



FOXES TEAM

---

Reference for Xnumbers.xla

# **Numeric Calculus in Excel**

REFERENCE FOR XNUMBERS.XLA

# **Numeric Calculus in EXCEL**

---

© 2005, by Foxes Team  
ITALY  
Sept 2005

# Index

<b>About this Tutorial .....</b>	<b>8</b>
<b>Array functions .....</b>	<b>9</b>
What is an array-function? .....	9
How to insert an array function.....	9
<b>How to get the help on line .....</b>	<b>13</b>
<b>Xnumbers installation .....</b>	<b>15</b>
How to uninstall .....	16
<b>Multiprecision Floating Point Arithmetic.....</b>	<b>17</b>
Why using extended precision numbers? .....	17
Multiprecision methods.....	19
How to store long number .....	19
<b>Functions.....</b>	<b>21</b>
General Description.....	21
Using Xnumbers functions .....	21
Using extended numbers in Excel.....	22
Functions Handbook .....	23
Precision.....	24
Formatting Result.....	24
Arithmetic Functions.....	25
Addition.....	25
Subtraction.....	25
Accuracy lack by subtraction .....	25
Multiplication .....	26
Division .....	26
Inverse.....	26
Integer Division .....	27
Integer Remainder .....	27
Sum .....	29
Product .....	29
Raise to power.....	30
Square Root.....	30
N <sup>th</sup> - Root .....	30
Absolute.....	31
Change sign.....	31
Integer part .....	31
Decimal part.....	31
Truncating.....	32
Rounding .....	32
Relative Rounding .....	33
Extended Numbers manipulation .....	34
Digits count.....	34
Significant Digits count.....	34
Compare numbers .....	34
Extended Numbe Check.....	35
Format Extended Number.....	35
Check digits .....	36
SortRange.....	36
Digits sum.....	36
Vector Inversion.....	37
Scientific Format .....	37
Split scientific format.....	37
Convert Extended Number .....	38
Macros X-Edit .....	39
Statistical Functions .....	41
Factorial.....	41
Factorial with double-step.....	41

Combinations.....	41
Permutations.....	42
Arithmetic Mean.....	42
Geometric Mean.....	42
Quadratic Mean.....	43
Standard Deviation.....	43
Variance.....	43
Linear Regression Coefficients.....	44
Linear Regression Formulas.....	47
Linear Regression Covariance Matrix.....	48
Linear Regression Statistics.....	49
Linear Regression Evaluation.....	50
Summary of Linear Regressions.....	51
Sub-Tabulation.....	52
Data Conditioning.....	52
Data Conditioned Linear Regression Coefficients.....	53
Linear Regression with Robust Method.....	55
Linear Regression Min-Max.....	56
Certification Results for Linear Regression.....	58
Transcendental Functions.....	61
Logarithm natural (Napier's).....	61
Logarithm for any base.....	61
Exponential.....	61
Exponential for any base.....	61
Constant "e".....	62
Constant Ln(2).....	62
Constant Ln(10).....	62
Hyperbolic Sine.....	62
Hyperbolic ArSine.....	62
Hyperbolic Cosine.....	63
Hyperbolic ArCosine.....	63
Hyperbolic Tangent.....	63
Hyperbolic ArTangent.....	63
Euler's constant gamma.....	64
Trigonometric Functions.....	65
Sin.....	65
Cos.....	65
Computation effect of $\cos(\pi/2)$ .....	65
Tan.....	66
Arcsine.....	66
Arccosine.....	66
Arctan.....	66
Constant $\pi$ .....	66
Complement of right angle.....	67
Polynomial Rootfinder.....	68
Input parameters.....	69
Printing Results.....	70
How to use rootfinder macros.....	71
Root Error Estimation.....	72
Integer roots.....	74
Central Polynomial.....	77
Coefficients Transformation.....	78
Circle of the Roots.....	79
Polynomial Functions.....	80
Polynomial evaluation.....	80
Polynomial derivatives.....	82
Polynomial coefficients.....	83
Polynomial writing.....	84
Polynomial addition.....	84
Polynomial multiplication.....	84
Polynomial subtraction.....	85
Polynomial division quotient.....	85
Polynomial division remainder.....	85
Hermite's and Cebychev's polynomials.....	86
Legendre's Polynomials.....	86
Polynomial shift.....	89
Polynomial center.....	89
Polynomial roots radius.....	90
Polynomial building from roots.....	91

Polynomial building with multi-precision.....	93
Polynomial solving .....	94
Integer polynomial .....	94
Polynomial interpolation.....	95
Sub-tabulation.....	95
Polynomial System of 2 <sup>nd</sup> degree.....	97
Bivariate Polynomial .....	98
Partial fraction decomposition.....	99
Orthogonal Polynomials.....	102
Orthogonal Polynomials evaluation .....	103
Weight of Orthogonal Polynomials.....	106
Zeros of Orthogonal Polynomials.....	106
Coefficients of Orthogonal Polynomials.....	107
<b>Complex Arithmetic and Functions.....</b>	<b>108</b>
How to insert a complex number .....	108
Complex Addition.....	109
Complex Subtraction .....	109
Complex Multiplication.....	109
Complex Division .....	110
Polar Conversion .....	110
Rectangular Conversion .....	110
Complex absolute.....	111
Complex power.....	111
Complex Roots.....	111
Complex Log.....	112
Complex Exp .....	112
Complex inverse .....	112
Complex negative .....	113
Complex conjugate.....	113
Complex Sin .....	113
Complex Cos .....	113
Complex Tangent .....	113
Complex ArcCos.....	113
Complex ArcSin.....	114
Complex ArcTan.....	114
Complex Hyperbolic Sine.....	114
Complex Hyperbolic Cosine.....	114
Complex Hyperbolic Tan.....	114
Complex Inverse Hyperbolic Cos.....	114
Complex Inverse Hyperbolic Sin.....	114
Complex Inverse Hyperbolic Tan.....	115
Complex digamma.....	115
Complex Exponential Integral.....	115
Complex Error Function.....	115
Complex Complementary Error Function.....	115
Complex Gamma Function .....	116
Complex Logarithm Gamma Function .....	116
Complex Zeta Function.....	116
Complex Quadratic Equation .....	117
<b>Number Theory .....</b>	<b>118</b>
Maximum Common Divisor.....	118
Minimum Common Multiple .....	118
Rational Fraction approximation .....	119
Check Prime .....	120
Next Prime.....	120
Modular Power.....	120
Perfect Square.....	121
Check odd/even.....	121
Factorize .....	121
Factorize function .....	122
Prime Numbers Generator.....	123
Fermat's Prime Test.....	123
Diophantine Equation .....	125
<b>Linear Algebra Functions .....</b>	<b>126</b>
Matrix Addition .....	126
Matrix Subtraction.....	126
Matrix Multiplication .....	126
Matrix Inverse .....	126
Matrix Determinant .....	127

Matrix Modulus .....	127
Scalar Product .....	127
Similarity Transformation .....	127
Matrix Power .....	128
Matrix LU decomposition .....	128
Matrix $LL^T$ decomposition .....	129
Vector Product .....	129
Solve Linear Equation System .....	130
Solve Linear Equation System with Iterative method .....	131
Square Delta Extrapolation .....	132
Multiprecision Matrix operations (macro) .....	134
<b>Integrals &amp; Series .....</b>	<b>136</b>
Discrete Fourier Transform .....	136
Discrete Fourier Inverse Transform .....	137
Discrete Fourier Spectrum .....	138
Inverse Discrete Fourier Spectrum .....	139
2D Discrete Fourier Transform .....	139
2D Inverse Discrete Fourier Transform .....	140
Macro DFT (Discrete Fourier Transform) .....	141
Macro Sampler .....	143
Data Integration (Romberg method) .....	146
Function Integration (Romberg method) .....	147
Function Integration (Double Exponential method) .....	148
Function Integration (mixed method) .....	150
Complex Function Integration (Romberg method) .....	152
Data Integration (Newton-Cotes) .....	154
Data integration for random point. ....	156
Function Integration (Newton-Cotes formulas) .....	156
Integration: symbolic and numeric approaches .....	158
Integration of oscillating functions (Filon formulas) .....	160
Integration of oscillating functions (Fourier transform) .....	162
Infinite Integration of oscillating functions .....	163
Double Integral .....	166
Double Integration macro .....	166
Double integration function .....	168
Infinite integral .....	170
Series Evaluation .....	173
Series acceleration with $\Delta^2$ .....	174
Complex Series Evaluation .....	175
Double Series .....	176
Trigonometric series .....	177
Trigonometric double serie .....	178
Discrete Convolution .....	180
<b>Interpolation .....</b>	<b>182</b>
Interpolation with continue fraction .....	182
Interpolation with Cubic Spline .....	184
Cubic Spline 2nd derivatives .....	185
Cubic Spline Coefficients .....	186
Multi-variables Interpolation .....	187
2D Interpolation .....	188
<b>Interpolation of Tabulated data function .....</b>	<b>189</b>
Cubic Spline interpolation .....	189
Cubic poly interpolation .....	192
Observations .....	193
Other test functions .....	194
High and low interpolation degree .....	195
Continued fraction interpolation .....	196
<b>Differential Equations .....</b>	<b>198</b>
ODE Runge-Kutta 4 .....	198
<b>ODE Multi-Steps .....</b>	<b>202</b>
Multi-step coefficients tables .....	203
Predictor- Corrector .....	205
PECE algorithm of 2 <sup>nd</sup> order .....	205
PECE algorithm of 4 <sup>th</sup> order .....	207
<b>Nonlinear Equations .....</b>	<b>209</b>
Bisection .....	209
Secant .....	210
<b>Derivatives .....</b>	<b>212</b>

First Derivative.....	212
Second Derivative.....	213
Gradient.....	213
Jacobian matrix.....	214
Hessian matrix.....	214
Non-linear equation solving with derivatives.....	216
<b>Conversions .....</b>	<b>218</b>
Decibel.....	218
Base conversion .....	218
Log Relative Error.....	219
<b>Special Functions.....</b>	<b>221</b>
Error Function $\text{Erf}(x)$ .....	221
Exponential integral $\text{Ei}(x)$ .....	221
Exponential integral $\text{En}(x)$ .....	221
Euler-Mascheroni Constant $\gamma$ .....	221
Gamma function $\Gamma(x)$ .....	222
Log Gamma function .....	223
Gamma quotient .....	223
Gamma F-factor.....	224
Digamma function.....	224
Beta function.....	224
Combinations function .....	225
Bessel functions.....	226
Cosine Integral $\text{Ci}(x)$ .....	226
Sine Integral $\text{Si}(x)$ .....	227
Fresnel sine Integral .....	227
Fresnel cosine Integral.....	227
Fibonacci numbers .....	227
Hypergeometric function .....	228
Zeta function $\zeta(s)$ .....	228
<b>Formulas Evaluation.....</b>	<b>230</b>
Multiprecision Expression Evaluation .....	230
Complex Expression Evaluation .....	233
Math expression strings.....	235
List of basic functions and operators .....	237
<b>Function Optimization.....</b>	<b>239</b>
Macros for optimization on site .....	239
Example 1 - Rosenbrock's parabolic valley .....	241
Example 2 - Constrained minimization .....	243
Example 3 - Nonlinear Regression with Absolute Sum.....	244
<b>References.....</b>	<b>246</b>
<b>Analytical index.....</b>	<b>249</b>

**WHITE PAGE**



## About this tutorial

*This document is the reference guide for all functions and macros contained in the Xnumbers addin. It is a printable version of the help-on-line, with a larger collection of examples.*



**XNUMBERS.XLA** is an Excel addin containing useful functions for numeric calculus in standard and multiprecision floating point arithmetic up to 200 significant digits.

