RLPlot

From Data to Graphs

Version 2.2

Reinhard Lackner, Inst. f. Zoologie, University of Innsbruck

document last updated: 7-June-2026

Contents

Introduction

RLPlot is a tool to create graphs from data. This program was developed in a natural sciences environment. Imagine you have a piece of paper, a ruler, and a pocket calculator and you wish to create a graph using an ink pen. That's how it started off leading to the development of RLPlot.

RLPlot starts with a spreadsheet. This spreadsheet is intended to accept all the variables needed to create Graphs. RLPlot is not rivaling with other spreadsheet programs, RLPlot is a graphics program. However, you may find it useful to use the spreadsheet as a replacement for a pocket calculator. It may also serve as a replacement for statistical tables as all statistical functions and distributions are available.

There is a close relation between graphs and statistics. RLPlot offers both, graphs and simple statistics as needed especially for explorative approaches. Graphs are vector graphics for highest possible resolution, easy modification by other graphic programs to give ready to publish quality graphs. Please bear in mind that editors and publishers are not happy if statistical items of the graphs are a mere decoration.

Compatibility with prior versions: RLPlot is upward compatible. Any files written with an older version of RLPlot, from any platform, can be opened and processed with the most recent versions.

If you want to create an impressive chart based on few data RLPlot is probably the wrong choice for you. You should consider a simple table in your presentation containing the few data. This is probably not so impressive but more honest.

RLPlot was written in the hope that it is useful and easy to use. I hope you like it and I wish all much success.

Installing RLPlot

RLPlot is usually distributed as executable file. Thus, no specific installation is required. Just download the file, store it wherever you like, and execute it by double clicking the file.

Removing RLPlot from your System

RLPlot stores some user specific data in a file called RLPlot or .RLPlot. The actual position of this file can be found in RLPlot selecting from the menu ?/Configure on the Dialogs tab. Newer installations of Windows tend to offer a hidden directory for application specific data. RLPlot uses the path from the registry `at HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders` to read/write this file. It is a text file which can be opened or modified by a text editor. It can be safely removed. If you leave it, subsequent installations of RLPlot will use the information therein. If you delete the RLPlot executable and all shortcuts to it all of the program is removed.

RLPlot uses two types of files namely those with an extension *.rlp (graphs) and with an extension *.rlw (RLPlot workbook; a spreadsheet possibly containing graphs). To my knowledge no other program than RLPlot can operate can operate on these files. When RLPlot writes a file and the file already exist it is renamed with an extension of the style *.001, *.002 … *.003. They can be opened by RLPlot probably being renamed with an *.rlp or *.rlw extension. All RLPlot files are text files and can be examined with a text editor. If you are sure that you do not need the information in these files they can be safely deleted. RLPlot is upward compatible. So even very old files can be opened with newer versions of RLPlot. Drag and drop of files to the spreadsheet window usually works, also for backup files.

The RLPlot Spreadsheet

When RLPlot is executed, it will display a worksheet after displaying the welcome banner. This spreadsheet will contain the data from which you wish to create a graph.

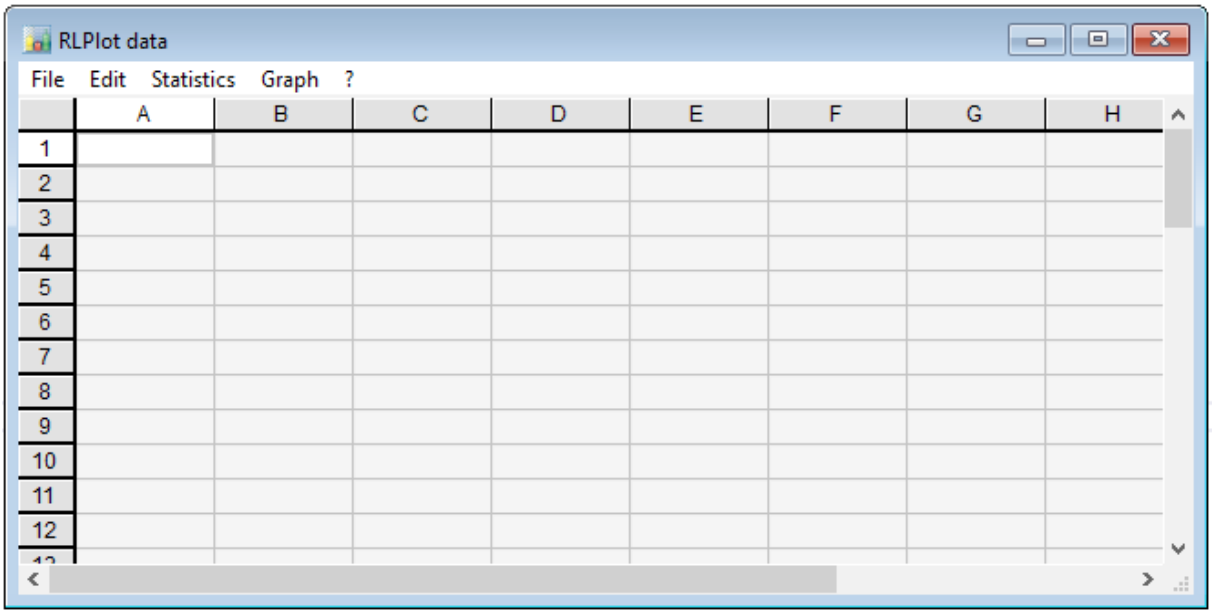


Fig. 1: The empty spreadsheet as displayed upon start of RLPlot.

Selecting a Range

The data which will be used in subsequent operations can be selected with the mouse or with the arrow keys | <PgUp> | <PgDown> Keys. Pressing the <Shift> or <Ctrl> key together with the mouse allows the selection of multiple ranges.

