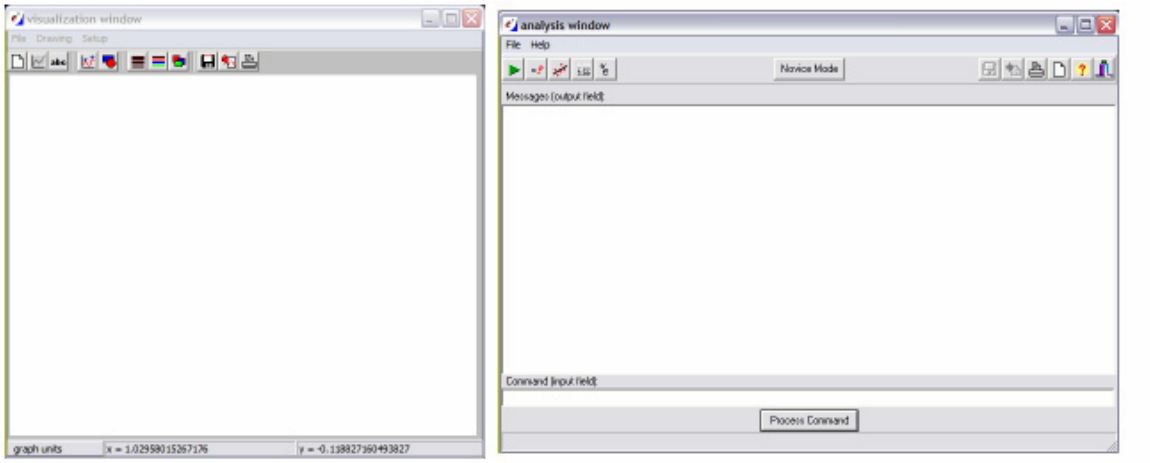
Extrema Tutorial

Getting Started



Introduction

This tutorial is an introduction to using **Extrema** for data analysis and visualization. It is not a comprehensive manual detailing all Extrema features, but rather a concise guide to accomplishing the types of tasks most commonly performed by researchers. It intended to be useful to new users, and for experienced users who would like a quick introduction to features or methods they have not used before.

In most cases, this tutorial instructs by example. Typical data analysis and visualization tasks are described and then performed, both using the GUI and using the scripting language.

The tasks described are generally simple, but practical. Extending them to cases that are more complex is generally straightforward, although you may need to consult the other documentation for further details.

Documentation that is more comprehensive is available in the following references:

Extrema Command Reference: a comprehensive guide to the Extrema command language.

Extrema Online Help: detailed online help is available through the program itself.

In many of the brief procedures described in this guide, the reader will be referred to commands and functions for more information. These commands and functions may be looked up in the above references.

Conventions used in the Tutorials

Examples of messages and prompts written by the program, as well as examples of user typed input are displayed in `typewriter type style.`

Curly brackets, `{}`, enclose parameters that are optional and/or have default values, and indicate that it is not necessary to enter these parameters.

Parentheses, `()`, besides being used in mathematical expressions, also enclose formats.

The backslash, `\`, separates a command from a command qualifier, or a parameter from its qualifier.

Literal quote strings can be delimited by the opening quote, `` ` ``, and the single quote, `'`, or by the single quote at the beginning and the end, or by the double quote, `"`, at the beginning and the end. For example, `` `ABC' ``, `'ABC'`, and `"ABC"` are all valid literal quote strings.

Parentheses, the back slash and quotes **must** be included where indicated.

Extrema is case-insensitive, so input may be provided in upper or lower case, or with a mix of cases. In the examples in this guide, **Extrema** keywords are given in UPPER CASE, while variables and user-defined words are given in lower case, but this is simply for clarity; you do not have to follow this convention. Words that you should replace with your own variable names are given in *italics*.

Installing Extrema

Extrema is distributed as a self-extracting compressed file. You simply need to execute the `extremainstall` program to begin the installation process. After agreeing to the licensing agreement, you then select an installation directory (the default is `C:\Extrema`); and everything else is automatic. The installation program places the **Extrema** icon on the desktop. **Extrema** does not modify the registry, so to uninstall **Extrema** simply delete the files.

Running Extrema

Double-click on the **Extrema** icon to launch the program. By default, the program raises the visualization (graphics) window and the analysis (command input) window. Commands may be typed directly into the analysis window, if you are familiar with the **Extrema** command language. If not, you will probably be more comfortable selecting your actions from the menus and toolbars.

A typical **Extrema** session involves the following steps:

1. load or generate data to work on
2. graph the data, or
3. analyze the data
4. repeat previous two steps until the results are satisfactory
5. customize the presentation of the graph(s)

6. output the graph(s)
7. save data for archival purposes, or for further analysis

Once you become familiar with **Extrema**, or with your particular data analysis and presentation requirements, many of the above steps can be automated. In that case, you can build scripts to automatically perform the routine steps in the above sequence, and possibly the entire sequence itself. **Extrema**'s scripting capabilities include looping and decision-making features, so a fair amount of intelligence can be built into your scripts.

The examples in this guide include instructions for performing operations interactively using the GUI, or using the command language. Command language examples can be used interactively in the command window, or in scripts.

