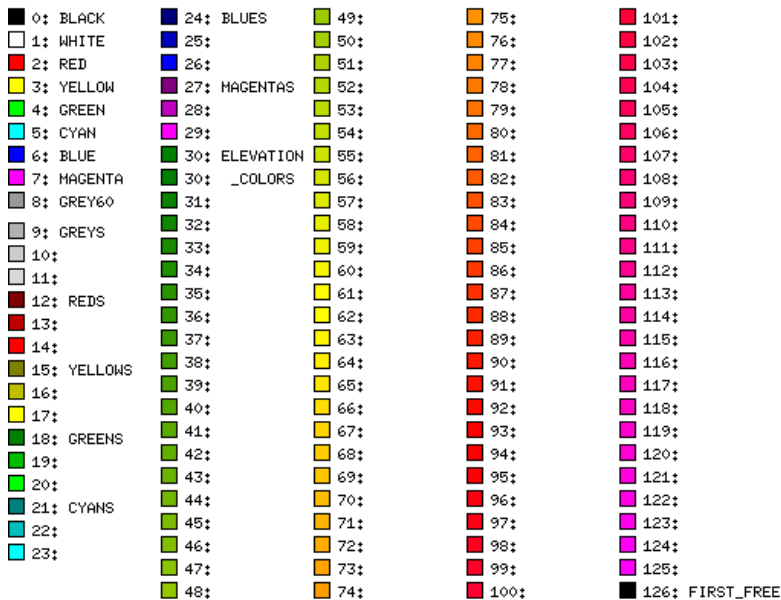


Figure 4.1: Default color map

You are free to redefine any of these colors as you see fit. You can define an additional 130 colors starting at **FIRST\_FREE**.

## 4.2 Line Styles

Some of the graphics commands associated with axes and background grids allow you to specify how to draw lines associated with their activities. A style value is a 32 bit integer that is used to draw 32 bits of a straight line. A 1 bit in the integer causes a dot to be draw, a 0 not. Figure 4.2 shows some common line style values and the values associated with them.

The global variables **SOLID**, **FINEDOT**, **COARSEDOT**, **FINEDASH**, **COARSEDASH**, **FINEDOTDASH**, and **COARSEDOTDASH** have these values preassigned. You are of course free to invent your own values - the last line is a random number with red tic marks indicating every 32nd bit where the pattern repeats.