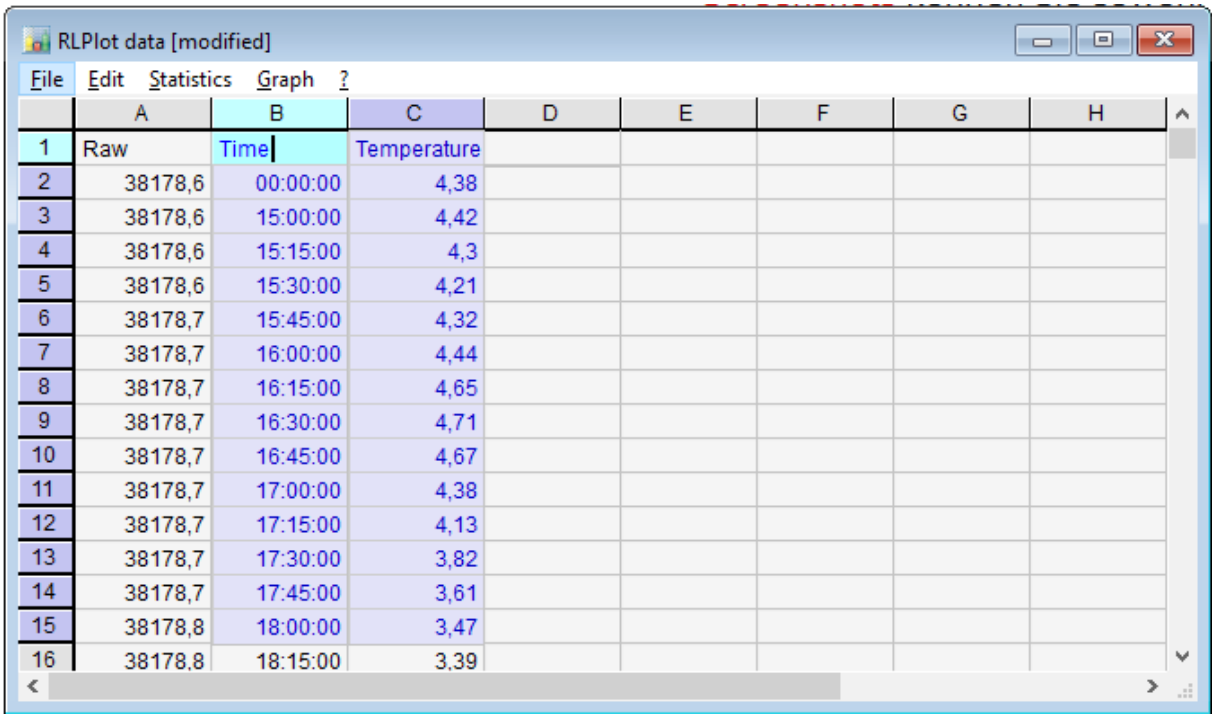


Fig. 2: The RLPlot spreadsheet wit some sample data. Note the selected range including column headers.

In the above image the range b1:c15 is selected. If you create a graph this selection is taken as variables for the graph. Note that the range includes column headers. This field is taken as a name for the column and used for labelling the graphs and reports.

Ranges can also be used to copy from and paste to the spreadsheet. You may prefer to copy data from your favorite spreadsheet program to the RLPlot spreadsheet and vice versa. For most users this may be superior to entering data via the keyboard.

Keyboard Usage

<Arrow keys>	Move the text cursor within the spreadsheet.
<PgUp, PgDown>	Scroll Spreadsheet up or down.
<Tab, Shift Tab>	Move to next column left or right.
<Esc>	Clear marks, empty clipboard …
<Ctrl + c>	Copy data
<Ctrl + v>	Paste data
<Ctr + x>	Cut data
<Del>	Delete the character right of the cursor or the selected range.
<Backspace>	Delete the character left of the cursor or the selected range.
<Ctrl + z>	Undo last actions. There is a cache and you can undo several steps until the cache is empty.
<Pos1><Pos Last>	Position the cursor at the beginning or end of the text.

Most other keys input text or values into the spreadsheet.

Getting started: Your first Graph

Let's start with the spreadsheet as displayed in Fig. 2, including the selected range. We will turn this data into a Graph. Now, from the menu select Graph/Create Graph. A 'Create Graph' dialog will appear.

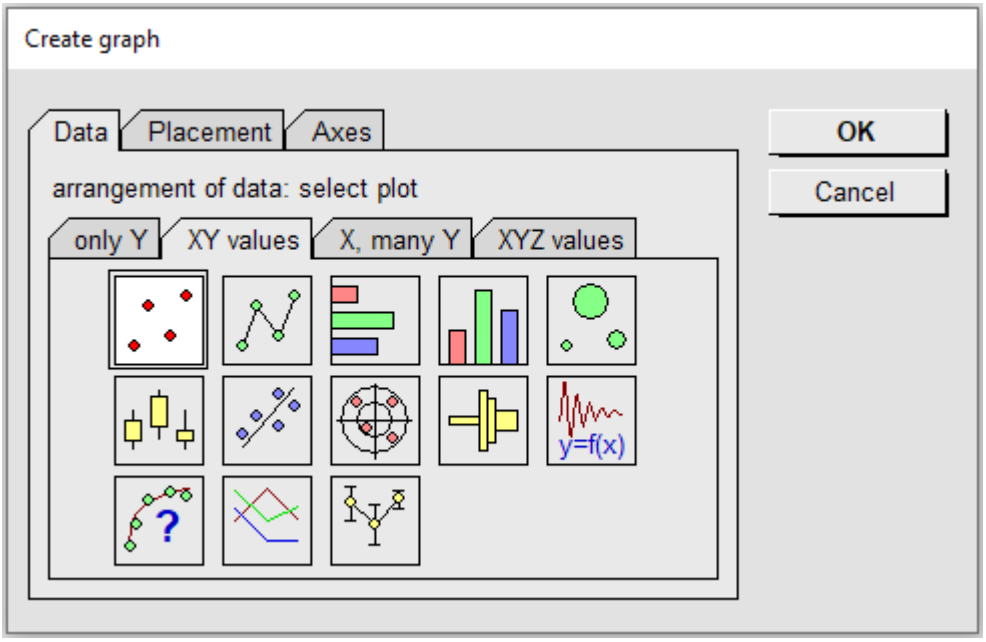


Fig. 3: The Create Graph dialog.

From the icons select the  icon if not already active. Click the [OK] button to continue. Clicking an active icon is equivalent to clicking [OK]. Also double clicking an icon is understood as select and OK. This will open the XY Plot properties dialog.

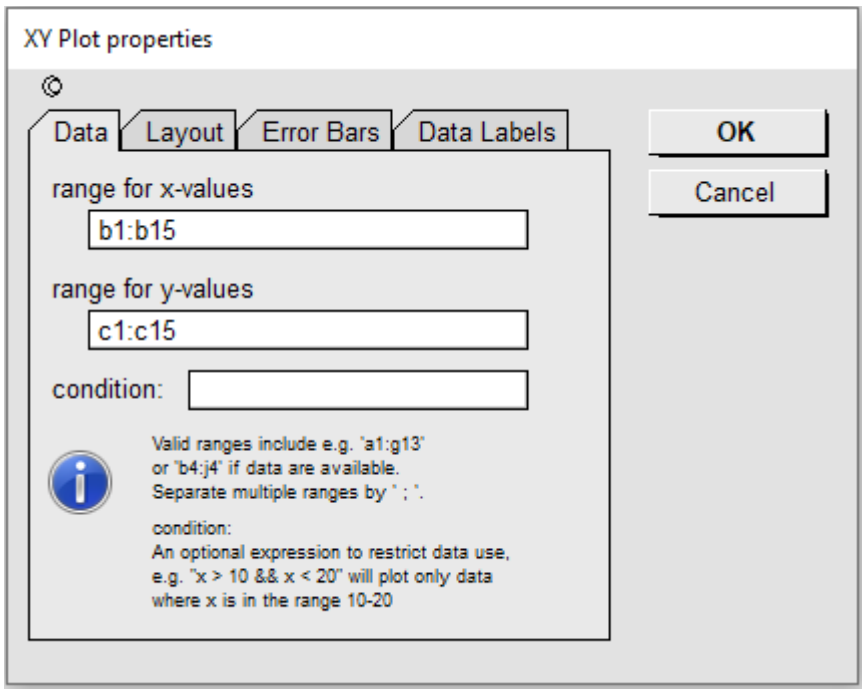


Fig. 4: Dialog to create an XY-plot or scatter plot.