Data Representation

Data is stored internally in **variables**, which have names that you use to reference the data they contain. Except for a few automatically generated variables, these names are chosen by the user. The first character of a variable name **must** be an alphabetic character, that is, `A` to `Z`, and the maximum number of characters in a name is thirty-two (32). Except for these restrictions, variable names can be any combination of: alphabetic characters (`ABC ... XYZ`), digits (`0123456789`), underscore (_), and dollar sign (`$`).

Note	Variable names are case-insensitive, e.g., variable x is the same as X. Function names are reserved names and cannot be used as variable names.

Variables can contain character data or numeric data. Numeric data are always stored as double-precision real values.

Character (or string) variables can be one of the following types:

- **string scalar**: a simple string of text
- **string array**: an array of text strings

Numeric variables can be one of the following types:

- **scalar**: a number
- **vector**: a one-dimensional array of numbers
- **matrix**: a two-dimensional array of numbers
- **tensor**: a three-dimensional array of numbers (*to be implemented*)

The contents of arrays are indexed sequentially, with a starting index of one (1).

Except for physical memory limitations, there is no limit to the number of variables, or to the length of strings, or to the size of arrays.

Addressing parts of arrays

To refer to an entire array, simply use the variable's name.

To select an individual element from the array, provide the index of the element in square brackets:

```
x[8]        ! 8th element of vector x
y[2,6]      ! value from 2nd row, 6th column of matrix y
```

In all of the above cases, you are referring to a single value, i.e., a scalar. You can also specify a range of indices using the colon (`:`) character:

```
x[8:20]     ! 8th through 20th elements of vector x
y[1:10,1]   ! first 10 rows from the first column of y
```

It is also possible to replace any part of an index with a mathematical expression. For example:

```
x[2^3:10*2]             ! 8th through 20th elements of vector x
y[1:sqrt(100),1]        ! first 10 rows from the first column of y
```

Variables can also be used in indices. For example, suppose you have a vector `z` which holds the values `1, 2,…, 10`. The following are then valid:

```
x[z[2]^3:z[#]*2]        ! 8th through 20th elements of vector x
y[z,1]                  ! first 10 rows from the first column of y
```

Such expressions can result in scalars, arrays, vectors, or matrices, depending on the number of dimensions of the result.

The special characters `*` and `#` are also available for use in indices. For example:

```
x[*]            ! all values from vector x
x[#]            ! the last value from vector x
x[#-1]          ! the next to last value from vector x
m[*,*]          ! all rows and all columns of matrix m
m[*,#]          ! all rows and the last column of matrix m
m[*,1:#-1]      ! all rows and all but last column of matrix m
```

Constants

You can type numeric values or constants anywhere a scalar variable or value is expected.

Constant arrays are expressed as a list of values inside square brackets. When typing out vector or matrix values, separate successive indexes with a comma, and successive values within an index with a semicolon.

```
5.03E-8                 ! scalar value
```

```
[1;2;4;8]                    ! vector with 4 values
[1;0;0, 0;1;0, 0;0;1]        ! 3 by 3 identity matrix
```

You can also use the `[start:stop:step]` notation to specify regular sequences of values with which to fill the variable:

```
[0:2*pi:0.1]      ! vector from 0 to 2π in steps of 0.1
[10:-10:-2]       ! descending sequence from 10 to –10 in steps of 2
```

Expressions

Extrema allows you to use mathematical expressions anywhere it would expect a variable or value, provided the expression evaluates to the expected type. Simple expressions involving dimensioned variables generally return a value of the same dimension. Thus, if `x` has `10` values, then the expression `sin(x)+1` also has `10` values. Other examples:

```
m[x,#-2:#]        ! the rows denoted in x, and the last 3 columns of m
x*m[n,*]          ! x times the nth row of m
sin(a+b)          ! the sines of the sums of respective values in a and b
x^2*sin(x)+1      ! a non-linear function of the values in x
```

Note	There is no limit to the length or complexity of a mathematical expression in **Extrema** .

You can also index the results of an expression, e.g.,

```
(SIN(x)+1)[4:8]          ! selects 4th through 8th values of the expression
```

Reading Data From Files

In most cases, your data will be contained in files. You will need to read these files into **Extrema** variables before you can operate on the data. **Extrema** is quite flexible in allowing you to read files of different formats, although in more complex cases you will need to know the details of the file's data format.

Text files

Text files are human-readable, that is, they contain data written in ASCII format, and they can be viewed or edited with a simple text editor such as Notepad ®. Many spreadsheets can export their data into such a format.

If your text file contains data arranged in rows and columns, with columns delimited by commas or white space, then reading the file is simple.

To read each column into its own vector:

```
READ file1.dat x y                ! read 2 columns into vectors x and y
READ file2.dat a a_err b b_err    ! read data and errors into vectors
```

To read all columns into a single matrix, you must also specify the number of rows in the matrix:

```
READ\matrix file3.dat m nrows  ! read file into matrix m
```

There are also options to read matrices in other ways, for example, by specifying the number of columns in the matrix. See the READ command for more information.

Generating Data

Commonly, you will need to create data spontaneously. In simple cases, you can type in the data directly. Usually, however, you will be working with data sizes that make this approach too tedious. There are numerous methods you can use for bulk data generation.