The main purpose of this document is to give a reference guide for numeric calculus functions of this package, showing how to work with multiprecision arithmetic in Excel. Much of the material contained in this document comes from the Xnumbers help-on-line. You may print it in order to have a handle paper manual. This tutorial is written with the aim of teaching how to use the Xnumbers functions. Of course it speaks about math and numeric calculus but this is not a math book. You rarely find here theorems and demonstrations. You can find, on the contrary, many explaining examples.

I thank all those who suggested me to write this tutorial and - indeed - who encouraged me. I am grateful to all those who will provide constructive criticisms.

Special thanks to everyone that have kindly collaborated.

Leonardo Volpi

## Array functions

### What is an array-function?

A function that returns multiple values is called "array-function". Xnumbers contains many of these functions. Those that return a matrix or a vector are array functions. Matrix operations such as the inversion, the multiplication, the sum, etc. are examples of array-functions. Also complex numbers are arrays of two cells. On the contrary, in the real domain, the logarithm, the exponential, the trigonometric functions, etc. are scalar functions because they return only one value.



In a worksheet, an array-function always returns a (n x m) rectangular range of cells. To enter it, you must select this range, enter the function as usually and give the keys sequence CTRL+SHIFT+ENTER. Keep down both keys CTRL and SHIFT (do not care the order) and then press ENTER.

### How to insert an array function

The following example explains, step-by-step, how it works

#### The System Solution


Assume to have to solve a 3x3 linear system. The solution is a vector of 3 values.

$$Ax = b$$

Where:

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 3 & 4 \end{bmatrix} \quad b = \begin{bmatrix} 4 \\ 2 \\ 3 \end{bmatrix}$$

The function **SYSLIN** returns the solution **x**. To see the three values you must select before the area where you want to insert these values.

Now insert the function either from menu or by the icon 

	G5			=				
	A	B	C	D	E	F	G	H
1								
2		Ax = b						
3								
4		A			b		x	
5	1	1	1		4			
6	1	2	2		2			
7	1	3	4		3			
8								
9								
10								
11								

Select the area you want to paste the result x

# Xnumbers Tutorial

The screenshot shows an Excel spreadsheet with the following data:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1																	
2	Ax = b																
3																	
4	A				b		x										
5	1	1	1		4		=										
6	1	2	2		2												
7	1	3	4		3												

The 'Incolla funzione' (Paste Function) dialog box is open, showing the 'CATEGORIA' (Category) as 'Definite dall'utente' (Defined by the user) and the 'NOME FUNZIONE' (Function Name) as 'SYSLIN'. The description for SYSLIN is 'Solve Linear System.'

Select the area of the matrix **A** "A5:C7" and the constant vector **b** "E5:E7"

Excel spreadsheet showing the solution of a linear system using the SYSLIN function.

The formula bar displays: `=SYSLIN(A5:C7,E5:E7)`

The spreadsheet content is as follows:

	A	B	C	D	E	F	G	H	I	J	K	L	M	
1														
2		<b>Ax = b</b>												
3														
4		<b>A</b>				<b>b</b>		<b>x</b>						
5	1	1	1		4	5:E7								
6	1	2	2		2									
7	1	3	4		3									

The dialog box 'SYSLIN' displays the matrix selection process:

- Mat A5:C7 = {1;1;1;1;2;2;1;3;4}
- v E5:E7 = {4;2;3}
- = {6;-5;3}

Solve Linear System.

Risultato formula = 6

Now - **attention!** - give the "magic" keys sequence CTRL+SHIFT+ENTER

That is:

- Press and keep down the CTRL and SHIFT keys
- Press the ENTER key

All the values will fill the cells that you have selected.

	A	B	C	D	E	F	G	H	I
1									
2		Ax = b							
3									
4		A			b		x		
5	1	1	1		4		6		
6	1	2	2		2		-5		
7	1	3	4		3		3		
8									
9									
10									
11									
12									

Note that Excel shows the function around two braces { }. These symbols mean that the function return an array (you cannot insert them by hand).

An array function has several constrains. Any cell of the array cannot be modified or deleted. To modify or delete an array function you must selected before the entire array cells.

### Adding two matrices

The CTRL+SHIFT+ENTER rule is valid for any function and/or operation returning a matrix or a vector

Example - Adding two matrices

$$\begin{bmatrix} 1 & -2 \\ 2 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

We can use directly the addition operator "+". We can do this following these steps.

- 1) Enter the matrices into the spreadsheet.
- 2) Select the B8:C9 empty cells so that a 2 × 2 range is highlighted.
- 3) Write a formula that adds the two ranges. Either write =B4:C5+E4:F5 Do not press <Enter>. At this point the spreadsheet should look something like the figure below. Note that the entire range B8:C9 is selected.

	A	B	C	D	E	F
1						
2						
3						
4		1	-2		1	0
5		2	1		0	1
6						
7						
8		=B4:C5+E4:F5				
9						
10						

- 4) Press and hold down <CTRL> + <SHIFT>
- 5) Press <ENTER>.

If you have correctly followed the procedure, the spreadsheet should now look something like this

## Xnumbers Tutorial


	A	B	C	D	E	F
3						
4		1	-2		1	0
5		2	1		0	1
6						
7						
8		2	-2			
9		2	2			
10						
11						

This trick can also work for matrix subtraction and for the scalar-matrix multiplication, but not for the matrix-matrix multiplication.

Let's see this example that shows how to calculate the linear combination of two vectors

	A	B	C	D	E	F	G
10		<b>v1</b>		<b>v2</b>		<b>v3</b>	
11		1		0		34	
12	34	-2	22	1		-46	
13		4		-1		114	
14							
15							
16							
17							

{=A12\*B11:B13+C12\*D11:D13}



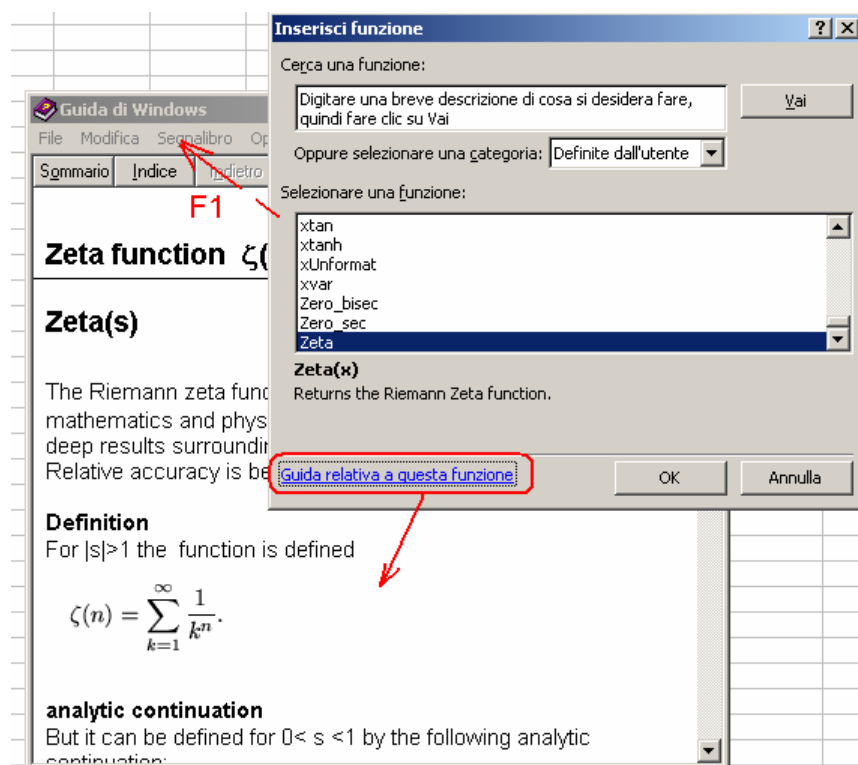
### Functions returning optional values

Some function, such as for example the definite integral of a real function  $f(x)$ , can return one single real value or optional extra data (iterations, error estimation, etc...)

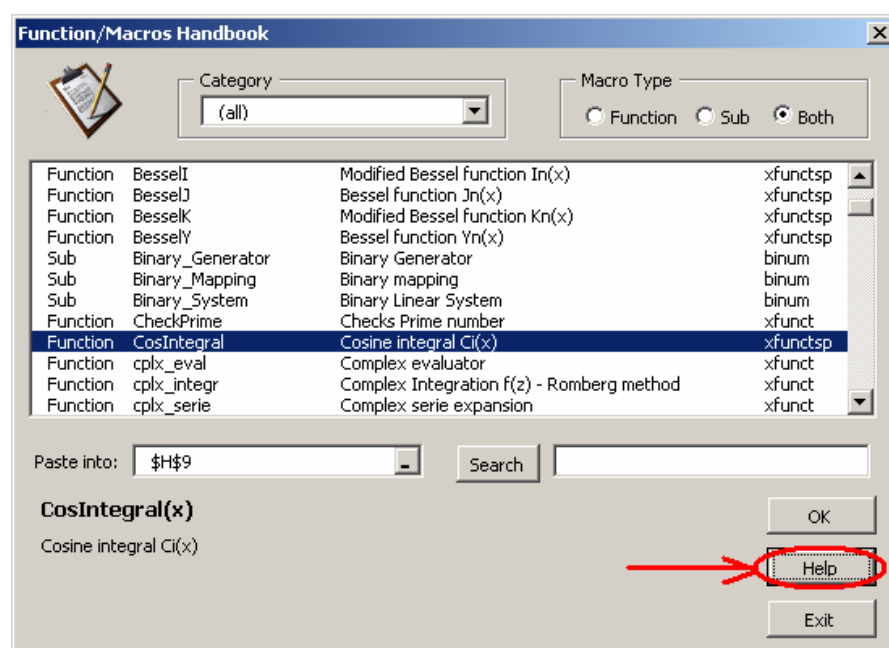
If you do not want to see this additional information simply select one cell and insert the function with the standard procedure. On the contrary, if you want to see also the extra information, you must select the extra cells needed and insert it as an array-function

## How to get the help on line

Xnumbers provides the help on line that can be recalled in the same way of any other Excel function. When you have selected the function that you need, press the **F1** key or click on the ["guide hyperlink"](#)



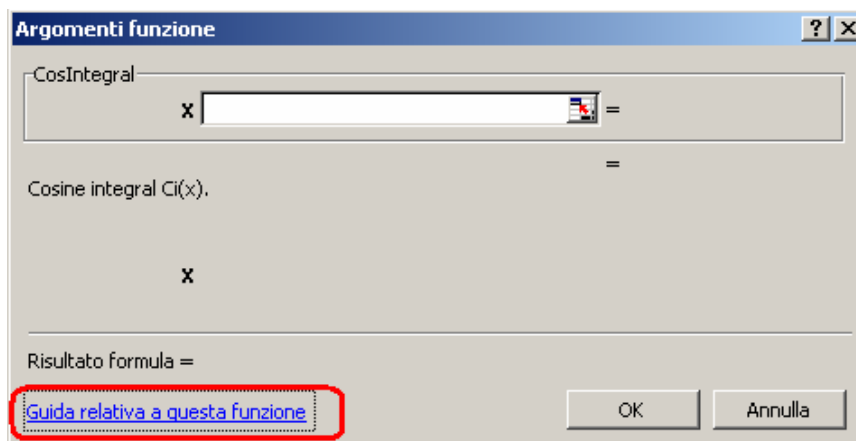
There is also another way to get the help-on-line. It is from the Xnumbers Function Handbook



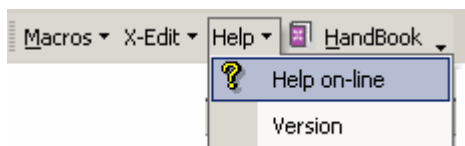
Select the function that you want and press the Help button

## Xnumbers Tutorial

You can also recall the help guide from the function wizard window



Of course you can open the help on-line from the Xnumber menu






or directly by double clicking on the Xnumbers.hlp file



## Xnumbers installation

This addin for Excel 2000/XP is composed by the following files:

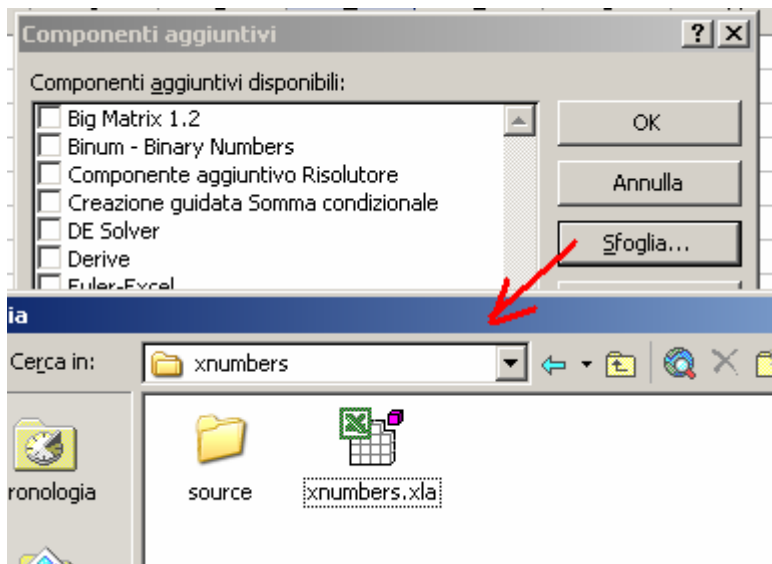
Addin file	Help file	Handbook file
(It contains the Excel macros and functions)	(It contains the help notes)	(It contains the macros and functions description list for the Xnumbers Handbook)
 xnumbers.xla	 Xnumbers.hlp	 xnumbers.csv

This installation is entirely contained in the folder that you specify.