The ranges selected in the spreadsheet are now copied to the range for x-values and range for y-values fields. The ranges for variables may also entered manually into these fields. At this point you probably don't need any further adjustments: just click the [OK] button. The dialog reminds you that you should have visited the Layout tab because the default settings probably will not fit your desires. To continue, just press the [OK] button again.

A graph is created and displayed. The whole graph is like a graphical menu: double click any item to get another dialog where the settings of this item can be seen and modified. Alternatively, you may select an item and pressing [Return] will give the same dialog. Selected items can be deleted by pressing the [Del] button.

In most cases RLPlot accepts text values as variables. They are assigned ordinal values starting with one up to the last in the order of their appearance (1 .. n).

Zoom in and zoom out

Create a rectangular mark with the mouse and select <zoom in> from the menu or press <Ctrl>+ +. The area of the mark will be displayed as a magnified view to the graph. This piece of the graph maintains the original aspect ratio. Selecting <zoom out> or pressing <Ctrl>+ - will revert to the previous view. Several levels of zoom in and zoom out are supported. If no rectangular area is present before zooming in a magnifying glass cursor is displayed allowing you to select a rectangular region to be magnified. The <Ctrl>+ + command an <Ctrl>+ - commands are equivalent to the menu zoom in and zoom out commands

Copy and paste

RLPlot supports the clipboard to exchange graphical objects. Usually, the whole graph is copied. Select a graph and select <Copy> from the menu. Also <Ctrl>+c is understood. Pasting the contents of the clipboard to a word processor will usually insert a 300dpi bitmap into your text file. Some vector drawing programs will understand the SVG contents of the clipboard. In general, the SVG-format is recommended for any export from RLPlot. This format contains a maximum of information about a graph.

Pasting a graph into another graph may be used to create an inset into a graph. Pasting the graph to the spreadsheet will create a copy of the graph as a new graph. After issuing the paste command a small clipboard cursor allows to select a rectangle where the copied graph will be fitted in. If no rectangle is selected the graph is inserted with its original size.

RLPlot supports several different clipboard formats. Whenever you select copy RLPlot does not know which clipboard format will be preferred by the accepting program. When copying from RLPlot to RLPlot the preferred format is a specific RLPlot OEM format. A word processor might prefer text or a bitmap, a spreadsheet program might prefer the spreadsheet format SYLK, a graphic application might like the bitmap format, and some drawing programs even understand the SVG format. Thus, a negotiation may start and RLPlot creates the respective clipboard content on demand. Clipboard contents are volatile and intended for immediate copy/paste use.

Exporting graphs

If you need to polish your graphs with some different software you will need to export graphs from RLPlot. RLPlot supports bitmap and vector formats for export. If possible, the suggested format for

export is SVG. This is the most powerful vector format supported by RLPlot. For example, with this format vectorized fill patterns are grouped and may be handled as such. You may give your Graphs a final touch with programs like Inkscape, Corel Draw, or any other vector drawing program.

Using graphs as templates

Assume you have a bunch of data all with an equal layout and you want to create equal graphs for each data set. In this case create a graph from one data set. Open a new instance of RLPlot with another set of data. Now copy the graph of the first instance to the spreadsheet of the second instance. This copy of the graph does not update the values. Now select Update values from the menu and the Graph will be updated with the values of the new instance.

Mouse Usage

<Left button single click> Select object

<Left button double click> Select object and open the properties dialog for this object

<Right button single click> This is equivalent to <Left button double click>.

<Left button press and move> This marks a rectangular range as needed for zoom in operations. Some objects are marked as moveable. If such an object is selected subsequent press and move actions will reposition the object. Some objects have drag handles. They may be used to resize, move, or rotate the object, plot, or graph.

Keyboard Usage

<Arrow keys> Most 2D graphs will scroll up/down/left/right if the graph is selected. In 3D graphs these keys rotate the image. Some objects are marked as moveable. If one of these objects is selected the arrow keys move these objects accordingly.
<Pg Up><Pg Down> Scroll the image up or down.
<Esc> Clear marks, empty clipboard …
<Ctrl + c> Copy
<Ctrl + v> Paste
<Ctrl + z> Undo the last operation
<Tab><Shift Tab> If an object like a symbol or bar out of list is selected, these keys select the next or previous object of the list. If no object is selected the graph is scrolled left or right.

<r><l> Specific for 3D graphs: Rotate the graph left or right.

<Enter> Display the properties of a selected object

About Dialogs

RLPlot has about 100 different dialog windows used to create graphs or to modify or inspect details of a graph. We have seen a typical dialog before. Let's go to some details of this dialog.