Sequences

Simple sequences can be generated using the [start:stop:step] array notation.

```
pi = ACOS(-1)     ! define scalar pi with value equal to π
X = [0:pi:.01]    ! make a sequence of values from 0 to π in increments of 0.01
```

You can create a regular sequence of values using the GENERATE facility. The generated data can be specified using any of the following methods:

- minimum value, maximum value, number of values
- minimum value, maximum value, step size
- minimum value, step size, number of values

You can also request random values instead of a regular step size.

Functions

By applying an expression to an already-existing variable, you can generate a new variable in which every element of the input variable has been modified by the expression. Capture this data in a new variable by simply setting the new variable to equal the expression:

```
y = 10*SIN(x)          ! x is a vector of values
```

If your source data is a monotonically increasing sequence (see above) that serves as the dependent variable, then you will get a fair representation of the function itself over that range. For instance, to produce data representing the function SIN(x) over the range 0 to 2π:

```
pi = ACOS(-1)
x = [0:2*pi:0.01]
y = SIN(x)
```

Operators

In addition to the simple arithmetic operators:

+	-plus	-	-minus

*	-times	/	-divide
^	-exponentiation	()	-grouping

there are also special vector and matrix operators:

><	- outer product	<>	- inner product
<-	- matrix transpose	>-	- matrix reflect
/|	- vector union	/&	- vector intersection

and a set of Boolean operators that return true (1) or false (0) values:

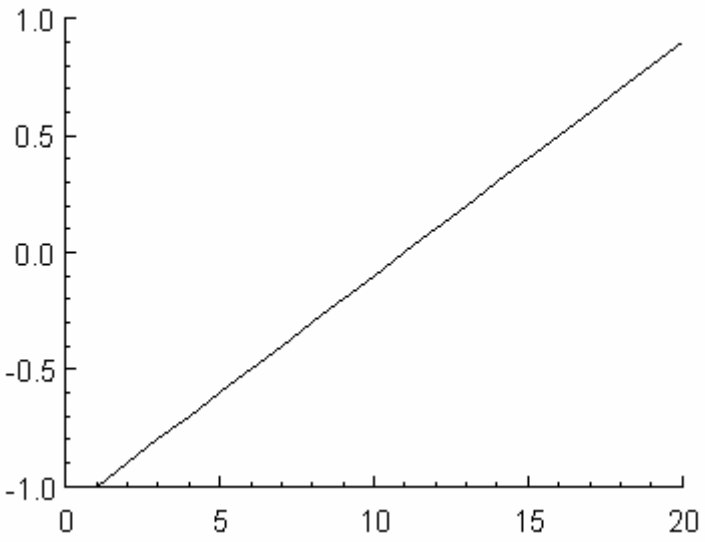
|	- or	||	- exclusive or
&	- and	\	- not
=	- equal to	~=	- not equal to
>	- greater than	<	- less than
>=	- greater than or equal to	<=	- less than or equal to

One-dimensional graphs

One-dimensional graphs are created from a single data vector.

```
GRAPH [-1:1:0.1]
```

Note	If you do not provide an independent variable to graph against, **Extrema** will use the vector index as the independent variable.



Alternatively, you can bin the values in the vector, to turn it into data pairs (i.e., bins and counts). The resulting vectors can be plotted using any of the two-dimensional graph types.

Two-dimensional graphs

Two-dimensional graphs represent data pairs. Typically, you will have two vectors of the same size, which should be plotted against each other in some way.

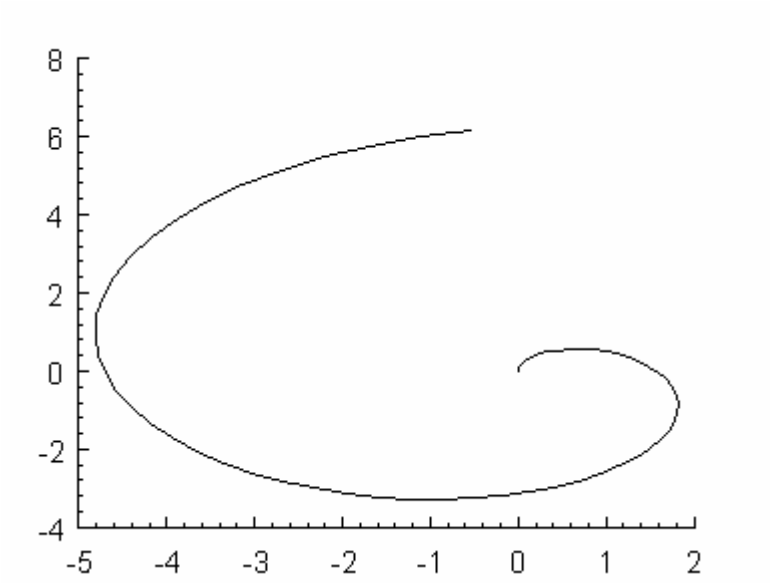
Line graphs

Line graphs connect each subsequent *(x,y)* data point with a line. This presumes that the points are ordered, so that they are connected in sequence.

```
GRAPH x y          ! draw y(x) as a line graph
```

A parametric line graph is also easy to make. If our parametric independent variable is `T`, then we generate `X` and `Y` vectors by passing `T` through appropriate parametric functions:

```
t = [0:2*pi:.1]
x = t*SIN(t)
y = t*COS(t)
GRAPH x y
```



Note	The plotting symbol (characteristic `PLOTSYMBOL`) must be set to 0 to get a line graph. This is the default, so no special action needs to be taken unless the plotting symbol has been otherwise set.