The following code was used to generate Figure 4.2.

```
image width=640,height=420
TEXT_FONT = F8x13Bold
DRAW TEXT X=0,Y=400,MSG="0xFFFFFFFF"
DRAW LINE 100,400,500,400, STYLE = SOLID
DRAW TEXT X=510,Y=400, MSG="SOLID"

DRAW TEXT X=0,Y=375, MSG="0xAAAAAAAA"
DRAW LINE 100,375,500,375, STYLE = FINEDOT
DRAW TEXT X=510,Y=375, MSG="FINEDOT"

DRAW TEXT X=0,Y=350, MSG="0xCCCCCCCC"
DRAW LINE 100,350,500,350, STYLE = FINEDASH
DRAW TEXT X=510,Y=350, MSG="FINEDASH"

DRAW TEXT X=0,Y=325, MSG="0xEEEEEEEE"
DRAW LINE 100,325,600,325, STYLE=0xEEEEEEEE

DRAW TEXT X=0,Y=300, MSG="0x11111111"
DRAW LINE 100,300,500,300, STYLE = COARSEDOT
DRAW TEXT X=510,Y=300, MSG="COARSEDOT"

DRAW TEXT X=0,Y=275, MSG="0xFF0FFF0"
DRAW LINE 100,275,500,275, STYLE = COARSEDASH
DRAW TEXT X=510,Y=275, MSG="COARSEDASH"

DRAW TEXT X=0,Y=250, MSG="0xFFFFFFFF0"
DRAW LINE 100,250,600,250, STYLE=0xFFFFFFFF0
```

```
DRAW TEXT X=0,Y=225, MSG="0xFFFF0000"  
DRAW LINE 100,225,600,225, STYLE=0xFFFF0000  
  
DRAW TEXT X=0,Y=200, MSG="0xF6F6F6F6"  
DRAW LINE 100,200,500,200, STYLE = FINEDOTDASH  
DRAW TEXT X=510,Y=200, MSG="FINEDOTDASH"  
  
DRAW TEXT X=0,Y=175, MSG="0xFF18FF18"  
DRAW LINE 100,175,500,175, STYLE = COARSEDOTDASH  
DRAW TEXT X=510,Y=175, MSG="COARSEDOTDASH"  
  
DRAW TEXT X=0,Y=150, MSG="0xFFFF03C0"  
DRAW LINE 100,150,600,150, STYLE=0xFFFF03C0  
  
DRAW TEXT X=0,Y=125, MSG="0xFE66FE66"  
DRAW LINE 100,125,600,125, STYLE=0xFE66FE66  
  
DRAW TEXT X=0,Y=100, MSG="0xFF0F0F00"  
DRAW LINE 100,100,600,100, STYLE=0xFF0F0F00  
  
DRAW LINE 100,45,100,55, COLOR=2  
DRAW LINE 132,45,132,55, COLOR=2  
DRAW LINE 164,45,164,55, COLOR=2  
DRAW LINE 196,45,196,55, COLOR=2  
DRAW LINE 228,45,228,55, COLOR=2  
DRAW LINE 260,45,260,55, COLOR=2  
DRAW LINE 292,45,292,55, COLOR=2  
DRAW LINE 324,45,324,55, COLOR=2  
DRAW LINE 356,45,356,55, COLOR=2  
DRAW LINE 388,45,388,55, COLOR=2  
DRAW LINE 420,45,420,55, COLOR=2  
DRAW LINE 452,45,452,55, COLOR=2  
DRAW LINE 484,45,484,55, COLOR=2  
DRAW LINE 516,45,516,55, COLOR=2  
DRAW LINE 548,45,548,55, COLOR=2  
DRAW LINE 580,45,580,55, COLOR=2  
  
DRAW TEXT X=0,Y=50, MSG="0xFE73EDD"  
DRAW LINE 100,50,600,50, STYLE=0xFE73EDD  
  
ps "linestyle.ps"
```















<b>0xFFFFFFFF</b>		<b>SOLID</b>
<b>0xAAAAAAAA</b>		<b>FINEDOT</b>
<b>0xCCCCCCCC</b>		<b>FINEDASH</b>
<b>0xEEEEEEEE</b>		
<b>0x11111111</b>		<b>COARSEDOT</b>
<b>0xFFF0FFF0</b>		<b>COARSEDASH</b>
<b>0xFFFFFFFF</b>		
<b>0xFFFF0000</b>		
<b>0xF6F6F6F6</b>		<b>FINEDOTDASH</b>
<b>0xFF18FF18</b>		<b>COARSEDOTDASH</b>
<b>0xFFFF03C0</b>		
<b>0xFE66FE66</b>		
<b>0xFF0F0F00</b>		
<b>0xFE73EDD</b>		