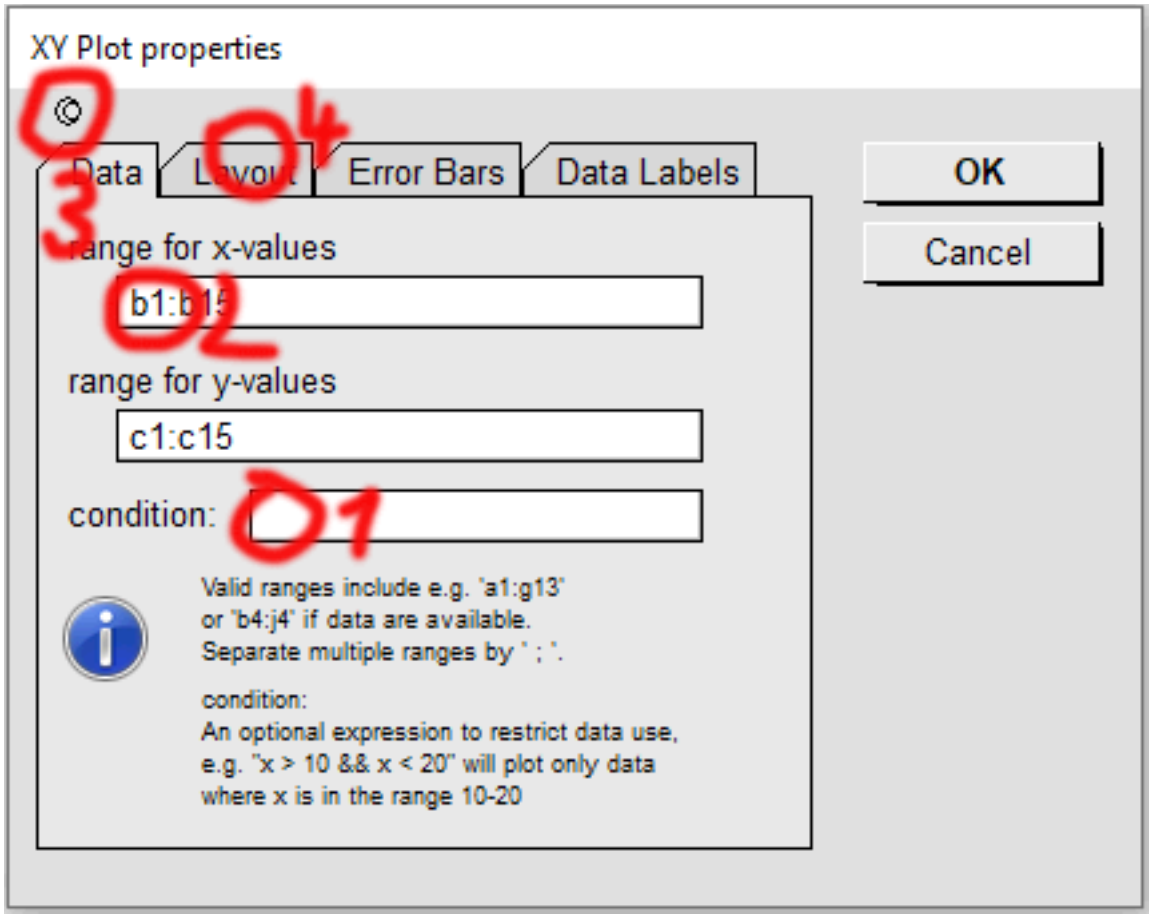


Fig. 4: Some features of a RLPlot dialog. 1 an input text field; 2 an input range field; 3 a check pin; 4 a dialog tab.

1) Input text fields are used to manually enter text and/or values. This field behaves identical to a cell of the spread sheet. You may copy and paste to this field. Also, formulas may be entered but these will be evaluated and only the result of this evaluation used as argument.

2) The input range field is a special input text field. It gives a reference to the spreadsheet data being used. It is filled with this range identifiers if in a previous operation ranges have been selected in the spreadsheet. Certainly, ranges can be also entered manually via the keyboard. Dialogs with an input range field usually also have a check pin (3).

3) The check pin. The default behavior of dialogs is that they are closed upon loose of focus. This behavior is modified by the check pin. You may toggle it on or off. If this pin is active the following procedure is possible: Place the text cursor into the input range field, go to the spreadsheet and select a range. This selected range descriptor is now copied to the input range field.

4) Some dialogs expand functionality by the use of tabs. The information within is usually optional and not always required. Have a look what's behind.

Formulas and Functions

The RLPlot spreadsheet has some basic functionality as known from other spreadsheet programs. Formulas start with an equal sign. E.g., the expression =sum(a1:a10) will display the sum of all variables in the range a1:a10. Formulas may contain references to cells and constants. =sum(100; a1:a10) will add 100 to above sum. Spread sheet cells are read/write. =d1 = sum(a1:a10) will display the sum of all variables in the range a1:a10 and additionally place the result in cell d1. =1+2 will display 3 but ="1"+"2" will display 12. The syntax =<cell>=<any function> can be useful to copy the result of a function to a different cell without coping the formula.

For a detailed list of available functions visit 'RLPlot formula parser' section.

In fact this also applies to dialogs. If you enter a string like '=2+3' into a field expecting numerical input the content is immediately replaced by '5'.

Date and Time Functions

RLPlot supports full calendar and time support. Date and time values are internally represented as days after 1900 with the fraction of the day behind the decimal point. Thus, these values may be used like numerical values. If you enter for example, '28.2.21' let's say into cell c1 it is recognized as a date value and displayed as '28.02.2021'. The actual display formats my be set in the menu ?/Configure/Date Time. You can use it for all operations. If you enter let's say into cell c2 '=c1+1' the displayed value in c2 will be '01.03.2021'. Changing the value of c1 into '28.2.20' the display in c2 will change to '29.2.2020' as the year 2020 is a leap year.

Sorry to those working with historical dates. Date values before 1900 are not supported and the calendar is Gregorian.

The rationale of implementing calendar functionality in RLPlot is to enable true date/time axes in graphs using date or time values as variables.

Spreadsheet Error Messages

#ERROR: There is a syntax error in the formula.

#CIRC: Circular reference in a formula. This message can occur when a formula is executed in a cell and the cell itself is referenced from inside the formula. In most cases there is a workaround inside RLPlot returning the last valid result of this cell.

#VALUE: The formula could not be evaluated. This is probably due to a corruption of internal data structures. You probably should never see this message.

#ARRAY: This is actually not an error message but indicates that the cell contains an array of data which cannot be displayed.

Fitting Functions with RLPlot

Fitting Functions is sometimes more an art than just a routine method. First some general considerations:

1) All dialogs are transparent to the RLPlot spreadsheet. Thus parameter names like `a1, b12, ab3` reference spreadsheet cells and read/write operations on these parameters modify spreadsheet values. Parameter names like `Start, const1, Km, Vmax` are treated as local values.

2) Parameters with starting values 0 cannot be optimized.

3) RLPlot offers two algorithms for fitting the curve. The Levenberg Marquart Algorithm is parametric. It may be considered as the workhorse for most functions. The Simplex Algorithm is a non parametric try and error method. It is usually slower but may be method of choice if a differential of the function does not make sense, for example fitting non-continuous functions or functions reaching undifined results.

4) It is probably a good idea to have a copy of the parametra and function in a text file and to copypaste them to the dialogs as required.

If an error message “Singular matrix“ is issued, this usually points to too few or malformed data.

Example 1: Fitting a Michaelis-Menten hyperbola

The example data:

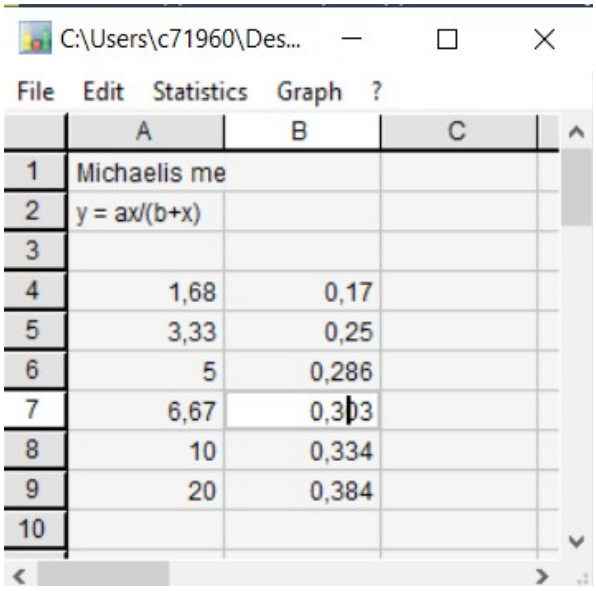


Fig. 5: Example data for a Michaelis-Menten curve.

Go to the Fit-Function dialog by pressing the button in the Create Graph dialog.