Put all these files in a same directory as you like.

Open Excel and follow the usually operations for the addin installation:

- 1) Select <addins...> from <tools> menu,
- 2) Excel shows the Addins Manager,
- 3) Search for the file **xnumbers.xla**,
- 4) Press OK,

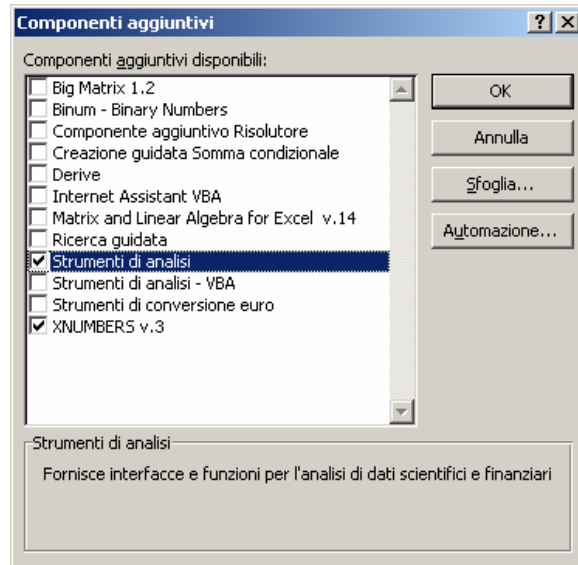


NB. Nella versione italiana di Excel, "Addin Manager" si chiama "Componenti aggiuntivi" e si trova nel menu <Strumenti> <Modelli e aggiunte...>



## Xnumbers Tutorial

After the first installation, Xnumbers.xla will be add to the Addin Manager

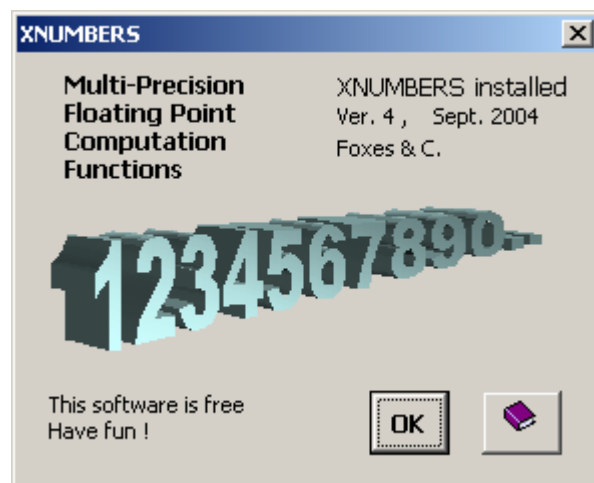


By this tool, you can load or unload the addins that you want, simply switching on/off the check-boxes.

At the starting, the addins checked in the Addins Manager will be automatically loaded

If you want to stop the automatic loading of xnumbers.xla simply deselect the check box before closing Excel

If all goes right you should see the welcome popup of Xnumbers. This appears only when you activate the check box of the Addin Manager. When Excel automatically loads Xnumbers, this popup is hidden.



### ***How to uninstall***

If you want to uninstall this package, simply delete its folder. Once you have cancelled the Xnumbers.xla file, to remove the corresponding entry in the Addin Manager, follow these steps:

- 1) Open Excel
- 2) Select <Addins...> from the <Tools> menu.
- 3) Once in the Addins Manager, click on the Xnumbers entry
- 4) Excel will inform you that the addin is missing and ask you if you want to remove it from the list. Give "yes".

## Multiprecision Floating Point Arithmetic

Any computer having hardware at 32-bit can perform arithmetic operations with 15 significant digits, at the most. The only way to overcome this finite fixed precision is to adopt special software that extends the accuracy of the native arithmetic

### Why using extended precision numbers?

First of all, for example, to compute the following operation:

$$\begin{array}{r} 90000000002341 \times \\ 8067 = \\ \hline 726030000018884847 . \end{array}$$

Any student, with a little work, can do it. Excel, as any 32-bit machine, cannot! It always gives the (approximate) result 726030000018885000 , with a difference of +153.

But do not ask Excel for the difference. It replies 0!

The second, deeper, example regards numeric analysis. Suppose we have to find the roots of a 9<sup>th</sup> order Polynomial.

$$P(x) = \sum_{i=0}^n a_i x^i$$

Where its coefficients  $a_i$  are listed in the table below.

Coefficients	
a9	1
a8	-279
a7	34606
a6	-2504614
a5	116565491
a4	-3617705301
a3	74873877954
a2	-996476661206
a1	7738306354988
a0	-26715751812360

There are excellent algorithms for finding a numerical solution of this problem. We can use the Newton-Raphson method: starting from  $x = 32$  and operating with 15 significant digits (the maximum for Excel), we have:

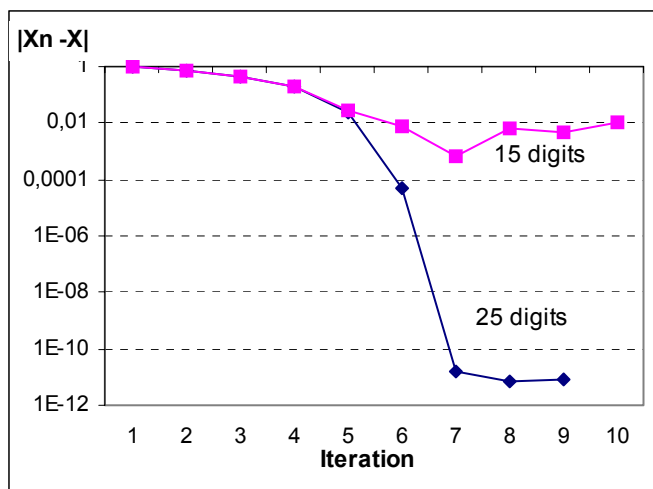
$x_n$	$P(x)$ (15 digit)	$P'(x)$ (15 digit)	$-P/P'$	$ x_n - x $
32	120	428	0,280373832	1
31,71962617	43,77734375	158,9873047	0,275351191	0,7196262
31,44427498	15,69921875	60,93164063	0,257652979	0,444275
31,186622	4,78125	29,46289063	0,162280411	0,186622
31,02434159	0,65625	24,10644531	0,02722301	0,0243416
30,99711858	-0,07421875	24,01953125	-0,003089933	0,0028814
31,00020851	0,23828125	24,04980469	0,009907825	0,0002085
30,99030069	-0,52734375	23,98925781	-0,021982495	0,0096993
31,01228318	0,2421875	24,02050781	0,01008253	0,0122832
31,00220065	-0,03515625	23,99023438	-0,00146544	0,0022007

As we can see, the iteration approaches the solution  $x = 31$  but the error  $|x_n - x|$  remains too high. Why? Multiple roots? No, because  $P'(x) \gg 0$ . Algorithm failed? Of

course not. This method is very well tested. The only explanation is the finite precision of the computation. In fact, repeating the calculus of  $P(x)$  and  $P'(x)$  with 25 significant digits, we find the excellent convergence of this method.

$x_n$	$P(x)$ (25 digit)	$P'(x)$ (25 digit)	$-P/P'$	$ x_n - x $
32	120	428	0,28037383	1
31,71962617	43,71020049043	158,979858019937	0,27494175	0,719626
31,44468442	15,71277333004	61,059647049872	0,25733482	0,444684
31,1873496	4,83334748037	29,483621556222	0,1639333	0,18735
31,02341629	0,56263326884	24,082301045236	0,02336294	0,023416
31,00005336	0,00128056327	24,000000427051	5,3357E-05	5,34E-05
31	0,00000000053	23,999999999984	2,2083E-11	1,54E-11
31	0,00000000004	23,999999999995	1,6667E-12	6,66E-12

The graph below resumes the effect of computation with 15 and 25 significant digits.



The application field of multi-precision computation is wide. Overall it is very useful during the testing of numeric algorithms. In the above example, we had not doubt about the Newton-Raphson method, but what about the new algorithm that you are studying? This package helps you in this work.

## ***Multiprecision methods***

Several methods exist for simulating variable multi-precision floating point arithmetic. The basic concept consists of breaking down a long number into two or more sub-numbers, and repeating cyclic operations with them. The ways in which long numbers are stored vary from one method to another. The two most popular methods use the "string" conversion and the "packing"

## ***How to store long number***

### **String Extended Numbers**

In this method, long numbers are stored as vectors of characters, each representing a digit in base 256. Input numbers are converted from decimal to 256 base and vice versa for output. All internal computations are in 256 base. this requires only 16 bit for storing and a 32 bit accumulator for computing. Here is an example of how to convert the number 456789 into string

$$(456789)_{10} \equiv (6, 248, 85)_{256}$$

String = chr(6)&chr(248)&chr(85)

This method is very fast, and efficient algorithms for the input-output conversion have been realized. A good explanation of this method can be found in "NUMERICAL RECIPES in C - The Art of Scientific Computing", Cambridge University Press, 1992, pp. 920-928. In this excellent work you can also find efficient routines and functions to implement an arbitrary-precision arithmetic.

Perhaps the most critical factor of this method is the debug and test activity. It will be true that the computer does not care about the base representation of numbers, but programmers usually do it. During debugging, programmers examine lots and lots of intermediate results, and they must always translate them from base 256 to 10. For this kind of programs, the debugging and tuning activity usually takes 80 - 90% of the total develop time.

### **Packet Extended Numbers**

This method avoids converting the base representation of long numbers and stores them as vectors of integers. This is adopted in all FORTRAN77 routines of "MPFUN: A MULTIPLE PRECISION FLOATING POINT COMPUTATION PACKAGE" by NASA Ames Research Center. For further details we remand to the refined work of David H. Bailey published in "TRANSACTIONS ON MATHEMATICAL SOFTWARE", Vol. 19, No. 3, SEPTEMBER, 1993, pp. 286-317.

Of course this add-in does not have the performance of the mainframe package (16 million digits) but the method is substantially the same. Long numbers are divided into packets of 6 or 7 digits.

For example, the number 601105112456789 in packet form of 6 digits becomes the following integer vector:

456789
105112
601

As we can see, the sub-packet numbers are in decimal base and the original long number is perfectly recognizable. This a great advantage for the future debugging operation.