If your data is already in polar coordinates, you can graph it directly using the `\POLAR` option, e.g.:

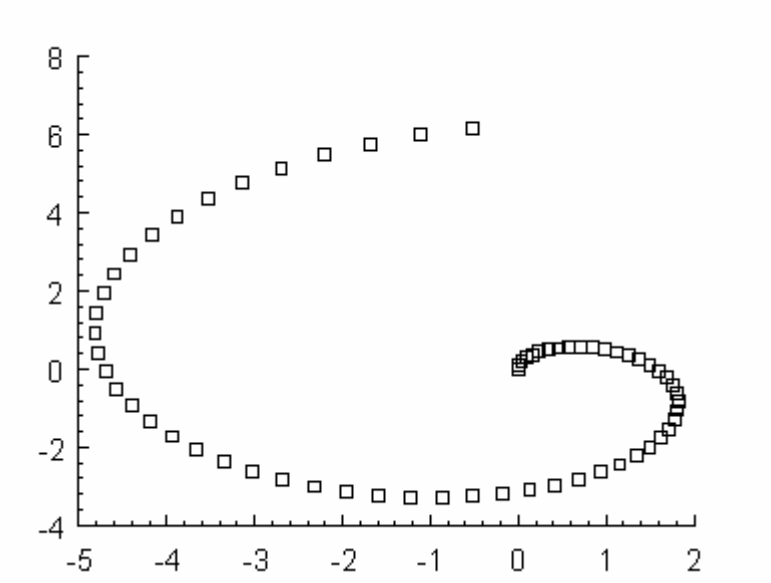
```
GRAPH\POLAR radius_vector angle_vector
```

Note	Angles are presumed to be in degrees.

Scatter plots

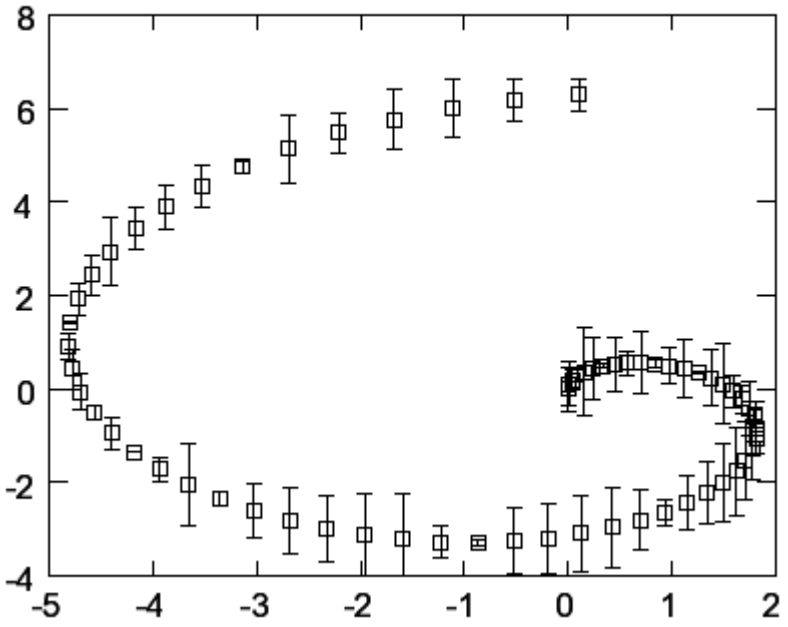
Scatter plots take the corresponding elements of each vector, and plot them as *(x,y)* data points, using whatever plot symbol has been selected. There is no requirement that the vectors be ordered in any particular way. To plot scattered points that are not joined by a line, select a negative symbol type.

```
SET PLOTSYMBOL -1
GRAPH x y
```



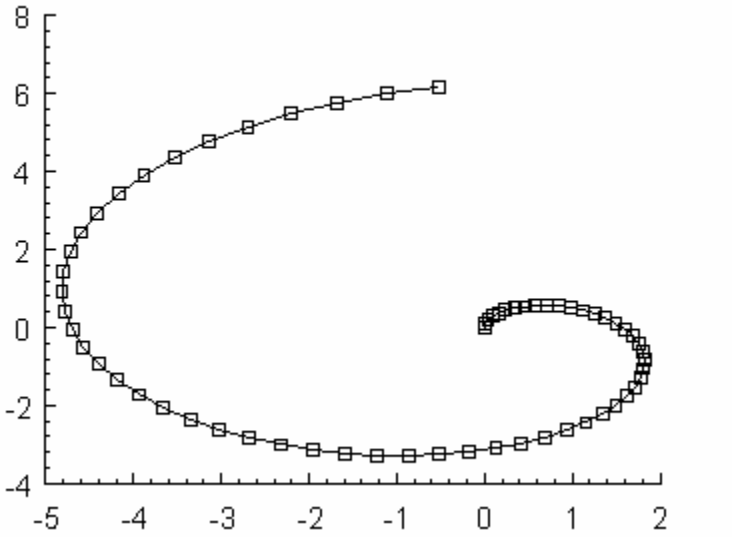
If we have errors in the data also stored in a matching vector, then we can add that information to the plot by specifying the error vector(s):

```
GRAPH x y yerr
```