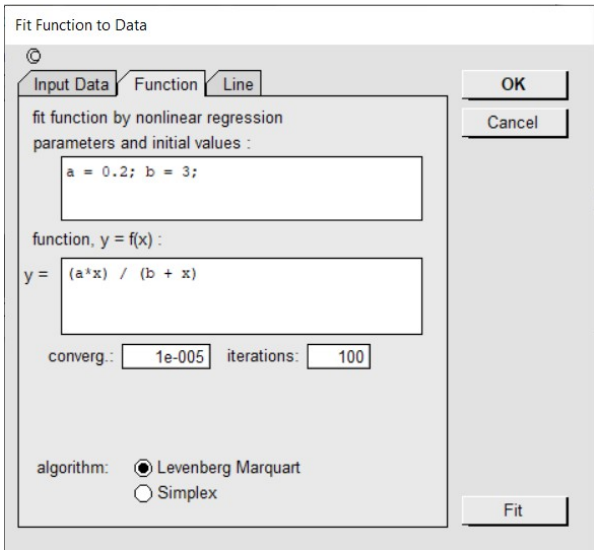


Fig.6: Initial settings of the Dialog

Pressing the [Fit] button gives the following result:

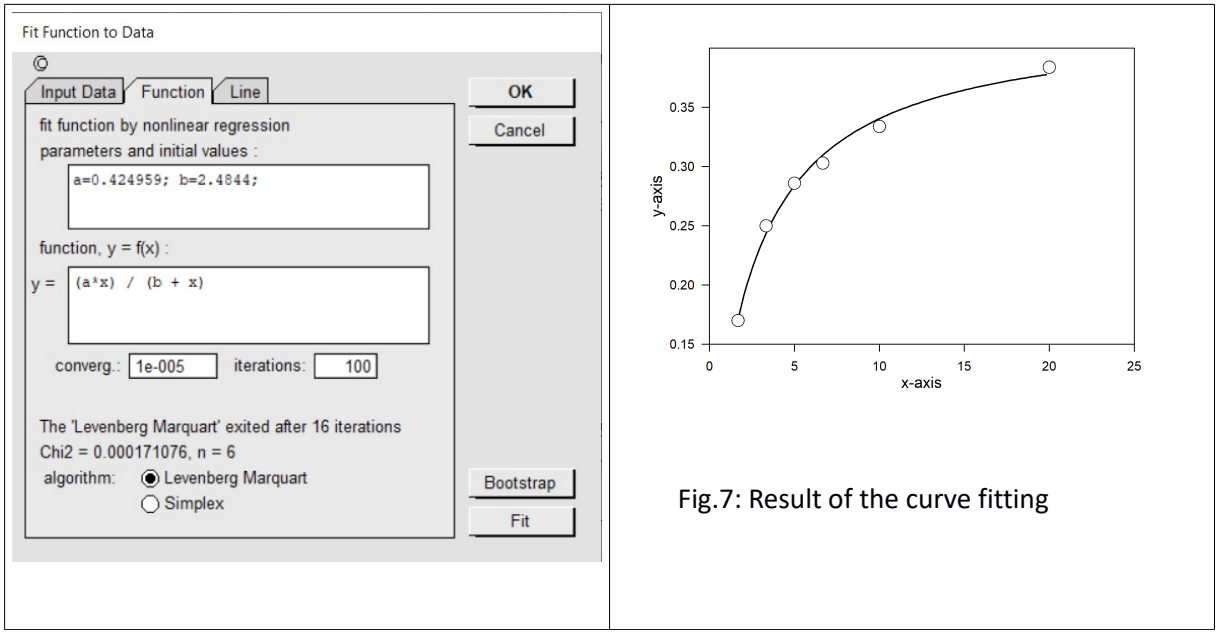
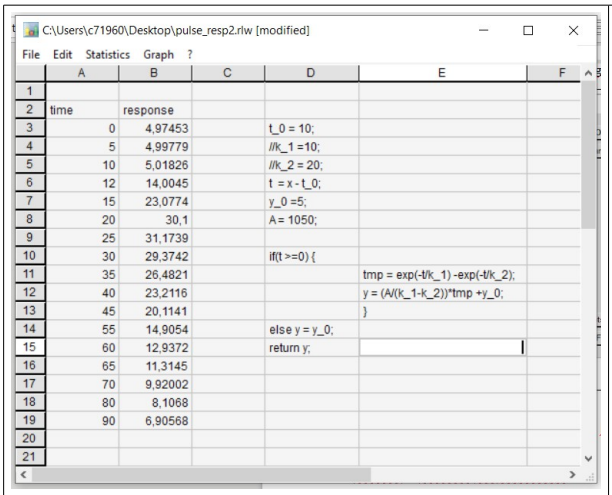


Fig.7: Result of the curve fitting

After a successful fit the button [Bootsrap] is available which can give you some statistical evaluation on the quality of the fit.

Example 2: Involving the spreadsheet

First some sample data ...



	A	B	C	D	E	F
1						
2	time	response				
3	0	4,97453		t_0 = 10;		
4	5	4,99779		//k_1 =10;		
5	10	5,01826		//k_2 = 20;		
6	12	14,0045		t = x - t_0;		
7	15	23,0774		y_0 =5;		
8	20	30,1		A = 1050;		
9	25	31,1739				
10	30	29,3742		if(t >=0) {		
11	35	26,4821			tmp = exp(-t/k_1) -exp(-t/k_2);	
12	40	23,2116			y = (A/(k_1-k_2))*tmp +y_0;	
13	45	20,1141			}	
14	55	14,9054		else y = y_0;		
15	60	12,9372		return y;		
16	65	11,3145				
17	70	9,92002				
18	80	8,1068				
19	90	6,90568				
20						
21						

Fig.8: Sample data describing a pulse response.

The data in the range a1:b20 describe a pulse response. The range d1:d20 contain a script. In general scripts and functions in RLPlot largely follow C-language conventions. Some details:

parameters:	t_0 i	s the start time of the pulse
	k_1, k_2	are time constants for rise and fall of the function in this example they are commented out because this parameters will be fitted.
	t	is the time correct for the start time
	y_0	is a fixed offset of the function
	A	a scaling factor
	tmp	a local variable: local variable need not explicitly defined. They are initialized with 0;
	y	a local variable but many parent functions read this variable as a result

In this example the return command in line 15 is actually not needed as a script always returns the result of the last operation. Some parameters (t_0, y_0 …) are defined inside the script, they are excluded from optimization. The may be moved to the parameter section for further refinement of the fit.

Select the ranges a1:b20 and go to the Fit-Function dialog by pressing the button in the Create Graph dialog.

The Function Tab of the Fit Function dialog is filled now like this:

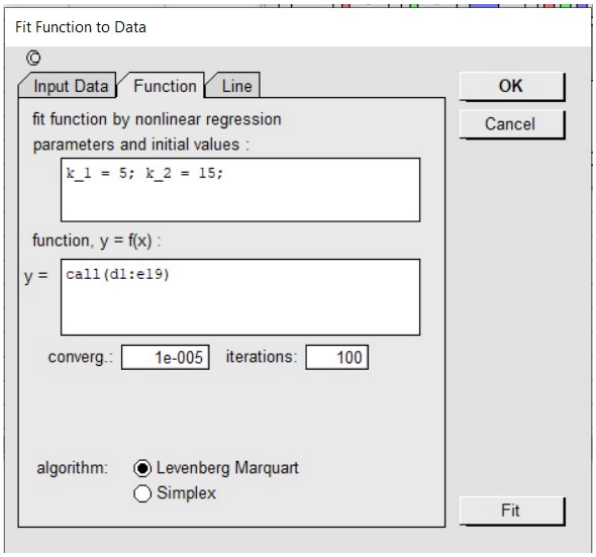


Fig. 9: Start screen of the fit.