An example of arithmetic operation - the multiplication  $A \times B = C$  - between two packet numbers is shown in the following:

## Xnumbers Tutorial

A		B
456789		654321
105112	x	
601		

The schema below illustrates the algorithm adopted:

carry		A		B		C'		C
0	+	456789	x	654321	=	298886635269	=>	635269
298886	+	105112	x	654321	=	68777287838	=>	287838
68777	+	601	x	654321	=	393315697	=>	315697
393	+	0	x	654321	=	393	=>	393

The numbers in the accumulator C' are split into two numbers. The last 6 digits are stored in C, the remaining left digits are copied into the carry register of the next row. As we can see, the maximum number of digits is reached in accumulator C'. In the other vectors, the numbers require only six digits at most. The maximum number of digits for a single packet depends of the hardware accumulator. Normally, for a 32-system, is 6 digits.. This is equivalent to conversion from a decimal to a  $10^6$  representation base. This value is not critical at all. Values from 4 to 7 affect the computation speed of about 30 %. But it does not affect the precision of the results in any case.

## Functions

### General Description

Xnumbers is an Excel addin (xla) that performs multi-precision floating point arithmetic. Perhaps the first package providing functions for Excel with precision from 15 up to 200 significant digits. It is compatible with Excel XP and consists of a set of more than 270 functions for arithmetic, complex, trigonometric, logarithmic, exponential and matrix calculus covering the following main subjects.

The basic arithmetical functions: addition, multiplication, and division were developed at the first. They form the basic kernel for all other functions.

All functions perform multiprecision floating point computations for up to 200 significant digits. You can set a precision level separately for each function by an optional parameter. By default, all functions use the precision of 30 digits, but the numerical precision can easily be regulated continually from 1 to 200 significant digits. In advance some useful constants like  $\pi$ ,  $\text{Log}(2)$ ,  $\text{Log}(10)$  are provided with up to 400 digits.

### Using Xnumbers functions

These functions can be used in an Excel worksheet as any other built-in function. After the installation, look up in the functions library or click on the icon



Upon "user's" category you will find the functions of this package.

From version 2.0 you can manage functions also by the **Function Handbook**. It starts by the Xnumbers menu



All the functions for multi-precision computation begin with "x". The example below shows two basic functions for the addition and subtraction.

	A	B
2	123456789,123456	
3	0,0123456789	
4	123456789,1358020000	=A2+A3
5	123456789,1358016789	=xadd(A2,A3)
6	123456789,1111100000	=A2-A3
7	123456789,1111103211	=xsub(A2,A3)
8		

As any other functions they can also be nested to build complex expressions. In the example below we compute  $x^4$  with 30 digits precision

	A	B
2	1234567	
3	2323050529221950000000000	=A2^4
4	2323050529221952581345121	=xmult(A2;xmult(A2;xmult(A2,A2)))
5		

## Using extended numbers in Excel



If you try to enter a long number with more than 15 digits in a worksheet cell, Excel automatically converts it in standard precision eliminating the extra digits. The only way to preserve the accuracy is to convert the number in a string. It can be done by prefixing it with the hyphen symbol ' '.

This symbol is invisible in a cell but avoid the conversion.

Example: enter in a cell the number 1234567890123456789.

Arial		10	G C S			
B2			=1234567890123456789			
	A	B	C			
1						
2		1234567890123456789				
3		1.23457E+18				
4						

We have inserted the same number with the hyphen in B2 and without the hyphen in B3. Excel treats the first number as a string and the second as a numbers  
Note also the different alignment

B4			=B2*2			
	A	B	C			
1						
2		1234567890123456789				
3		1.23457E+18				
4		2.46914E+18				
5						

We have inserted a long numbers with full precision as a string in B2  
If we try to multiply the cell B2 for another number, example for 2, Excel converts the string into number before performing the multiplication. In this way the originally accuracy is destroyed

B5			=xmult(B2;2)			
	A	B				
1						
2		1234567890123456789				
3		1.23457E+18				
4		2.46914E+18				
5		2469135780246913578				
6						

The only way to perform arithmetic operations preserving the precision is to use the multiprecision functions of the Xnumbers library.  
In that case we use the function xmult  
Note from the alignment that the result is still a string

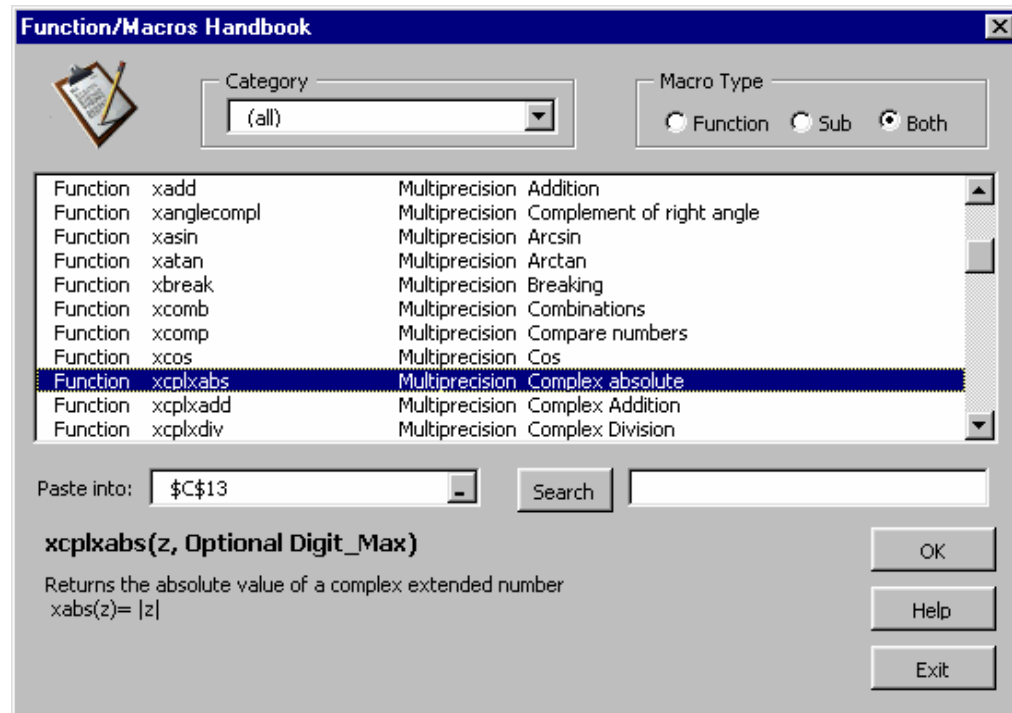
You can also insert extended numbers directly in the function. Only remember that, for preserving Excel to convert them, you must insert extended numbers like string, within quote "...".

```
2469135780246913578 =xmult( "1234567890123456789" , 2 )
```

## Functions Handbook



Xnumbers includes a new application for searching and pasting the Xnumbers functions, that are cataloged by subject. This feature (born to overcome the poor standard Excel function wizard) can also submit the Xnumbers macros. You can activate the Functions Handbook from the menu bar "Help > Function manager".



**Category:** you can filter macros by category (Arithmetic, Statistical, Trigonometric, etc.)

**Macro Type:** filters by macro Functions, by macro Subroutines, or both

**Paste Into:** choose the cell you want to paste a function, default is the active cell

**Search:** searches macros by words or sub-words contained into the name or description. For example, if you input "div" you list all macros that match words like (div, divisor, division,...)

You can associate more words in AND/OR. Separate words with comma "," for OR, with plus "+", for AND. For example, if you type "+div +multi" you will get all the rows containing words like (div, divisor, division,...) and words like (multi, multiprecision,...). On the contrary, if you type "div, multi", you get all the rows that contain words like (div, divisor, division,...) or also the words like (multi, multiprecision,...). Remember to choose also the Category and Macro Type. Example, if you enter the word "hyperbolic", setting the Category "complex", you find the hyperbolic functions restricted to the complex category.

**Help:** recalls the help-on-line for the selected function.

**OK:** insert the selected function into the worksheet ". This activates the standard Excel function wizard panel. If the macro selected is a "sub", the OK button activates the macro.



### **Precision**

Most functions of this package have an optional parameter - **Digit\_Max** - setting the maximum number of significant digits for floating point computation, from 1 to 200 (default is 30). The default can be changed from the menu X-Edit\Default Digits

This parameter also determines how the output is automatically formatted. If the result has fewer integer digits than Digit\_Max, then the output is in the plain decimal format ( 123.45, -0.0002364, 4000, etc.), otherwise, if the number of integer digits exceeds the maximum number of digits allowed (significant or not), the output is automatically converted in exponential format (1.23456789E+94).

The exponent can reach the extreme values of +/- 2,147,483,647.

The output format is independent of the input format.

In synthesis, the Digit\_Max parameter limits:

The significant digits of internal floating point computation

The maximum number output digits, significant or not.

The default of Digit\_Max can be changed from the *X-Edit* menu . This affects any multiprecision function and macro.

### **Formatting Result**

The user can not format an extended number with standard Excel number format tools, because, it is a string for Excel. You can only change the alignment. To change it you can use the usual standard Excel format tools.



It is possible to separate the digits of a x-numbers in groups, by the user function **xFormat()** and **xUnformat()** <sup>1</sup>.

It work similar at the built-in function Format(x, "#,##0.00")

2,469,135,780,246,913,578 = xformat("2469135780246913578",3)

---

<sup>1</sup> These functions were original developed by Ton Jeursen for the add-in XNUMBER95, the downgrade version of XNUMBERS for Excel 5. Because they are very useful for examining long string of number, we have imported them in this package

## Arithmetic Functions

### Addition

**xadd(a, b, [Digit\_Max])**

Performs the addition of two extended numbers:  $\text{xadd}(a, b) = a + b$ .

	A	B
2	123456789,123456	
3	0,0123456789	
4	123456789,1358020000	=A2+A3
5	123456789,1358016789	=xadd(A2,A3)
6	123456789,1111100000	=A2-A3
7	123456789,1111103211	=xsub(A2,A3)
8		

### Subtraction

**xsub(a, b, [Digit\_Max])**

Performs the subtraction of two extended numbers:  $\text{xsub}(a, b) = a - b$ .

NB. Do not use the operation  $\text{xadd}(a, -b)$  if “b” is an extended number. Excel converts “b” into double, then changes its sign, and finally calls the xadd routine. By this time the original precision of “b” is lost. If you want to change sign at an extended number and preserve its precision use the function **xneg()**

### Accuracy lack by subtraction

The subtraction is a critical operation from the point of view of numeric calculus. When the operands are very near each others, this operation can cause a lack of accuracy. Of course this can happen for addition when the operands are near and have opposite signs. Let's see this example

Assume one performs the following subtraction where the first operand has a precision of 30 significant digits

800000.008209750361424423316366	(digits)
800000	30
0.008209750361424423316366	6
	25

The subtraction is exact (no approximation has been entered). But the final result have 25 total digits, of which only 22 are significant. 8 significant digits are lost in this subtraction. We cannot do anything about this phenomenon, except to increase the precision of the operands, when possible.

## Multiplication

---

### xmult(a, b, [Digit\_Max])

Performs the multiplication of two extended numbers:  $\text{xmult}(a, b) = a \times b$ .

The product can often lead to long extended numbers. If the result has more integer digits than the ones set by Digit\_Max, then the function automatically converts the result into exponential format.

	A	B	C	D
1	<b>Digits Max</b>	<b>x</b>		
2	30	831402	831402	=B2
3		1339481	1113647182362	=xmult(B3;C2;\$A\$2)
4		291720	324873156038642640	=xmult(B4;C3;\$A\$2)
5		1650649	536251550142029435073360	=xmult(B5;C4;\$A\$2)
6		255255	136880889431503723449650506800	=xmult(B6;C5;\$A\$2)
7		1205776	1.65047691335160833646225789487E+35	=xmult(B7;C6;\$A\$2)
8		2387242	3.94008780758332018835283346146E+41	=xmult(B8;C7;\$A\$2)

## Division

---

### xdiv(a, b, [Digit\_Max])

Performs the division of two extended numbers:  $\text{xdiv}(a, b) = a / b$ .