Figure 4.2: Different linestyles and their STYLE values

# Index

- LaTeXutilization, 14
- ABS function, 18
- addition, 16
- ALTERNATE keyword, 54
- ANGLE keyword, 68, 81
- ARGV0 global, 21
- ARGV1 global, 21
- ARGV*n* globals, 12, 21
- ARROW\_COLOR global, 62
- ARROW\_LENGTH
  - global, 62
- ARROW\_STYLE global, 62
- ARROW\_WIDTH
  - global, 62
- ASCENDING sort keyword, 102
- ASTERISK global, 94
- AVERAGEDEVIATION function, 18
- Axes options, 23
- AXES\_FILL keyword, 23, 30, 104
- AXES\_FONT
  - global, 23
  - keyword, 23
- AXES\_HORIZONTAL\_CHARS\_SEPARATION
  - global, 24
- AXES\_LABEL\_FONT
  - global, 23
  - keyword, 23
- AXES\_VERTICAL\_CHARS\_SEPARATION
  - global, 24
- B global variable, 36, 93
- BACKGROUND keyword, 24
- BACKGROUND\_COLOR
  - global, 72
  - keyword, 73
- BAR
  - command, 26
  - example, 27, 28
- BAR3D command, 30
- BAR3D\_AXES\_FILL global, 23, 30
- BAR3D\_COLOR global, 30
- BAR3D\_PERCENT\_WIDTH global, 30
- BAR3D\_ROTATEY global, 31
- BAR3D\_XOFF global, 31
- BAR3D\_YOFF global, 31
- BAR3D\_ZOFF global, 31
- BAR\_COLOR
  - example, 27, 28
  - global, 26
  - keyword, 26
- BAR\_SEPARATION
  - global, 26
  - keyword, 26
- BASIS keyword, 54
- BEZIER\_COLOR global, 64
- BEZIER\_COUNT global, 64
- BEZIER\_STYLE global, 64
- BLACK color, 50
- BLUE
  - color, 50
  - keyword, 51
- BLUES
  - colors, 30, 50
  - global, 23
- BNP
  - command, 41
  - example, 27
- BOUNDARY keyword, 104
- BOUNDARY\_COLOR keyword, 104
- BOX
  - example, 38
- BOX command, 35
- BOX\_COLOR global, 35

- BOX\_RANGE global, 35
- BOX\_SIZE global variable, 36
- BOX\_WHISKER\_STYLE global, 37
- BUCKET\_SIZE
  - global, 42
  - keyword, 42
- BUCKETS
  - Example, 27
- BUCKETS command, 42
- bytearray, 16
- CIRCLE global, 94
- CIRCLES command, 45
- COARSEDASH global, 116
- COARSEDOT global, 116
- COARSEDOTDASH global, 116
- COLOR
  - command, 50
  - keyword, 30, 35, 45, 62, 64, 66, 68, 72, 76, 93
- color maps, 115
- COLOR1 keyword, 54
- COLOR2 keyword, 55
- COLORMAP command, 52
- COLORMAP\_COUNT global, 52
- COLORS keyword, 45, 55, 81
- CONTOUR command, 53
- CONTOUR\_ALTERNATE global, 54
- CONTOUR\_BASIS global, 54
- CONTOUR\_COLOR1 global, 54
- CONTOUR\_COLOR2 global, 55
- CONTOUR\_COLORS global, 55
- CONTOUR\_DIRECTION global, 55
- CONTOUR\_XBASE global, 56
- CONTOUR\_XINC global, 56
- CONTOUR\_YBASE global, 56
- CONTOUR\_YINC global, 56
- COUNT keyword, 45, 52, 64
- CYAN color, 50
- CYANS
  - colors, 30, 50
  - global, 23
- DATA keyword, 42
- DECORATION\_COLOR
  - global, 104
  - keyword, 104
- DECORATION\_COLOR keyword, 73
- DECORATION\_STYLE keyword, 73
- DESCENDING sort keyword, 102
- DIAMOND global, 94
- DIFFUSE global, 106
- DIRECTION keyword, 55
- DISPLAY
  - example, 28
- DISPLAY command, 61
- DISPLAY\_WITH global, 61
- division, 17
- DRAW ARROW command, 62
- DRAW BEZIER command, 64
- DRAW LINE command, 66
- DRAW TEXT
  - example, 97
  - use, 18
- DRAW TEXT command, 68
- E global, 55
- ELEVATION colors, 50
- ELEVATION global, 106
- ELEVATION\_COLOR keyword, 55
- EOF
  - READCSV terminator, 90
  - READONED terminator, 92
- EQUALCOLOR
  - keyword, 45
- F12x24 font, 68
- F5x7 font, 68
- F6x10 font, 68
- F8x13 font, 68
- F8x13bold font, 68
- F9x15 font, 68
- FILE keyword, 61
- FILLEDCIRCLE global, 94
- FINEDASH global, 116
- FINEDOT global, 116
- FINEDOTDASH global, 116
- FIRSTFREE color, 50
- FLAT global, 106
- float, 15
- floatarray, 16
- floatvector, 16