If your data is ordered, and you would like the data points to be joined with a line, then simply use a positive plotting symbol number:

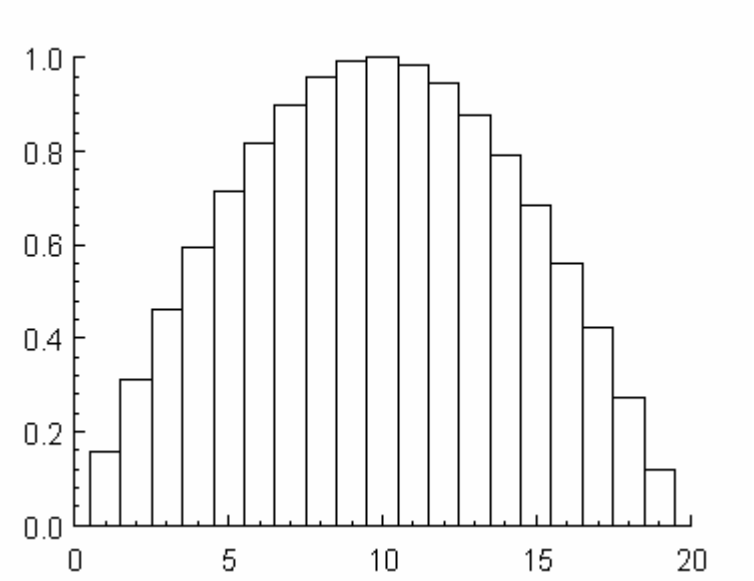
```
SET PLOTSYMBOL 1
GRAPH x y
```



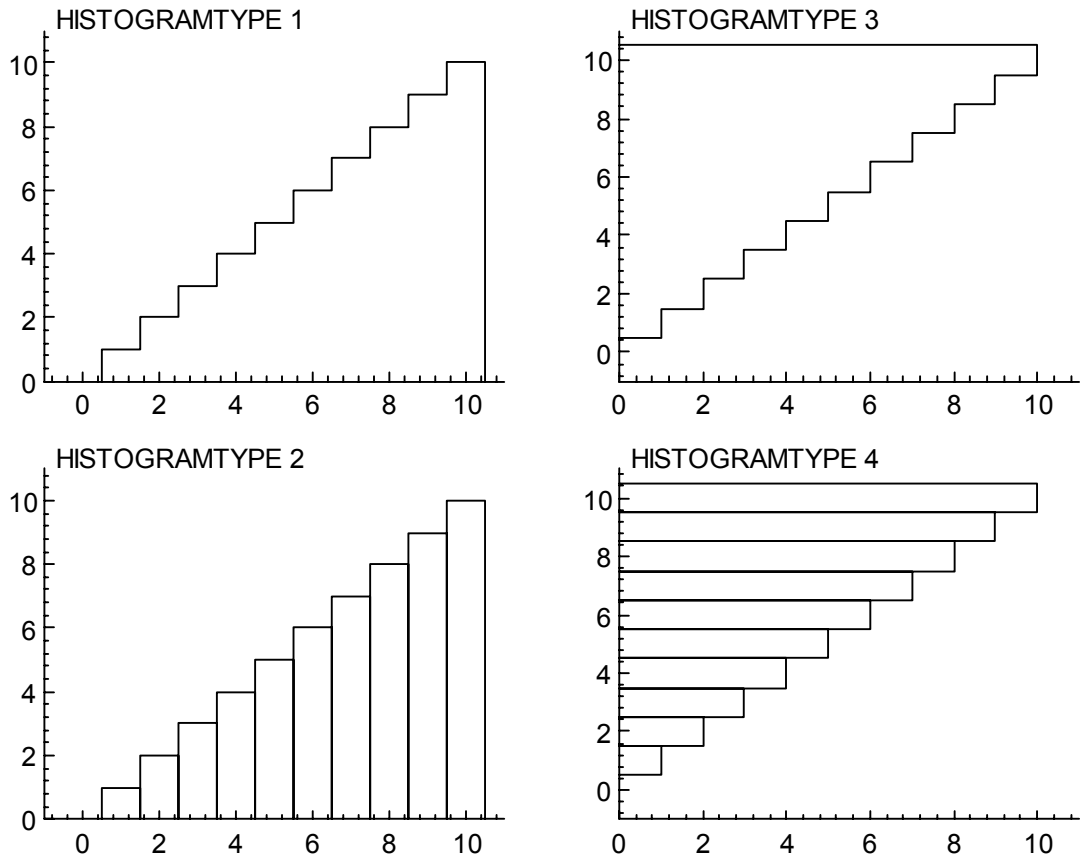
Histograms and bar charts

Histograms (bar charts) with tails going to y=0 are drawn by using the `\HISTOGRAM` qualifier with the `GRAPH` command. x values are assumed to be bins, and y values are assumed to be counts.

```
X = [1:19]
Y = SIN(x/(2*pi))
GRAPH\HISTOGRAM x y
```



The other types of histogram are shown below, each with the appropriate value of HISTOGRAMTYPE.