If  $b = 0$  the function returns "?". The division can return long extended numbers even when the operands are small. In the example below we see the well-known periodic division  $1 / 7 = 0,142857 \dots$ . Excel breaks the results after 15 digits, while the xdiv shows up to 30 digit

	A	B	C
12		1	
13		7	
14		0,142857142857143000000000000000	=B12 / B13
15		0,142857142857142857142857142857	=xdiv(B12; B13)
16			

## Inverse

---

### xinv(x, [Digit\_Max])

It returns the inverse of an extended number

$$\text{xinv}(x) = 1 / x$$

If  $x = 0$ , the function returns "?".

## Integer Division

---

### **xdivint(a, b, [Digit\_Max])**

Returns the quotient of the integer division:  $\text{xdivint}(a, b) = \text{INT}(a / b)$ ,  
If  $b = 0$  the function returns “?”.

$$a = b \cdot q + r, \text{ with } 0 < r < b$$

$$\text{xdivint}(a, b) = q$$

## Integer Remainder

---

### **xdivrem(a, b, [Digit\_Max])**

Returns the remainder of the integer division:  
If  $b = 0$  the function returns “?”.

$$a = b \cdot q + r, \text{ with } 0 < r < b$$

$$\text{xdivrem}(a, b) = r$$

### **How to test multiprecision functions ?**

This test is the most important problem in developing multiprecision arithmetic. This activity, absorbs almost the 60% of the totally realization effort.

Apart the first immediate random tests, we can use many known formulas and algorithms. The general selecting criterions are:

1. Formulas should be iterative
2. Formulas should have many arithmetic elementary operations
3. Final results should be easily verified
4. Intermediate results should be easily verified
5. Algorithms should be stable
6. Efficiency is not important

For example, a good arithmetic test is the Newton algorithm to compute the square root of a number. The iterative formula:

$$x_{n+1} = \frac{x_n}{2} + \frac{1}{x_n} = \frac{x_n \cdot x_n + 2}{2 \cdot x_n}$$

converges to  $2^{1/2}$ , starting from  $x_0 = 1$ .

We have rearranged the formula in order to increase the number of operations (remember: the efficiency is not important). In this way we can test multiplication, division and addition.

$$x_0 = 1$$

$$x_1 = 1.5$$

$$x_2 = 1.41...$$

$$x_3 = 1.41421.....$$

Look at a possible Excel arrangement. We have limited the number of the significant digits to 100 only for the picture dimensions, but there is no difficult to repeat it with the maximum digits.

For each iterate only the blu digits are exacts. We see the progressive convergence. By the way, we note that this algorithm is also very efficient. The rate of convergence is quadratic. The number of digits approximately doubles at each iteration (In fact this is just the algorithm used by the xsqr multiprecision function)  
But, as said, for testing, the efficiency has no influence. It is important only that the algorithm involves the most multiprecision functions as possible.

## Initialize

## Iteration

$$Y = Y/X$$

$$T = 2T$$

Accuracy: approximately 12 decimal digits every 5 iterations)

Below, step by step, a possible Excel arrangement:

The Digit\_Max parameter is in the A1 cell. By this parameter we can modulate the arithmetic accuracy. We have set 30 digits only for the picture dimensions. But you can try with 60, 100 or more.

---

28

## Xnumbers Tutorial

Note that, in order to have a more compact form, we have used the **xeval** function for calculating the X and P formulas that are inserted into the cells B3 and E3 respectively. Selecting the last row (range A6:F6) and dragging it down, we get the following iteration table

	A	B	C	D	E	F
1	30	<=Digit_Max				
2						
3		$(2+x)^{(1/2)}$			$y*t*(5*y^6/122+3*y^4/40+y^2/6+1)$	
4	i	X	Y	T	~P	DP
5	0	1.7320508075688772935274463415	0.5	6	3.14098360655737704918032786883	
6	1	1.93185165257813657349948639945	0.258819045102520762348898837625	12	3.14158723844893787167289449156	6.036E-04
7	2	1.98288972274762082228911505384	0.130526192220051591548406227897	24	3.14159258878164217158215056614	5.350E-06
8	3	1.99571784647720701347613958253	6.54031292301430668153155587764E-2	48	3.14159265265862780660300654045	6.388E-08
9	4	1.99892917495273128885967289247	3.27190828217761420636599263181E-2	96	3.1415926535755661052797423287	9.169E-10
10	5	1.9997322758191235657254942981	1.63617316264867816429719234847E-2	192	3.14159265358957220264154960473	1.401E-11
11	6	1.99993306783480220691520762115	8.1811396039371292851991232949E-3	384	3.14159265358978978971601742189	2.176E-13
12	7	1.99998326688870129829511724112	4.09060402623478959462105417169E-3	768	3.14159265358979318459527058918	3.395E-15
13	8	1.99999581671780036208332744864	2.04530629116409511441305892323E-3	1536	3.14159265358979323762104105147	5.303E-17
14	9	1.9999989541791766552219647492	1.02265368033830454119296114608E-3	3072	3.14159265358979323844949364137	8.285E-19
15	10	1.99999973854477707409715031033	5.11326907013697501235819349761E-4	6144	3.14159265358979323846243791986	1.294E-20
16	11	1.9999999346361932004174774429	2.55663461862417314061649520852E-4	12288	3.14159265358979323846264017305	2.023E-22
17	12	1.9999999836590482334769327605	1.27831731975654740261724508019E-4	24576	3.14159265358979323846264333326	3.160E-24
18						
19						

The convergence to pi greek is evident.

## Sum

### xsum(v, [Digit\_Max])

This is the extended version of the Excel built-in function SUM. It returns the sum of a vector of numbers. The argument is a standard range of cells.

$$\sum_i v_i = v_1 + v_2 + \dots v_n$$

Note that you can not use the standard function SUM, because it recognizes extended numbers as strings and it excludes them from the calculus.

	A	B	C
1			
2		3.14159265358979323846264338327	
3		1.33333333333333333333333333333333	
4		4.18879020478639098461685784434	
5		9.45655	
6			
7	Σ =	18.1202661917095175564128345609	=xsum(B2:B5)
8			

## Product

### xprod(v, [Digit\_Max])

Returns the product of a vector of numbers.

$$\prod_i v_i = v_1 \cdot v_2 \cdot \dots \cdot v_n$$

A	B	C	D
	8314		
	133941		
	29172		
	16506		
	25525		
	12057		
	2387		
		=xprod(B1:B7)	
$\Pi =$	393902768814756765393608578800		

Note that the result is an extended number even if all the factors are in standard precision

## Raise to power

---

### xpow(x, n, [Digit\_Max])

Returns the integer power of an extended number.  $\text{xpow}(x, n) = x^n$

$\text{xpow}("0.39155749636098981077147016011", 90) = 1.9904508921478176508981155284\text{E-}7$

$\text{xpow}(5, 81, 60) = 5^{81} = 413590306276513837435704346034981426782906055450439453125$

## Square Root

---

### xsqr(x, [Digit\_Max])

Returns the square root of an extended number  $\text{xsqr}(x) = \sqrt{x}$

The example below shows how to compute the  $\sqrt{2}$  with 30 and 60 significant digits:

$\text{xsqr}(2) = 1.41421356237309504880168872420969807$

$\text{xsqr}(2, 60) = 1.41421356237309504880168872420969807856967187537694807317667973799$

## N<sup>th</sup>- Root

---

### xroot(x, n, [Digit\_Max])

Returns the n<sup>th</sup> root of an extended number  $\text{xroot}(x, n) = \sqrt[n]{x}$

The root's index must be a positive integer.

The example below shows how to compute the  $\sqrt[3]{100}$  with 30 and 60 significant digits:

$\text{xroot}(100, 9) = 1.66810053720005875359979114908$

$\text{xroot}(100, 9, 60) = 1.66810053720005875359979114908865584747919268415239470704499$

## Absolute

---

### xabs(x)

Returns the absolute value of an extended number  $xabs(x) = |x|$

Do not use the built-in function "abs", as Excel converts x in double, then takes the absolute value. By that time the original precision of x is lost.

## Change sign

---

### xneg(x)

Returns the opposite of an extended number:  $xneg(x) = -x$

Do not use the operator “-” (minus) for extended numbers. Otherwise Excel converts the extended numbers into double and, afterwards, changes its sign. By that time the original precision is lost. In the following example the cell B8 contains an extended number with 18 digits. If you use the “-” as in the cell B9, you lose the last 3 digits. The function xneg(), as we can see in the cell B10, preserves the original precision.

	A	B	C
7			
8		123456789,123456789	
9		-123456789,123456000	=-B8
10		-123456789,123456789	=xneg(B8)
11			

## Integer part

---

### xint(x)

Returns the integer part of an extended number, thus the greatest integer less than or equal to x.

Examples:

```
xint(2.99) = 2
xint(2.14) = 2
xint(-2.14) = -3
xint(-2.99) = -3
xint(12345675.00000001) = 12345675
xint(-12345675.00000001) = -12345676
```

## Decimal part

---

### xdec(x)

Returns the decimal part of an extended number

Examples:



## Xnumbers Tutorial

```
xdec(2.99) = 0.99  
xdec(-2.14) = - 0.14
```

## Truncating

---

### xtrunc(x)

Eliminates the decimal part of an extended number.

Examples:

```
xtrunc(2.99) = 2  
xtrunc(2.14) = 2  
xtrunc(-2.14) = -2  
xtrunc(-2.99) = -2  
xtrunc(12345675.00000001) = 12345675  
xtrunc(-12345675.00000001) = -12345675
```

If  $x > 0$  this function returns the same value of `xInt()`

## Rounding

---

### =xround(x, [dec])

Rounds an extended number, the parameter "*dec*" sets the decimal number of is to keep (default 0). It works like standard round function. "*dec*" can be negative, in that case  $x$  is rounded to the integer number, starting to count back from decimal point. See the following examples.

number to round	dec	number rounded
6.2831853071795864769	0	6
6.2831853071795864769	1	6.3
6.2831853071795864769	2	6.28
6.2831853071795864769	3	6.283
6.2831853071795864769	4	6.2832
100352934.23345	0	100352934
100352934.23345	-1	100352930
100352934.23345	-2	100352900

When the number is in exponential format, it is internally converted into decimal before the rounding.

number to round	Decimal format	Dec	number rounded
1.238521E-17	0.0000000000000001238521	16	0
1.238521E-17	0.0000000000000001238521	17	1.E-17
1.238521E-17	0.0000000000000001238521	18	1.2E-17
1.238521E-17	0.0000000000000001238521	19	1.24E-17

## Relative Rounding

---

### =xroundr(x, [dgt])

Returns the relative round of a number. The optional parameter Dec sets the significant digits to keep. (default = 15)  
 This function always rounds the decimal place no matter what the exponent is

number to round	dgt	number rounded
1.23423311238765E+44	15	1.23423311238765E+44
1.23423311238765E+44	14	1.2342331123876E+44
1.23423311238765E+44	13	1.234233112388E+44
1.23423311238765E+44	12	1.23423311239E+44
1.23423311238765E+44	11	1.2342331124E+44
1.23423311238765E+44	10	1.234233112E+44

## Extended Numbers manipulation

## Digits count

**xdgt(x)**

Returns the number of digits, significant or not, of an extended number. It is useful for counting the digits of long numbers

[illegible]

### Significant Digits count

**xdgts(x)**

Returns the number of significant digits of a number, assuming that trailing zeros are not significant

```
xdgts("1240100000") = 5
```

## Compare numbers

**xcomp(a [b])**

Compares two extended numbers. It returns the value  $y$  defined by:

$$y = \begin{cases} 1 \Rightarrow a > b \\ 0 \Rightarrow a = b \\ -1 \Rightarrow a < b \end{cases}$$

The number  $b$  is optional (default  $b=0$ )

If the second argument is omitted, the function returns the **sign(a)**

```
xcomp(300, 299) = 1
xcomp(298, 299) = -1
xcomp(300, 300) = 0
```

if b is missing, then  $b = 0$  for default and we get the  $\text{sign}(a)$

```
xcomp(3.58E-12)= 1
xcomp(0)= 0
xcomp(-0.0023)= -1
```

### Extended Numbe Check

---

#### isXnumbers(x)

Returns TRUE if x is a true extended number.