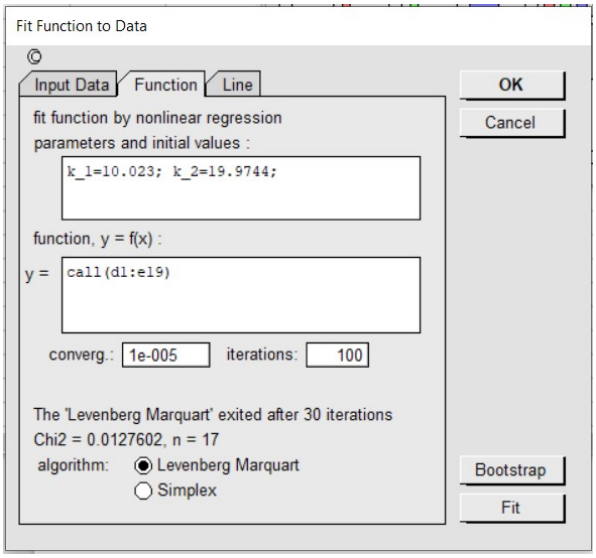
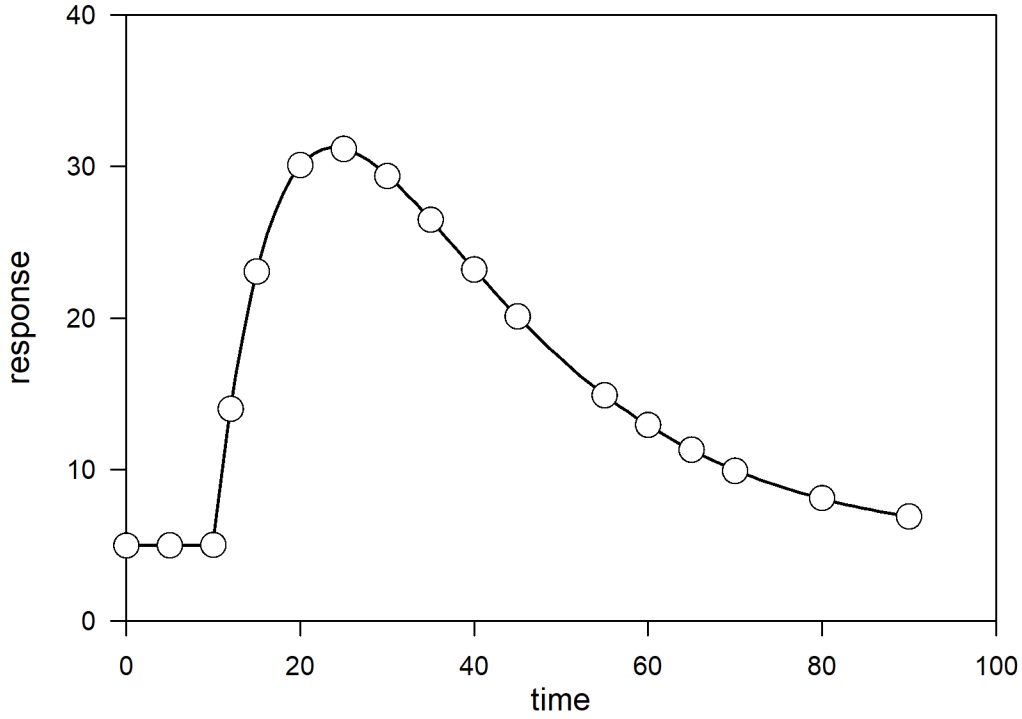


Fig 10: Result of the curve fit.

The actual curve fit is shown after pressing the [OK] button:

Fig. 11: The source data and the fitted function.



Note that different start options may give slightly different results.

RLPlot formula parser

Contents:

RLPlot uses a common parser for spread sheet cells, function plotting and curve fitting. This parser interprets formulas which are readable to humans and executes them on the computer. Spread sheet formulas are preceded by a '=' sign. For example '=1+2' assigns a value of 3 to the current cell and '=b3+2' assigns the value of cell b3 to the cell and increments it by 2. Function plots accept formulas with several expressions on a line separated by ';' or on different lines. By default the last expression is used as result. In this case you may explicitly state 'y= ...' to determine which expression gives the result.

Operators:

^	exponentiation	**>=**	logical greater or equal
*****	multiplication	**&&**	logical AND
/	division	**||**	logical OR
+	addition	**++**	increment
-	subtraction	**--**	decrement
==	logical equal	**+=**	add value
<	logical less than	**-=**	subtract value
>	logical greater than	***=**	multiply with value
!=, <>	logical not equal	**/=**	divide by value
<=	logical less or equal		

Conditional assignment

A conditional assignment has the following syntax: `exp1 ? exp2 : exp3`.
This means that the whole expression evaluates to `exp2` if `exp1` is *true*, and to `exp3` if `exp1` is *false*.

Select cases '`where`'

Functions operating on ranges or arrays can be modified by adding a '`where`' clause. The most common use of this clause is probably the count() function. A dummy variable, `$$`, is introduced which represents each element of the array. This clause is available for all functions operating on ranges and arrays.

Examples: `=count(a1:b20 where $$>0)`

count all values in a1:b20 which are higher than 0.0

`=count(a1:b20 where $$>=10 && $$<20)`

count all values in a1:b20 which are between 10 and 20

`=mean(a1:b20 where $$!=0)`

calculate the mean of all values in a1:b20 which are not zero

String operations

Strings can be concatenated using a '+' sign. This may also be used to combine strings and values.

Note: This operation results in a string which cannot be used for further calculations.

Examples:
```
=a3/b1*100+" %"
="Value is "+a2>a1?"higher":"lower"
```

Constants

Constants usually consist of digits, decimal point, and a preceding unary '-'.
Valid constants are: 1, 3.14, -123, 1.0e-13.

Predefined constants:
pi = 3.14159265358979
e = 2.71828182845905
true = 1
false = 0
inf = a very huge number

Variables

Local variables are any combination of letters. They are case sensitive and may contain digits bracketed by letters.

Spread sheet cells may be considered as global variables. They may be used as arguments on the right and left side of an assignment. References to spread sheet cells begin with a letter and end with a digit (any '`$`' sign is ignored). A reference to outside the spread sheet is treated like a local variable.

Examples:

`ab=1+2*3`	assign a value of '`7`' to the local variable '`ab`'
`b3=a1+2*3`	take the value of cell '`a1`', increment the value by '`6`' and place the result in the spread sheet cell '`b3`'

Predefined variables:

`zdiv = 1.0`	return this value upon zero divide error
`$$`	syntactical part of the '`where`' clause

NOTE: Names of predefined constants and functions are reserved. They must not be used as variable names. All these names are lowercase. A save way to define variables is to include uppercase letters or an underscore '_' in the variable names.