The above picture was created with the following commands:

```
CLEAR
WINDOW 5
DEFAULTS
SET
 %XNUMBERHEIGHT 6
 %YNUMBERHEIGHT 6
  HISTOGRAMTYPE 1

SCALES -1 11 0 11
GRAPH [1:10]
SET
 %XTEXTLOCATION 12
 %YTEXTLOCATION 90
 TEXTALIGN 1
 TEXTINTERACTIVE 0
 %TEXTHEIGHT 6

TEXT 'HISTOGRAMTYPE 1'
WINDOW\INHERIT 6 5
SET HISTOGRAMTYPE 2
SCALES -1 11 0 11
GRAPH [1:10]
SET TEXTINTERACTIVE 0
TEXT 'HISTOGRAMTYPE 2'
WINDOW\INHERIT 7 5
SET HISTOGRAMTYPE 3
SCALES 0 11 -1 11
GRAPH [1:10]
SET TEXTINTERACTIVE 0
TEXT 'HISTOGRAMTYPE 3'
WINDOW\INHERIT 8 5
SET HISTOGRAMTYPE 4
SCALES 0 11 -1 11
GRAPH [1:10]
SET TEXTINTERACTIVE 0
TEXT 'HISTOGRAMTYPE 4'
```

Multiple plots on the same drawing

Researchers commonly need to combine graphs into the same drawing, plot multiple data sets on the same graph, draw different graphs with a common axis, and so on. There are many ways **Extrema** can be used to get these effects.

Graphics sub-windows

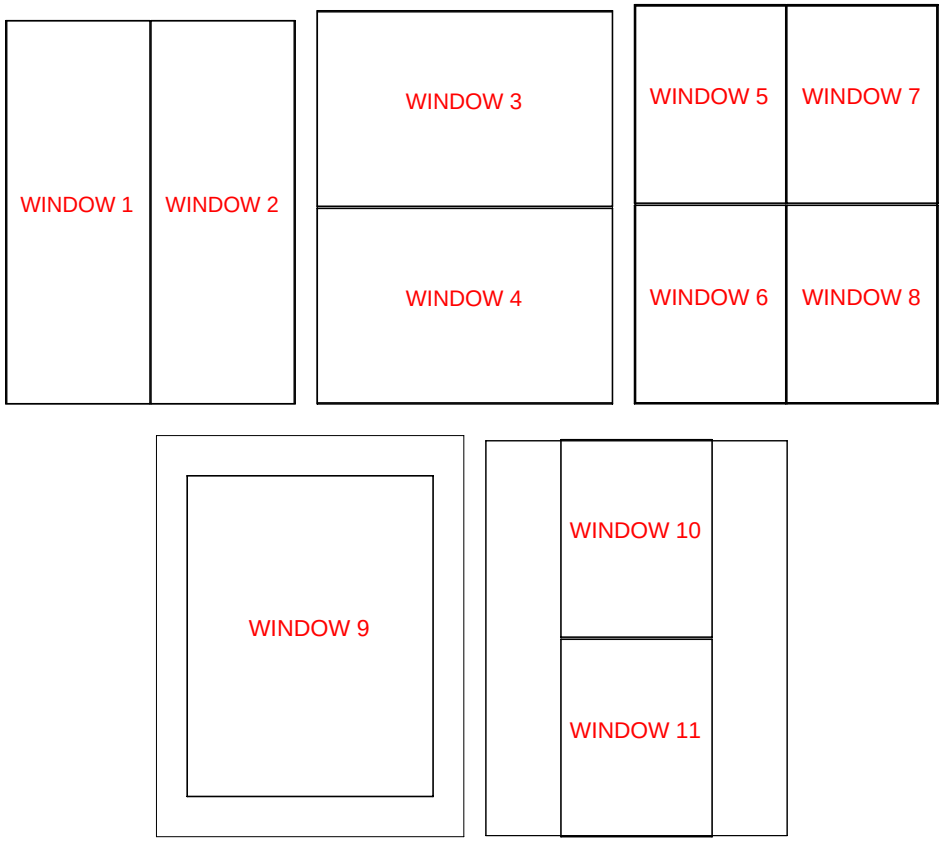
Use the `WINDOW` command to choose and/or define a graphics sub-window. Graphics sub-windows are an easy way to subdivide the graphics output page into rectangular regions, allowing multiple graphs and/or multiple figures and/or multiple text regions. A window is a

subset of the page. A window, other than the default zero level window, has a smaller plotting unit range than the full page.

Note	Commensurateness is never lost in a sub-window.

Pre-defined windows

Some of the initial pre-defined windows in PORTRAIT orientation are displayed below.