That is, x cannot be converted into double precision without lost of significant digits. It happens if a number has more than 15 significant digits.

```
isXnubers(1641915798169656809371) = TRUE  
isXnubers(1200000000000000000000) = FALSE
```

### Format Extended Number

---

#### =xFormat(str, [Digit\_Sep])

#### =xUnformat(str)

This function<sup>3</sup> separates an extended number in groups of digits by the separation character of you local system ( e.g. a comma "," for USA, a dot "." for Italy). Parameter "str" is the string number to format, Digit\_Sep sets the group of digits ( 0 means no format)

The second function removes any separator character from the string

Example (on Italian standard):

```
x = 1230000012,00002345678  
xFormat(x,3) = 1.230.000.012,000.023.456.79  
xFormat(x,6) = 1230.000012,000023.45679
```

Example (on USA standard):

```
xFormat(x,3)= 1,230,000,012.000,023,456,78  
xFormat(x,6)= 1230,000012.000023,45678
```

---

<sup>3</sup> These functions were original developed by Ton Jeursen for his add-in XNUMBER95, the downgrade version of XNUMBERS for Excel 5.  
Because it works well and it is very useful for examining long string of number, I have imported it in this package.

## Check digits

### DigitsAllDiff(number)

This function<sup>4</sup> return TRUE if a number has all digits different.

DigitsAllDiff(12345) = TRUE

DigitsAllDiff(123452) = FALSE

Argument can be also a string. Example

DigitsAllDiff(12345ABCDEFHIM) = TRUE

DigitsAllDiff(ABCDA) = FALSE

## SortRange

### =SortRange (ArrayToSort, [IndexCol], [Order], [CaseSensitive])

This function returns an array sorted along a specified column

**ArrayToSort:** is the (N x M ) array to sort

**IndexCol:** is the index column for sorting (1 default)

**Order:** can be "A" ascending (default) or "D" descending

**CaseSensitive:** True (default) or False. It is useful for alphanumeric string sorting

Example: The left table contains same points of a function f(x,y). The right table is ordered from low to high function values (the 3-th column)

	A	B	C	D	E	F	G	H	I
1	Xi	Yi	f(x,y)		Index		Xi	Yi	f(x,y)
2	0.162	2.7	97		3		0.057	38.37	61
3	0.519	-10.8	111		Sort		0.157	36.923	63
4	0.417	20.1	80		a		0.417	20.091	80
5	0.157	36.9	63				0.162	2.737	97
6	0.057	38.4	61				0.519	-10.81	111
7	0.602	-46.7	147				0.972	-29.95	131
8	0.972	-29.9	131				0.602	-46.72	147
9									

## Digits sum

### sumDigits(number)

This useful<sup>5</sup> function returns the digits sum of an integer number (extended or not)

sumDigits(1234569888674326778876543) = 137

<sup>4</sup> This function appears by the courtesy of Richard Huxtable

<sup>5</sup> This function appears by the courtesy of Richard Huxtable

## Vector Inversion

---

### Flip(v)

This function returns a vector in inverse order  $[a_1, a_2, a_3, a_4] \Rightarrow [a_4, a_3, a_2, a_1]$

	A	B	C		F	G	H	I	J
1					degree	coef		degree	coef
2		{=flip(A4:A8)}			0	112345		4	1
3					1	-2345		3	8
4	123		100		2	-124		2	-124
5	44		1		3	8		1	-2345
6	-34		-34		4	1		0	112345
7	1		44						
8	100		123						
9									

## Scientific Format

---

### xcvexp(mantissa, [exponent])

This function converts a number into scientific format. Useful for extended numbers that, being string, Excel cannot format.

```
xcvexp(-6.364758987642234, 934) = -6.364758987642234E+934
```

```
xcvexp(1.2334567890122786, ) = 1.2334567890122786E-807
```

This function is useful also to convert any xnumbers into scientific notation, simply setting exponent = 0 (default)

```
xcvexp(12342330100876523, 0) = 1.2342330100876523E+16
```

```
xcvexp(0.000023494756398348) = 2.3494756398348E-5
```

## Split scientific format

---

### xspllit(x)

This function returns an array containing the mantissa and exponent of a scientific notation.

If you press Enter this function returns only the mantissa. If you select two cells and give the CTRL+SHIFT+ENTER sequence, you get both mantissa and exponent

```
xspllit( 2.3494756398348E-5 ) = { 2.3494756398348 , -5 }
```

```
xspllit( -1.233456E-807 ) = { -1.2334567890122786 , -807 }
```

Note that, in the last case, you cannot convert directly into double (for example, using the VALUE function), even if the number of digits is less than 15. The exponent is too large for the standard double precision.

## Convert Extended Number

## **=xcdbl(str)**

This function converts an extended number into standard double precision. It can be nested with other functions and/or array-functions.

Usually the extended numbers are too long for a compact visualization. So, after, the multiprecision computation, we would like to convert the result in a compact standard precision.

For example, if you invert the matrix

1	1	2
4	5	3
-2	1	5

using the multiprecision **xMatInv** function, you will get a matrix like the following

0.91666666666666666666	-0.12499999999999999999	-0.29166666666666666666
-1.08333333333333333333	0.375	0.20833333333333333333
0.58333333333333333333	-0.125	4.166666666666666666E-2

If you use the functions **xcdbl** nested with the multiprecision function, the matrix will be rounded in standard precision and the result will have a more compact format

	A	B	C	D	E	F	G
1							
2	1	1	2		0.91667	-0.125	-0.2917
3	4	5	3		-1.0833	0.375	0.20833
4	-2	1	5		0.58333	-0.125	0.04167
5							
6							
7							



`{=xcdbl(xMatInv(A2:C4))}`

## Macros X-Edit

These simple macros are very useful for manipulating extended numbers in the Excel worksheet. They perform the following operations:

<b>Format</b>	Separates groups of digits
<b>Unformat</b>	Removes the separation character
<b>Double Conversion</b>	Converts multiprecision numbers into standard double precision
<b>Round</b>	Rounding multiprecision numbers
<b>Relative Round</b>	Relative rounding multiprecision numbers
<b>Mop-Up</b>	Converts small numbers into 0

From this menu you can also change the default **Digit\_Max** parameter  
Using these macros is very simple. Select the range where you want to operate and then start the relative macro. They work only over cells containing only numeric values, extended or standard. Cells containing function are ignored

**Tip.** For stripping-out a formula from a cell and leaving its value, you can select the cell and then click in sequence   (copy + paste values)

Here are some little examples:

### Format - group 6

31415926.53589793238462643		31,415926.535897,932384,62643
19831415926.53589793238462	⇒	19831,415926.535897,932384,62
0.535897932384626433832734		0.535897,932384,626433,832734

### Double Conversion

31415926.53589793238462643		31415926.5358979
19831415926.53589793238462	⇒	19831415926.5358
0.535897932384626433832734		0.535897932384626

### Rounding 3 decimals.

31415926.53589793238462643		31415926.536
19831415926.53589793238462	⇒	19831415926.536
0.535897932384626433832734		0.536

### Relative rounding - significant digits 15.

4.5399929762484851535591E-5		4.53999297624849E-05
1.0015629762484851535591E-6	⇒	1.00156297624849E-06
0.539929762484851535591E-12		5.39929762484852E-13



## Xnumbers Tutorial

### Mop-Up - Error limit 1E-15.

31415926.53589793238462643		31415926.53589793238462643	
-1.00E-15	⇒		0
5.78E-16			0
-1.40E-18			0

Note that the function mopup is used overall for improving the readability. The cells having values greater than the limit are not modified.

## Statistical Functions

### Factorial

---

#### **xfact(n, [Digit\_Max])**

Returns the factorial of an integer number  $\text{xfact}(n) = n!$

This example shows all 99 digits of 69!

```
xfact(69, 100) = 711224524281413113724683388812728390922705448935203693936480
40923257279754140647424000000000000000
```

If the parameter Digit\_Max is less than 99, the function returns the approximate result in exponential format:

```
xfact(69) = 1.71122452428141311372468338881E+98
```

For large number ( $n \gg 1000$ ) you can use the faster function xGamma(x). The relation between the factorial and the gamma function is:

$$\Gamma(n) = (n-1)!$$

### Factorial with double-step

---

#### **xfact2(n, [Digit\_Max])**

Returns the factorial with double step.

if n is odd  $\Rightarrow \text{xfact2}(n) = 1 \cdot 3 \cdot 5 \cdot 7 \cdot 9 \dots n$

if n is even  $\Rightarrow \text{xfact2}(n) = 2 \cdot 4 \cdot 6 \cdot 8 \dots n$

Note: In many books, this function is indicated improperly as "double factorial", or - even worse - with the confusing symbol "!!".

### Combinations

---

#### **xComb(n, k, [Digit\_Max])**

Returns the binomial coefficients, a combination of n, class k.  $\text{xcomb} = C_{n,k}$

The example below shows all the 29 digits of the combination of 100 objects grouped in class of 49 elements:

```
xComb(100, 49) = 98913082887808032681188722800
```

## Xnumbers Tutorial

	A	B	C	D
1	<b>N</b>	<b>K</b>	<b>Combinations</b>	<b>digits</b>
2	100	10	17310309456440	14
3	100	20	535983370403809682970	21
4	100	30	29372339821610944823963760	26
5	100	40	13746234145802811501267369720	29
6	100	50	100891344545564193334812497256	30
7	100	60	13746234145802811501267369720	29
8	100	70	29372339821610944823963760	26
9	100	80	535983370403809682970	21
10	100	90	17310309456440	14
11				
12			=xcomb(A10;B10)	=xdgts(C10)

Combinations of N = 100 objects in class of 10, 20, ... 90

Note the typical parabolic outline of the binomial coefficients

For large argument (n and k >>1000) use the faster function xcomb\_big(n,k) .

## Permutations

### xPerm(n, [k], [Digit\_Max])

Returns the permutation of n, class k. xperm(n,k)=  $P_{n,k}$ .

If k is omitted, the function assume k = n and in this case will be  $P(n) = n!$

Examples:

```
xPerm(100, 20, 60) = 1303995018204712451095685346159820800000
```

```
xPerm(100) = 9.33262154439441526816992388562E+157
```

## Arithmetic Mean

### xmean(x, [Digit\_Max])

Returns the arithmetic mean of n numbers, extended or not. The argument is a range of cells.

$$m = \frac{\sum_{i=1}^n x_i}{n}$$

## Geometric Mean

### xgmean(x, [Digit\_Max])

Returns the geometric mean of n numbers, extended or not.

$$GM = \sqrt[n]{x_1 x_2 x_3 \dots x_n}$$

## Quadratic Mean

---

### **xqmean(x, [Digit\_Max])**

Returns the quadratic mean of n numbers, extended or not.

$$QM = \sqrt{\frac{\sum x^2}{n}}$$

## Standard Deviation

---

### **xstdev(x, [Digit\_Max])**

Returns the standard deviation of n numbers, extended or not.

$$\sigma = \sqrt{\frac{n \sum x^2 - (\sum x)^2}{n^2}}$$

## Variance

---

### **xvar(x, [Digit\_Max])**

Returns the variance of n numbers, extended or not.

$$v = \frac{n \sum x^2 - (\sum x)^2}{n^2}$$

## Linear Regression Coefficients

**xRegLin\_Coeff( Y, X, [DgtMax], [Intcpt])**

**RegLin\_Coeff( Y, X , [Intcpt] )**

Computes the multivariate linear regression with the least squares method in multi-precision.