Arithmetic Functions

The following functions may be used in formulas and expressions:

abs(expr)
absolute value of 'expr'

acos(expr)
arccosine of 'expr'

asin(expr)
arcsine of 'expr'

atan(expr)
arctangent of 'expr'

cos(expr)
cosine of 'expr'

cosh(expr)
hyperbolic cosine of 'expr'

exp(expr)
exponential of 'expr'

floor(expr)
truncate 'expr' to integer

ln(expr)
natural logarithm of 'expr', synonym to 'log(expr)'

log(expr)
natural logarithm of 'expr', synonym to 'ln(expr)'

log10(expr)
base-10 logarithm of 'expr'

rand()
return a random number in the range 0.0 – 1.0

sign(expr)
sign (1, 0, or -1) of 'expr'

sin(expr)
sine of 'expr'

sinh(expr)
hyperbolic sine of 'expr'

sqrt(expr)
square root of 'expr'

srand(expr)
use 'expr' as new seed for random numbers

tan(expr)
tangent of 'expr'

tanh(expr)
hyperbolic tangent of 'expr'

String Functions

asc(string)
returns a value representing the character code corresponding to the first letter of string.

call(string[;parameters])
executes the script in 'range'. An optional string defining 'parameters' may be specified. 'Call' uses the same name space as the calling process and variables of the calling process are global to the called script.
see also exec()

chr(value)
returns a string containing the character associated with the character code value. If value is not a valid number '?' is returned.

eval(expr)
evaluates the expression 'expr' and returns the result.
Examples: eval("1+2") returns 3
a=2; eval("a*4") returns 8
eval('"hello"+" world"')
returns "hello world"

exec(range[;parameters])
executes the script in 'range'. An optional string defining 'parameters' may be specified. 'Exec' creates a new instance of the parser and variables of the calling process are hidden.
see also call()
example: exec(c1:e10; "x=4")

ltrim(string)
removes leading white space from string.

rtrim(string)
removes trailing white space from string.

strlen(string)
returns the length of string.
Example: strlen("hello") returns 5.

strpos(needle; haystack)
returns the position of the first occurrence of needle in haystack, -1 on error.
example: strpos("r"; "world") returns 2.

strrepl(search; replace; haystack)
replaces all occurences of search with replace in haystack.
Example: strrepl("a"; "e"; "hallo") returns "hello".

substr(text; pos1; pos2)
returns a string containing the characters of text from pos1 through pos2.
Example: substr("hello"; 2; 3) returns "ll".

tolower(string)
convert string to lower case.

toupper(string)
convert string to upper case.

trim(string)
removes leading and trailing white space from string.

ucfirst(string)
make first character of string upper case.

ucword(string)
make first character of every word upper case.

Date- and Time Functions

RLPlot's internal storage of date- and time data is a floating point number representing days after 1-Jan-1900 starting with 1 while the decimals give the fraction of the day.

date(value)
make value a date

datestr(datetime[, format])
convert the datetime value to a string, optionally using the format given

datetime(value)
make value a date and time

dateval(string[, format])
convert string to a datetime value, optionally using the format given

dom(datetime)
extracts day of month from datetime (1-31)

dow(datetime)
extracts day of week from datetime (1-7)

doy(datetime)
extracts day of year from datetime (1-366)

hours(datetime)
extracts hours of day from datetime (0-23)

leapyear(year)
returns true if year is a leapyear
example: leapyear(2000) returns ' true '

minutes(datetime)
extracts minutes from datetime (0-59)

month(datetime)
extracts month from datetime

now()
return the current system time

seconds(datetime)
extracts seconds from datetime (0-59)

time(value)
make value a time

today()
return the current system date

year(datetime)
extracts year from datetime

Formatting Date- and Time Values

RLPlot's internal storage of date- and time data is a floating point number representing days after 1-Jan-1900 starting with 1. The decimals give the fraction of the day. Thus the *5-Jan-2006* equals to *38721*, and *5-Jan-2006 12:00:00* equals to *38721.5*. Thus a time value alone is in the range 0.0 - 1.0, *12:00:00* equals to *0.5*. The conversion between internal values and a human readable form is controlled by formats, text strings where each character stands for a specific conversion of the date- and time value. Characters not included in the table below are either copied to the output or ignored during input.

The following characters are used as format specifiers in the format string:

Char.	Description	Type[1)]	Char.	Description	Type[1)]
Y	four digits year	I/O	**y**	two digits year	I/O
X	full name of month	I/O	**x**	three characters of month name	I/O
V	two digits month	I/O	**v**	same as V, no leading zero	I/O
W	single character month	O			
Z	two digits day of month	I/O	**z**	same as Z, no leading zero	I/O
D	full name of weekday	O	**d**	three character weekday	O
E	two digits weekday	O	**e**	same as E, no leading zero	O
F	single character weekday	O			
H	two digits hours	I/O	**h**	same as H, no leading zero	I/O
M	two digits minutes	I/O	**m**	same as M, no leading zero	I/O
S	two digits seconds	I/O	**s**	same as S, no leading zero	I/O
T	seconds, two decimals	I/O	**t**	same as T, no leading zero	

[1)] format specifier is available for input and output (I/O) or only for output (O)

Examples:

Input	Display
=val=dateval("05.01.2006")	38721
=dom(val)	5
=month(val)	1

The default formats used by RLPlot may be configured in the '?/Configure/DateTime ' dialog.

Statistical Functions

Some of these functions use a range to return results.

average(array)
returns the average of 'array', a synonym for 'mean(array)'

beta(u, v)
evaluate the beta function with shape factors 'u' and 'v'

betai(u, v, x)
the incomplete beta function

bincof(n, k)
the binomial coefficient for 0 < k < n

binomdist(s, n, p)
the cumulative binomial distribution for 0 < s < n, and 0 < p < 1

binomfreq(s, n, p)
the frequecies (densities) of the binomial distribution for 0 < s < n, and 0 < p < 1

cauchydist(x, location[, scale=1])
the cumulative Cauchy (Lorentz) distribution

cauchyfreq(x, location[, scale=1])
densities of the Cauchy (Lorentz) distribution

cauchyinv(p, location[, scale=1])
the inverse of the cumulative Cauchy (Lorentz) distribution

chidist(chi2, df)
returns probability for 'chi2' with 'df' degrees of freedom

chifreq(chi2, df)
the chi2 density (frequencies) function

chiinv(p, df)
returns the critical value for chi2 for probability 'p' with 'df' degrees of freedom,
a replacement for printed versions of the chi2 distribution.

lognormdist(x, m, s)
cumulative lognormal distribution

lognormfreq(x, m, s)
lognormal density function

lognorminv(p, m, s)
inverse cumulative lognormal distribution

max(array)
highest value of 'array'
examples: max(1, 2, 3), max(a1:a10), max(a1:a10, c1:d10)

mean(array)
returns the arithmetic mean of 'array'
examples: mean(1, 2, 3), mean(a1:a10), mean(a1:a10, c1:d10)

median(array)
returns the median of 'array', a synonym for quartile2(array)

min(array)
lowest value of 'array'
examples: min(1, 2, 3), min(a1:a10), min(a1:a10, c1:d10)

normdist(x, m, s)
probability of the cumulative normal distribution at 'x' with mean 'm' and standard deviation 's', a replacement for printed versions of the cumulative normal distribution

normfreq(x, m, s)
frequency (density) of the normal distribution at 'x' with mean 'm' and standard deviation 's'.