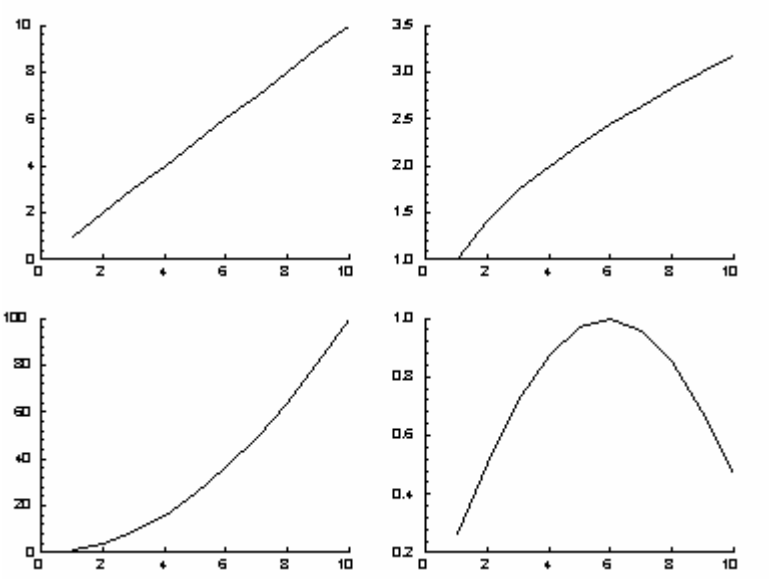


Tile numerous graphs on the same drawing

Extrema divides the drawing area into **windows**, which can be selected by their number to confine a graph to a particular section of the drawing. Window number 0 is the default.

For example, to tile four graphs on the same drawing, simply select windows 5, 6, 7, and 8 in order, and issue an appropriate GRAPH command for each.

```
WINDOW 5
GRAPH x y1
WINDOW 6
GRAPH x y2
WINDOW 7
GRAPH x y3
WINDOW 8
GRAPH x y4
```

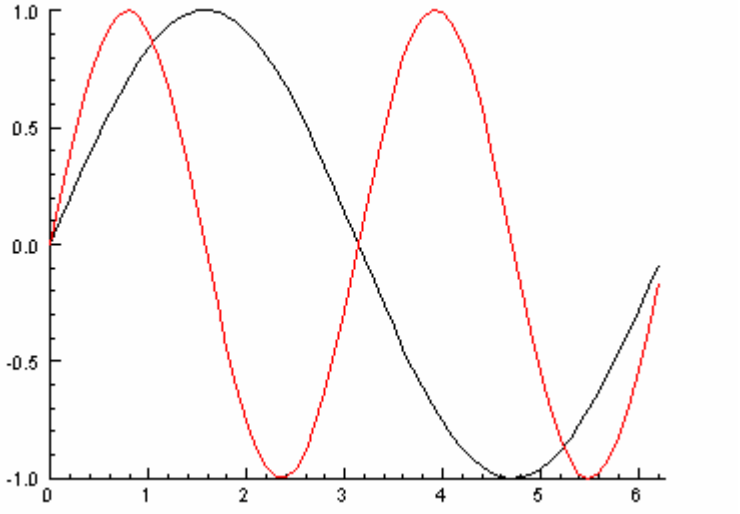


The WINDOW command can also be used to define your own custom set of drawing windows.

Draw two sets of data on the same graph at the same scale

Method 1: Manually set the axis scales to values that are appropriate for both graphs. Then draw your first graph. Without clearing the graph, draw the second graph without axes.

```
SCALE 0 2*pi -1 1          ! sets xmin xmax ymin ymax
GRAPH x y
SET CURVECOLOR RED         ! change color for overlaid curve
GRAPH\OVERLAY x z
```



Method 2: Draw the graph that should be used to autoscale the axes first, then freeze the axes at those values, before drawing the second graph without axes.

```
GRAPH x y
SCALES                  ! freezes the current scale
GRAPH\OVERLAY x z
```

Method 3: Graph either of the two data curves first, then overlay the second data curve. Then use the `REPLOT` command to redraw both curves on a common scale.

```
GRAPH x y
GRAPH\OVERLAY x z
REPLOT
```

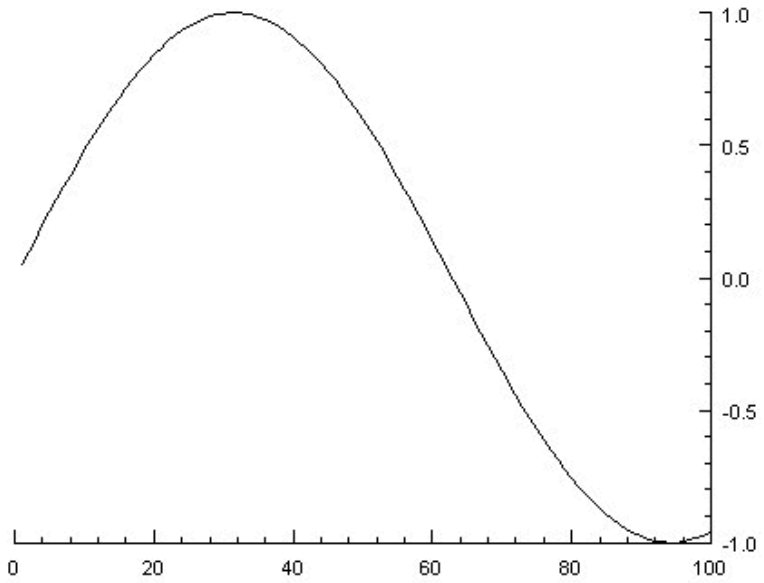
Draw two sets of data on the same graph, but at different Y-scales

Method 1: If you only need a labelled *y*-axis for one data set, the task is easy:

```
GRAPH x y                   ! y-axis is for this graph
GRAPH\OVERLAY x z           ! no y-scale shown for this graph
```

Method 2: By default, the y-axis is drawn at the left hand end of the *x*-axis. The `GRAPH\YONRIGHT` command draws the *y*-axis on the right. For example, the following commands produce the figure below.

```
X = [1:100]
GRAPH\YONRIGHT X SIN(X/20)
```