Parameter Y is a vector (n x 1) of dependent variable.

Parameter X is a list of the independent variable. It may be an (n x 1) vector for monovariate regression or a (n x m) matrix for multivariate regression.

Parameter Intcpt, if present, forces the Y intercept:  $Y(0) = \text{Intcpt}$

The function returns the coefficients of linear regression function. For monovariate regression, it returns two coefficients  $[a_0, a_1]$ , the first one is the intercept of Y axis, the second one is the slope.

For standard precision use the faster RegLin\_Coef

### Simple Linear Regression

Example. Evaluate the linear regression for the following xy data table

x	y
0.1	1991
0.2	1991.001046
0.35	1991.001831
0.4	1991.002092
0.45	1991.002354
0.6	1991.003138
0.7	1991.003661
0.8	1991.004184
0.9	1991.004707
1	1991.00523
1.5	1991.007845
1.8	1991.009414
2	1991.01046
3	1991.01569

The model for this data set is

$$y = a_0 + a_1 x$$

Where  $[a_0, a_1]$  are the unknown coefficients that can be evaluate by the **xRegLin\_Coeff** function

We can also compute the factor  $r^2$  in order to measure the goodness of the regression

This can be done by the **xRegLinStat** function

	A	B	C	D
1	<b>x</b>	<b>y</b>		<b>Coefficients</b>
2	0.1	1991	<b>a0 =</b>	1990.9999102920727213168454672
3	0.2	1991.001046	<b>a1 =</b>	0.00528310949144214233068544754729
4	0.35	1991.001831		<b>{=xRegLin_Coeff(B2:B15;A2:A15)}</b>
5	0.4	1991.002092		
6	0.45	1991.002354		
7	0.6	1991.003138		
8	0.7	1991.003661		<b>Regression factor r^2</b>
9	0.8	1991.004184		0.999058626726373012818388072584
10	0.9	1991.004707		<b>=xRegLin_R2(B2:B15;A2:A15;D2:D3)</b>
11	1	1991.00523		
12	1.5	1991.007845		
13	1.8	1991.009414		
14	2	1991.01046		
15	3	1991.01569		

### Multivariate Regression

This function can also compute a multivariate regression. This is when y depends by several variables  $x_1, x_2, \dots, x_n$ . Look at this example

x1	x2	x3	y
0	0	-4	4000.8
0.1	0	-2	4000.7
0.2	0.5	-1	4001.55
0.3	0.5	0	4001.65
0.4	1	1.5	4002.4
0.5	1	2	4002.59

The model for this data set is

$$y = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3$$

Where  $[a_0, a_1, a_2, a_3]$  are the unknown coefficients

	A	B	C	D	E	F	G
1	x1	x2	x3	y			Coefficients
2	0	0	-4	4000.8		a0 =	4000.02448275862068965517241379
3	0.1	0	-2	4000.7		a1 =	2.89425287356321839080459770115
4	0.2	0.5	-1	4001.55		a2 =	1.50781609195402298850574712644
5	0.3	0.5	0	4001.65		a3 =	-0.19379310344827586206896551724
6	0.4	1	1.5	4002.4			
7	0.5	1	2	4002.59		r^2 =	0.999994016818349670223855132214
8							
9							
10							
11							

{=xRegLin\_Coeff(D2:D7;A2:C7)}

xRegLin\_R2(D2:D7;A2:C7;G2:G5)

### Polynomial Regression

The same algorithm for finding the linear regression can easily be adapted to the polynomial regression. In the example below we will find the best fitting polynomial of 3<sup>rd</sup> degree for the given data

x	y
10	1120
11	1473
12	1894
13	2389
14	2964
15	3625
16	4378
17	5229
18	6184
19	7249
20	8430

The model for this data set is

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

where  $[a_0, a_1, a_2, a_3]$  are the unknown coefficients

First of all we add at the given table two extra columns containing the power  $x^2$ ,  $x^3$ . They can easily be computed in an Excel worksheet as shown below.

The polynomial coefficients can be computed by xRegLin\_Coeff. The exact result is  $y = 10 + x + x^2 + x^3$

	A	B	C	D	E	F
1	y	x	x <sup>2</sup>	x <sup>3</sup>		xRegLin_Coeff
2	1120	10	100	1000	a0 =	10
3	1473	11	121	1331	a1 =	1
4	1894	12	144	1728	a2 =	1
5	2389	13	169	2197	a3 =	1
6	2964	14	196	2744	{=xRegLin_Coeff(A2:A12;B2:D12)}	
7	3625	15	225	3375		
8	4378	16	256	4096		REGR.LIN
9	5229	17	289	4913	a0 =	10.000000011532100
10	6184	18	324	5832	a1 =	0.999999997559196
11	7249	19	361	6859	a2 =	1.000000000167230
12	8430	20	400	8000	a3 =	0.999999999996282
13						
14	{=B12^2}		{=B12^3}		{=flip(MatT(REGR.LIN(A2:A12;B2:D12)))}	
15						

We can perform the same calculus with the Excel LINEST (REGR.LIN in italian version). The other nested functions - flip and MatT – have been used only to rearrange the LINEST output as vertical vector, in the same order of the xRegLin\_Coeff.

## Linear Regression Formulas

Generally, the multivariate linear regression function is:

$$y = a_0 + a_1x_1 + a_2x_2 + \dots a_mx_m$$

where:  $[a_0, a_1, a_2 \dots a_m]$

The coefficients of regression can be found by the following algorithm  
Make the following variables substitution:

$$X_i = x_i - \bar{x} \quad \text{for } i = 1..m$$

$$Y = y - \bar{y}$$

where the right values are the averages of samples **y** and **x** respectively:

$$\bar{y} = \frac{1}{n} \sum_k y_k \quad \bar{x}_i = \frac{1}{n} \sum_k x_{i,k}$$

After that, the coefficients **a** =  $[a_1, a_2, \dots a_n]$  are the solution of the following linear system

$$[C] \cdot \mathbf{a} = \mathbf{b}$$

where **[C]** is the cross-covariance matrix  
and **b** is the XY covariance

$$\mathbf{C} = \begin{bmatrix} \sum_j X_{1,j}^2 & \sum_j X_{1,j}X_{2,j} & \sum_j X_{1,j}X_{3,j} & \dots \sum_j X_{1,j}X_{m,j} \\ = & \sum_j X_{2,j}^2 & \sum_j X_{2,j}X_{3,j} & \dots \sum_j X_{2,j}X_{m,j} \\ = & = & \sum_j X_{3,j}^2 & \dots \sum_j X_{3,j}X_{m,j} \\ = & = & = & \sum_j X_{m,j}^2 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} \sum_j Y_jX_{1,j} \\ \sum_j Y_jX_{2,j} \\ \sum_j Y_jX_{3,j} \\ \dots \\ \sum_j Y_jX_{m,j} \end{bmatrix}$$

and the constant coefficient is given by:

$$a_0 = \bar{Y} - \sum_{i=1}^m a_i \bar{X}_i$$

For m=1 we obtain the popular formulas of monovariate linear regression

$$a_1 = \frac{\sum_j Y_j X_j}{\sum_j X_j^2} \quad a_0 = \bar{Y} - a_1 \bar{X}$$

This is the linear solution known as the Ordinary Least Squares (OLS). The analysis of this kind of approach shows that, for large dimensions of n (many measurement values) the matrix C can become nearly singular



## Linear Regression Covariance Matrix

---

**xRegLin\_Covar( Y, X , [DgtMax], [Intcpt] )**

**RegLin\_Covar( Y, X , [Intcpt] )**

Returns the (m+1 x m+1) covariance matrix of a linear regression of m independent variables

$$\hat{y} = a_0 + a_1x_1 + a_2x_2 \dots + a_mx_m$$

For a given set of n points  $P_i = (x_{1i} \ x_{2i} \ \dots \ x_{mi}, y_i)$

Parameter Y is an (n x 1) vector of dependent variable. Parameter X is a matrix of independent variables. It may be an (n x 1) vector for monovariate regression or an (n x m) matrix for multivariate regression.

Parameter Coeff is a vector of m+1 coefficients of the linear regression

For standard precision use the faster RegLin\_Covar

### Cross Covariance Matrix

Given the matrix **X** of the independent variables points

$$X = \begin{bmatrix} 1 & x_{11} & \dots & x_{m1} \\ 1 & x_{12} & \dots & x_{m2} \\ \dots & \dots & \dots & \dots \\ 1 & x_{1n} & \dots & x_{mn} \end{bmatrix}$$

The covariance matrix C is

$$C = s^2 \cdot (X \cdot X^T)^{-1}$$

where:

$$s^2 = \frac{\sum_i (y_i - \hat{y}_i)^2}{n - m - 1}$$

Note that the square roots of the diagonal elements of the covariance matrix

$$s_i = \sqrt{c_{ii}}$$

are the standard deviations of the linear regression coefficients

## Linear Regression Statistics

---

**xRegLinStat( Y, X, Coeff, [DgtMax], [Intcpt])**

**RegLinStat( Y, X, Coeff, [Intcpt])**

Returns some statistics about the linear regression

$R^2$	Square of the linear correlation factor
$S_{y,x}$	Standard deviation of the linear regression

Parameter Y is a vector (n x 1) of dependent variable.

Parameter X is a list of independent variable. It may be an (n x 1) vector for monovariate regression or a (n x m) matrix for multivariate regression.

Coeff is the coefficients vector of the linear regression function  $[a_0, a_1, a_2 \dots a_m]$ .

For standard precision use the faster RegLin\_Covar

### Formulas

The regression factor (better: the square of regression factor)  $R^2$  lie between 0 and 1 and roughly indicates how closely the regression function fits the given values Y.

Generally, it can be computed by the following formula:

$$R^2 = 1 - \frac{\sum_i (y_i - y_i^*)^2}{\sum_i (y_i - \bar{y})^2} = 1 - \frac{\sigma_{y-y^*}^2}{\sigma_y^2}$$

Where  $y^*$  is the value estimated by the regression function and  $\bar{y}$  is the mean of y values.

$$y^* = a_0 + a_1 x_1 + a_2 x_2 + \dots a_m x_m$$

$$\bar{y} = \frac{1}{n} \sum_k y_k$$

For monovariate regression (m=1), the above formula returns the popular formula:

$$R^2 = \frac{\sum x^2 - \frac{(\sum x)^2}{n}}{\sum y^2 - \frac{(\sum y)^2}{n}}$$

**Standard error** of the linear regression is:

Intercept calculated

$$s_{y,x} = \sqrt{\frac{\sum_i (y_i - y_i^*)^2}{n - gl - 1}}$$

Intercept constrained to 0

$$s_{y,x} = \sqrt{\frac{\sum_i (y_i - y_i^*)^2}{n - gl}}$$

Where gl = number of independent variables

## Linear Regression Evaluation

**= xRegLin\_Eval(Coeff, X)**

**= RegLin\_Eval(Coeff, X, [DgtMax])**

Evaluates the multivariate linear regression in multi precision arithmetic.