- FONT keyword, 68, 73
- fonts, 68
- GIF
  - example, 28
- GIF command, 71
- GRAPH\_BACKGROUND global, 24
- GREEN
  - color, 50
  - keyword, 51
- GREENS
  - colors, 30, 50
  - global, 23
- GREY60 color, 50
- GREYS
  - colors, 30, 50
  - global, 23
- grid, 24
- GRID\_COLOR
  - global, 24, 105
  - keyword, 24, 105
- GRID\_STYLE
  - global, 25, 105
  - keyword, 25, 105
- HEIGHT
  - global, 72
  - keyword, 72, 105
- HIGH keyword, 43
- IMAGE
  - example, 27, 28, 38
- IMAGE command, 72
- INDEXED
  - example, 97
  - function, 18
- intarray, 16
- integer, 15
- intvector, 16
- IQR global, 36
- LABEL
  - keyword, 30, 76, 93
- LASTX global, 63, 64, 66, 69
- LASTY global, 63, 64, 66, 69
- LEGEND command, 73
- LEGEND\_BACKGROUND\_COLOR global, 73
- LEGEND\_DECORATION\_COLOR global, 73
- LEGEND\_DECORATION\_STYLE global, 73
- LEGEND\_FONT global, 73
- LENGTH keyword, 62
- LIGHT\_U keyword, global, 105
- LIGHT\_V keyword, global, 105
- LIGHT\_W keyword, global, 105
- LINE command, 76
- LINE keyword, 76
- line styles, 116
- LINE\_COLOR global, 66, 76
- LINE\_STYLE global, 66, 76
- list, 15
- LLSQ
  - keyword, 36, 93
- LLSQ command, 79
- LOG
  - function, 19
  - transform, 112
- LOG10
  - function, 19
  - transform, 112
- LOG2
  - function, 19
  - transform, 112
- LOW keyword, 43
- M global variable, 36, 93
- MAGENTA color, 50
- MAGENTAS
  - colors, 30, 50
  - global, 23
- MARK\_COLOR keyword, 77
- MARKS keyword, 77
- MAX function, 20
- MAXX, 37, 46, 56, 77, 95
- MAXX global, 26
- MEAN function, 19
- MEDIAN function, 20
- MESSAGE keyword, 68, 73
- MIN function, 20
- MINMAX global, 36