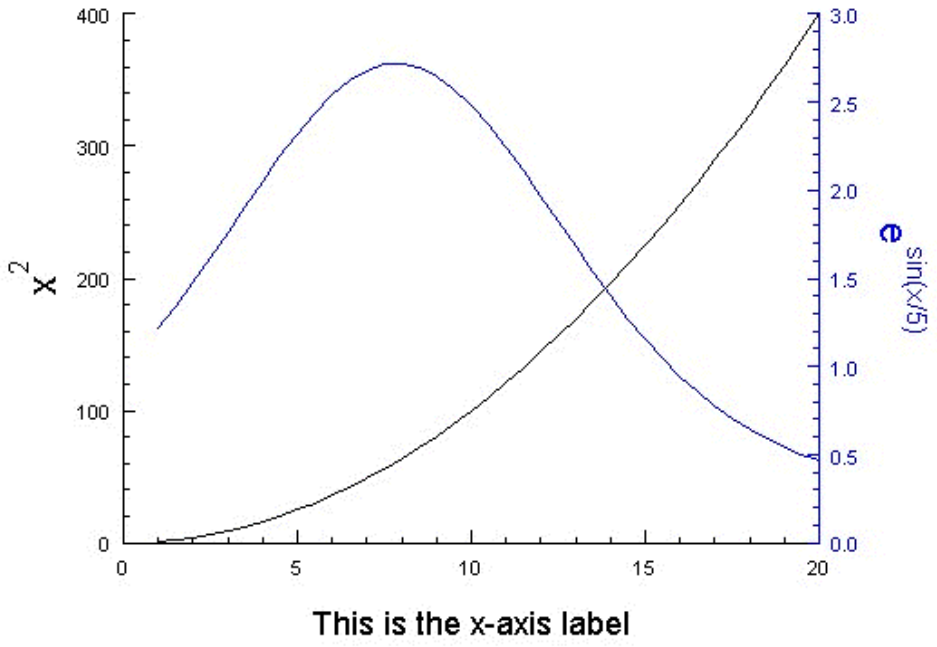


If you need a labelled *y*-axis for both data sets, just graph the first data curve with the *y*-axis on the left (the default), and then graph the second data curve with the *y*-axis on the right. You might want to change the color for the second *y*-axis and curve to distinguish it from the first. For example:

```
x=[1:20:.5]
y1=x^2
y2=EXP(SIN(x/5))
SET
 XLABEL 'This is the x-axis label'
 XLABELON 1
%XLABELHEIGHT 5
 YLABEL 'x<^>2'
 YLABELON 1
%YLABELHEIGHT 5

GRAPH x y1
SET
 YAXISCOLOR blue
 YNUMBERSCOLOR blue
 CURVECOLOR blue
 XAXIS 0
 YLABELCOLOR blue
 YLABEL 'e<^>sin(x/5)'
```

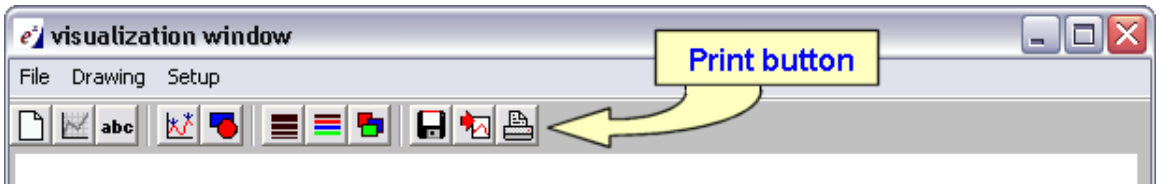
```
GRAPH\YONRIGHT x y2
```



Note	The `GRAPH\XONTOP` command draws the *x*-axis on the top

Printing graphics

Printing your graphs is very easy; simply select `Print` and proceed as you would for any other Windows printing job.



Exporting graphics

Researchers commonly need to include their graphs in other documents, such as research papers, written reports, or web pages. For these purposes, **Extrema** can export to several industry-standard graphics formats: PostScript (`EPS`), Portable Network Graphics (`PNG`), and Joint Photographic Experts Group (`JPEG`). The `HARDCOPY` command is used for saving the

graphics to a file in one of the supported formats. Encapsulated PostScript is the default format, if no qualifier is entered with the HARDCOPY command.

PostScript & EPS

PostScript (EPS) is the industry standard for printed documents; it provides excellent, publication-quality output that is completely scalable, and is compatible with documents conforming to the Portable Document Format (PDF) or to the TeX and LaTeX systems that are common in scientific publishing.

PNG

Portable Network Graphic (PNG) is a bitmap image format that is supported by most major web browsers, including Explorer and Netscape. As a bitmap format, it is inferior for regular publication purposes, but it is convenient for in-lined image display in web pages. It gives high rates of compression for conventional drawings and plots, and is the recommended graphics format for most drawings.

JPEG

Joint Photographic Experts Group (JPEG) images are also stored in a bitmap format, and suffer from all the drawbacks of PNG images. They are also optimized for displaying photographic images, and do not generally give good compression for conventional drawings and plots. Some complex drawings that involve smoothly varying gradients of tone or color may benefit from being exported to JPEG format, however.