norminv(p, m, s)
return the critical value of the cumulative normal distribution for probability 'p' with mean 'm' and standard deviation 's', a replacement for printed versions of the inverse cumulative normal distribution.

pearson(array1; array2[;"destination"])

classes(start, step, array, range)
create a frequency distribution using 'start' and 'step' to define classes for 'array'. Returns the number of items in 'array' as result and the size of each class in the destination range.
example: classes(0;1;a1:a10;b1:b5)

correl(range1; range2[;"destination"])
A synonym for pearson(). For non parametric correlations see spearman() and kendall().

count(array)
count elements of 'array'
examples: count(1, 2, 3), count(a1:a10), count(a1:a10, c1:d10)

covar(range1; range2)
returns the covariance between array1 and array2.

erf(expr)
evaluate the error function at 'expr'

erfc(expr)
the complementary error function

expdist(x, lamda)
the cumulative exponential distribution

expfreq(x, lamda)
the density function of the exponential distribution

expinv(p, lamda)
the inverse of the cumulative exponential distribution

factorial(expr)
returns the factorial of 'expr' (expr!) where expr is a positive integer

fdist(f, df1, df2)
returns the probability (1-alpha) of 'f' with 'df1' degrees of freedom in the numerator and 'df2' in the denominator

ffreq(f, df1, df2)
the probability density function of the F-distribution

Pearsons parametric correlation.
If destination is given this range is filled with Pearsons r, Fisher's z, the probability of r <> 0, and the number of valid cases.

poisdist(x, m)
the cumulative poisson distribution

poisfreq(x, m)
the frequencies (densities) of the poisson distribution

ptukey(q, nmeans, df[,nranges=1])
the cumulative density function of the Studentized Range Distribution.

qtukey(p, nmeans, df[,nranges=1])
returns the quantiles of the Studentized Range Distribution, the inverse of ptukey().

quartile1(array)
calculates the 25% quartile of 'array'

quartile2(array)
calculates the 50% quartile of 'array', a synonym for median(array)

quartile3(array)
calculates the 75% quartile of 'array'

rank(value; array)
returns the rank of 'value' in 'array' with mid-tie ranking. If 'value' is not present in 'array' a rank of 0 is returned.

regression(range1; range2; "destination")
linear regression analysis of independent variable 'array1' and dependent variable 'array2'. The destination range is filled with slope, intercept, mean1, mean2, SE of slope, variance1, variance2, variance of fit, F of regression and significance.
Example: regression(a1:a10;b1:b10;"c1:c10")

skew(array)
returnes the skewness of 'array'

finv(p, df1, df2)
returns the critical value of the F-distribution, a replacement for printed versions of the F-distribution

ftest(array1; array2[;"destination"])
compare variances and return the probability of the F-statistics.
If destination is given this range is filled with mean, SD, n of array1 and mean, SD, n of array2.
examples: ftest(a1:a10;b1:b10),
ftest(a1:a10; 1,2,3,4; "c1:c10")

gammaln(x)
returns the natural logarithm of the gamma function, ln(Γ(x))

gammp(a, x)
the incomplete gamma function

gammq(a, x)
complementary incomplete gamma function

geomdist(x, p)
the cumulative geometric distribution

geomfreq(x, p)
densities of the geometric distribution

gmean(array)
returnes the geometric mean of 'array'

hmean(array)
returnes the harmonic mean of 'array'

hyperdist(k, N, m, n)
the cumulative hypergeometric distribution

hyperfreq(k, N, m, n)
densities of the hypergeometric distribution

kendall(range1; range2[;"destination"])
Kendall's non parametric rank correlation.
If destination is given this range is filled with Kendall's tau, the number of standard deviations from zero, the two sided significance level, and the number of valid

spearman(range1; range2[;"destination"])
Spearmans non parametric rank correlation.
If destination is given this range is filled with sum of squared rank differences, number of SD's the sum differs from expected, the significance of this SD, Spearmans rs, the probability of rs <> 0, and the number of valid cases.

stdev(array)
standard deviation of mean
examples: stdev(1, 2, 3), stdev(a1:a10),
stdev(a1:a10, c1:d10)

sterr(array)
standard error of mean
examples: sterr(1, 2, 3), sterr(a1:a10),
sterr(a1:a10, c1:d10)

sum(array)
sum of all values of 'array'
examples: sum(1, 2, 3), sum(a1:a10),
sum(a1:a10, c1:d10)

tdist(t, df)
returns the probability (1-alpha) of 't' with 'df' degrees of freedom

tfreq(t, df)
the probability density function of the t-distribution

tinv(p, df)
returns the critical t of Student's t-distribution, a replacement for printed versions of the t-distribution

ttest(array1; array2[;"destination"])
compare means and return the probability of the t-statistics.
If destination is given this range is filled with mean, SD, n of array1, mean, SD, n of array2, probability for equal variances, Welch's corrected df, corrected probability.
examples: ttest(a1:a10;6,7,4),
ttest(a1:a10; b1:b10; "c1:c10")

cases.

kurt(array)
returnes the kurtosis of 'array'

logisdist(x, location[, scale=1])
the cumulative logistic distribution

logisfreq(x, location[, scale=1])
densities of the logistic distribution

logisdist(p, location[, scale=1])
the inverse cumulative logistic distribution

ttest2(range1; range2[;"destination"])
paired t-test for dependent samples. If destination is given this range is filled with mean and SD of range1, mean and SD of range2, number of valid cases and probability.

utest(array1; array2[;"destination"])
Mann-Whitney U Test, a non parametric alternative to t-test().
If destination is given this range is filled with rank sum 1, rank sum 2, U, Z, n1, n2, p-level, Z corrected for ties, and the corrected p-level.

variance(array)
variance of 'array'
examples: variance(1, 2, 3), variance(a1:a10), variance(a1:a10, c1:d10)

weibdist(x, shape[, scale=1])
the cumulative weibull distribution

weibfreq(x, shape[, scale=1])
densities of the weibull distribution

weibinv(p, shape[, scale=1])
inverse cumulative weibull distribution

Array Functions

Arrays are lists of numerical values. Note that some of the statistical functions are in fact array functions.

asort(array)
sorts 'array' and returns the sorted array

asort2(array1, array2)
sorts 'array1' making the corresponding rearrangements in 'array2'. The function returns the number of elements which have been sorted.

count(array)
return the number of elements in 'array'

crank(array)
replaces all elements of 'array by their rank. 'array' must be sorted before 'crank' is called. The function returns theta.

fill(array; dest)
copies 'array' to the spreadsheet range given by dest. The function returns the number of elements copied.
example: a=(a1:a10); fill(a;"b1:b10")
copy data from the spreadsheet to array 'a' and then the array back to the spreadsheet.

invert(array)
inverts 'array' and returns the inverted array

mkarr(range; MD)
returns an array which contains the same number of elements as 'range'. All missing and non-numerical data are replaced by the MD value.
example: a=(a1:a10;-9999)

randarr(array)
randomizes 'array' and returns the new array

resample(array)
resamples 'array' and returns the new array

subarr(array1; pos1[; pos2])
return an array containing the elements pos1 through pos2 of array1.

sum(array)
sum of all values of 'array'