- MINX, 37, 46, 56, 77, 95
- MINX global, 26
- multiplication, 17
- N global, 55
- NAMES keyword, 81
- NE global, 55
- NOHORIZONTAL\_GRID option, 25
- NOISY global, 21
- NONE transform, 112
- NOVERTICAL\_GRID option, 25
- NW global, 55
- OUTLINE keyword, 81
- OUTTAKE keyword, 82
- PERCENT keyword, 30
- PERCENTILE global, 36
- PIE command, 81
- PIE\_ANGLE global, 81
- PIE\_COLORS global, 81
- PIE\_OUTTAKE\_PIXELS global, 82
- PIE\_START\_ANGLE global, 82
- PIXELS keyword, 82
- PLACE
  - example, 38, 97
- PLACE command, 85
- PLUS global, 94
- POLYNOMIAL\_COEFFICIENTS
  - example, 97
  - global variable, 36, 94
- POLYNOMIAL\_COLOR
  - example, 97
  - global, 36, 93
  - keyword, 36, 93
- POLYNOMIAL\_DEGREE
  - example, 97
  - global variable, 36
  - keyword, 36, 94
- PREVX global, 64
- PREVY global, 64
- PS command, 87
- RADIUS keyword, 82
- RANGE keyword, 35
- READBIN
  - example, 38
- READBIN command, 88
- READCSV
  - example, 27, 28
- READCSV command, 90
- READONED command, 92
- RED
  - color, 50, 52
  - keyword, 51
- REDS
  - colors, 30, 50
  - global, 23
- ROTATEY keyword, 31, 105
- ROUND
  - function, 20
  - keyword, 43
- S global, 55
- SCALEX, 37, 46, 56, 77, 95
- SCALEX global, 26
- SCALEY, 37, 46, 56, 77, 95
- SCALEY global, 26
- SCATTER command, 93
- SCATTER\_COLOR global, 93
- SCATTER\_SIZE global, 94
- SCATTER\_SYMBOL global, 94
- SE global, 55
- SHADE keyword, 52, 55
- SHADING keyword, 105
- SHOW command, 100
- SIZE
  - function, 20
  - keyword, 36, 94
- SIZE keyword, 77, 106
- SIZES
  - keyword, 45
- SIZES keyword, 94
- SOLID global, 105, 116
- SORT command, 102
- SQUARE global, 94
- STANDARD\_DEVIATION global, 36
- STANDARDDEVIATION function, 20
- START keyword, 52
- START\_ANGLE keyword, 82
- string, 15
- STYLE keyword, 62, 64, 66, 76

- styles, 116
- subtraction, 17
- SUM
  - function, 20
- SURFACE command, 104
- SURFACE\_AXES\_FILL global, 23, 104
- SURFACE\_BOUNDARY\_COLOR global, 104
- SURFACE\_ROTATEY global, 105
- SURFACE\_SHADING global, 105
- SURFACE\_SHADING\_SIZE global, 106
- SURFACE\_XOFF global, 106
- SURFACE\_YOFF global, 106
- SURFACE\_ZOFF global, 106
- SW global, 55
- SYMBOL
  - command, 110
  - keyword, 94
- TEXT\_ANGLE global, 68
- TEXT\_COLOR global, 68
- TEXT\_FONT global, 68
- TRANSFORMW command, 112
- TRANSFORMX command, 112
- TRANSFORMY command, 112
- TRANSFORMZ command, 112
- TRIANGLE global, 94
- TRUNCATE keyword, 43
- VALUES keyword, 82
- VARIANCE function, 21
- VERSION global, 21
- W global, 55
- WHISKER\_STYLE keyword, 37
- WHITE color, 50
- WIDTH
  - global, 72
  - keyword, 62, 72
- WITH keyword, 61
- X keyword, 31, 37, 69, 74, 76, 85, 94
- XBASE keyword, 56
- XHIGH keyword, 24
- XINC keyword, 56
- XINDEX
  - keyword, 46
- XLABEL
  - example, 28
- XLABEL keyword, 24
- XLL, 37, 46, 57, 77, 95
- XLL global, 26, 31
- XLOW keyword, 24
- XNAMES keyword, 46
- XOFF keyword, 31, 106
- XUR, 37, 47, 57, 77, 95
- XUR global, 27, 31
- XVALUES keyword, 46
- Y keyword, 31, 37, 69, 74, 76, 85, 94
- YBASE keyword, 56
- YELLOW color, 50
- YELLOWS
  - colors, 30, 50
  - global, 23
- YHIGH keyword, 24
- YINC keyword, 56
- YINDEX keyword, 46
- YLABEL
  - example, 28
- YLABEL keyword, 24
- YLL, 37, 47, 57, 77, 95
- YLL global, 27, 31
- YLOW keyword, 24
- YNAMES keyword, 46
- YOFF keyword, 31, 106
- YUR, 37, 47, 57, 77, 95
- YUR global, 27, 31
- YVALUES keyword, 46
- ZHIGH keyword, 56
- ZLOW keyword, 56
- ZOFF keyword, 31, 106