Parameter Coeff is the coefficients vector [a0, a1, a2, ....] of the linear regression

Parameter X is the vector of independent variables. It is one value for a simple regression

For standard precision use the faster RegLin\_Eval function

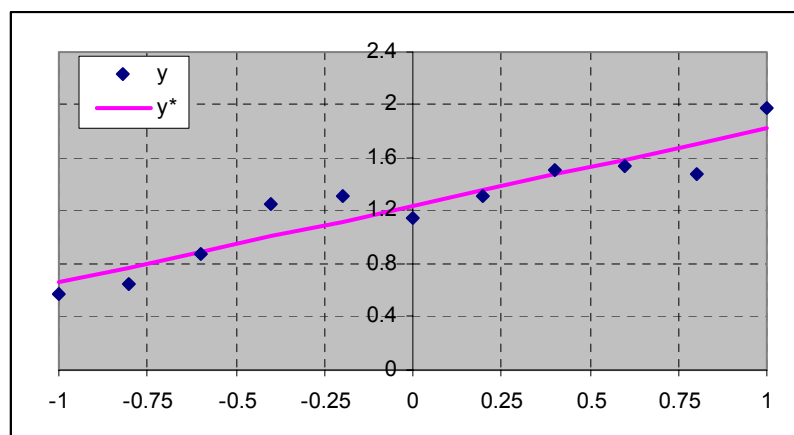
The functions return the linear combination.

$$y = a_0 + a_1x_1 + a_2x_2 + \dots a_nx_n$$

Example: Plot the linear regression for the following data set

x	y	A	B	C	D	E	F
-1	0.58	x	y	y*		coefficients	
-0.8	0.65	-1	0.58	0.66	a0 =	1.24	
-0.6	0.88	-0.8	0.65	0.77	a1 =	0.582272727	
-0.4	1.25	-0.6	0.88	0.89			
-0.2	1.32	-0.4	1.25	1.01			
0	1.14	-0.2	1.32	1.12			
0.2	1.31	0	1.14	1.24			
0.4	1.51	0.2	1.31	1.36			
0.6	1.54	0.4	1.51	1.47			
0.8	1.48	0.6	1.54	1.59			
1	1.98	0.8	1.48	1.71			
		1	1.98	1.82			

In this sheet , each value of linear regression  $y^*$  is computed by the RegLin\_Eval function. The coefficients are computed by the RegLin\_Coeff  
Selecting the three columns and plotting the data we get the following graphs



## Summary of Linear Regressions

Let's perform the linear regression of the following data set having 9 observations, 2 independent variables and 1 dependent variable

$x_1$	$x_2$	$y$
-4	0	-4
-3	0	-2.1
-2	1	-1
-1	1	1
0	2	2
1	2	4.1
2	3	5
3	3	7
4	4	8

First of all we have to compute the coefficients of the linear regression  $[a_0, a_1, a_2]$  by the RegLin\_Coeff

Then, with this coefficients, we can compute the regression factor  $R^2$  and the standard error by the RegLinStat.

We can also compute the covariance matrix, by the RegLin\_Covar, and the standard error of each coefficient

	A	B	C	D	E	F	G
1	$x_1$	$x_2$	$y$		$a_0 =$	4	
2	-4	0	-4		$a_1 =$	2.006666667	
3	-3	0	-2.1		$a_2 =$	-1	
4	-2	1	-1				
5	-1	1	1		{=RegLin_Coeff(C2:C10;A2:B10)}		
6	0	2	2				
7	1	2	4.1		$R^2 =$	0.99987327	
8	2	3	5		$S =$	0.05374838	
9	3	3	7				
10	4	4	8		{=RegLin_R2(C2:C10;F1:F3;A2:B10)}		
11							
12	Covariance matrix						
13	0.01676	0.00462	-0.00924		{=RegLin_Covar(C2:C10;A2:B10;F1:F3)}		
14	0.00462	0.00135	-0.0026				
15	-0.0092	-0.0026	0.0052				

The standard error of each coefficient  $a_0, a_1, a_2$  can be derived by the corresponding diagonal element of the covariance matrix

$$s_i = \sqrt{c_{ii}}$$

coefficients	value	err. std.
$a_0 =$	4	0.129443252
$a_1 =$	2.006666667	0.036717137
$a_2 =$	-1	0.072111026

## Sub-Tabulation

One important application of linear regression is the sub-tabulation, which is the method to extract a table of values with smaller step from an original table with bigger steps. In other words, we can obtain a fine tabulation from a table with a few values of a function. Let's see this example.

Example: Extract from the following dataset, a table having 10 values with step 0.1

x	y
0	5.1
0.2	4.7
0.5	4.5
0.6	4.3
0.7	4.2
1	3.6

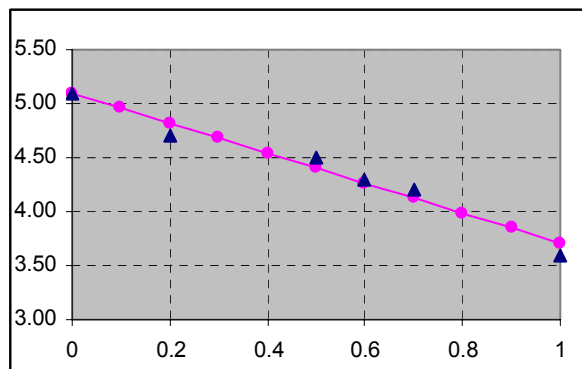
First of all we find the linear regression coefficients

$[a_0, a_1]$

Then we re-calculate the values

$$y_i = a_0 + a_1 x_i, \quad i = 1 \dots 10$$

	A	B	C	D	E
1	Table A			Table B	
2	x	y		x	y
3	0	5.1		0	5.10
4	0.2	4.7		0.1	4.96
5	0.5	4.5		0.2	4.82
6	0.6	4.3		0.3	4.68
7	0.7	4.2		0.4	4.54
8	1	3.6		0.5	4.40
9				0.6	4.26
10	coeff.			0.7	4.12
11	5.0953			0.8	3.98
12	-1.3906			0.9	3.84
13				1	3.70
14					
15	=RegLin_Eval(\$A\$11:\$A\$				
16	{=RegLin_Coeff(B3:B8;A3:A8)}				
17					



The graph shows the extra points added by the sub tabulation. Note that this method is different from the interpolation because the regression line does not pass through any of the original points. The new values of the table B are different from the ones table A even in the same x-values.

This feature came in handy when we want to regularize the row data.

## Data Conditioning

The conditioning of the data consists of subtracting the mean from the values of the sample. It can improve the accuracy of the linear regression, but the regression coefficients obtained - conditioned coefficients - are different from the regression coefficients of the row data. They can be re-obtained by the following method:

Given X and Y two data vectors, the linear regression polynomial of n degree will be:

$$p(x) = \sum_{i=0}^n a_i \cdot x^i$$

We made the data conditioning, making the average of X and Y

$$\bar{x} = \frac{1}{n} \sum x_i \quad \bar{y} = \frac{1}{n} \sum y_i$$

Substituting the old variables with the new variable u and v

$$u_i = x_i - \bar{x} \quad v_i = y_i - \bar{y}$$

Then, the new linear regression polynomial will be:

$$p(u) = \sum_{i=0}^n b_i \cdot u^i$$

The original  $a_i$  coefficients can be obtained from the new  $b_i$  coefficients by the following formulas.

$$a_0 = \bar{y} + \sum_{i=0}^n (-1)^i \cdot b_i \cdot \bar{x}^i \quad a_k = \sum_{i=k}^n (-1)^{i+k} \binom{i}{k} \cdot b_i \cdot \bar{x}^{i-k}$$

This method is often very useful for accuracy increasing

## Data Conditioned Linear Regression Coefficients

**= RLCondCoef(Coef, Ym, Xm)**

This function transforms the coefficients of the conditioned linear regression to the original coefficients

Regression coefficients with conditioned data		Regression coefficients with original data
[b <sub>0</sub> , b <sub>1</sub> , b <sub>2</sub> ,...]	⇒	[a <sub>0</sub> , a <sub>1</sub> , a <sub>2</sub> ,...]

The parameter Coef is the vector of the regression coefficients with data conditioned.

Parameter Ym is the mean of Y-values

Parameter Xm is the mean of X-values. It can be a vector for multivariate regression

Example: Compute the linear regression for the following dataset, where  $x_1$ ,  $x_2$  are the independent variables and y is the dependent variable

x1	x2	y
200	8000000	8000210
201	8120601	8120812
202	8242408	8242620
203	8365427	8365640
204	8489664	8489878
205	8615125	8615340
206	8741816	8742032

The model for this data set is

$$y = a_0 + a_1 x_1 + a_2 x_2$$

Where [a<sub>0</sub>, a<sub>1</sub>, a<sub>2</sub>] are the unknown coefficients

We use the Excel function LINEST

	A	B	C	D	E	F	G
1	x1	x2	y		a2	a1	a0
2	200	8000000	8000210	coeff. =	1.0000000005974	0.999261344327	10.099956646851
3	201	8120601	8120812	riferim. =	1	1	10
4	202	8242408	8242620	LRE =	8.22	3.13	2.00
5	203	8365427	8365640				
6	204	8489664	8489878				
7	205	8615125	8615340				
8	206	8741816	8742032				

**Formulas:**

- =mjkLRE(E2;E3;15)** (points to cell E4)
- =LINEST(C2:C8;A2:B8)** (points to cell G4)

We have also added the exact values  $a_0 = 10$ ,  $a_1 = 1$ ,  $a_2 = 1$ . In order to measure the accuracy we have computed the LRE (Log relative error) with the `mjLRE` function. We wonder if it would be possible to increase the accuracy without using the multiprecision arithmetic (slow) or changing the computer (expensive)? Yes, this is possible using the data conditioning method. Let's see how.

For each column of the original data set (raw data table) we compute the average. We can use the Excel function AVERAGE, for standard numbers, or xmean, for extended numbers. Then, we build a new table (conditioned data table) where each column-element is the difference between the raw column-element and the corresponding mean.

	A	B	C	D	E	F	G
1	x1	x2	y		u1	u2	v
2	200	8000000	8000210		-3	-367863	-367866
3	201	8120601	8120812		-2	-247262	-247264
4	202	8242408	8242620		-1	-125455	-125456
5	203	8365427	8365640		0	-2436	-2436
6	204	8489664	8489878		1	121801	121802
7	205	8615125	8615340		2	247262	247264
8	206	8741816	8742032		3	373953	373956
9							
10	x1 mean	x2 mean	y mean	=A8-A\$11	=B8-B\$11	=C8-C\$11	
11	203	8367863	8368076				

For definition, the conditioned data columns have mean 0.  
Now compute the linear regression of the conditioned data, using the LINEST function

E	F	G	H	I	J	K
u1	u2	v		a2	a1	a0
-3	-367863	-367866	coeff. =	1.00000000000000	1.00000000000000	0.00000000000000
-2	-247262	-247264				
-1	-125455	-125456		computed	references	LRE
0	-2436	-2436	a0	10.0000000000000	10	15
1	121801	121802	a1	1.00000000000000	1	15
2	247262	247264	a2	1.00000000000000	1	15
3	373953	373956				

(=LINEST(C2:C8;A2:B8))  
 (=RLCondCoef(flip(MatT(I2:K2));C11;A11:B11))

Now, surprisingly, the accuracy is excellent! The only fact is that the new coefficients are not exactly the coefficient of the given data. We can obtain the original coefficients by the formulas of the previous topic, or, more easily, by the **RLCondCoef** function. Note. We have used the **flip(MatT(I2:K2))** to reorder the coefficients vector as needed from the RLCondCoef

Tip: the data conditioning method works also for polynomial regressions

## Linear Regression with Robust Method

### RegLinRM(x, y, [Method])

This function<sup>6</sup> performs the linear regression with three different robust methods:

- SM: simple median
- RM: repeated median
- LMS: least median squared

Robust methods are suitable for data containing wrong points. When data samples have noise (experimental data), the basic problem is that classic LMS (least minimum squared) is highly affected by noisy points. The main goal of robust methods is to minimize as much as possible the influence of the wrong points when fitting the function

The parameter x and y are two vectors of the points to fit.

The optional parameter "Method" sets the method you want to use (default = SM)

The functions returns an array of two coefficients [a1, a0] where

$$y \cong a_1 \cdot x + a_0$$

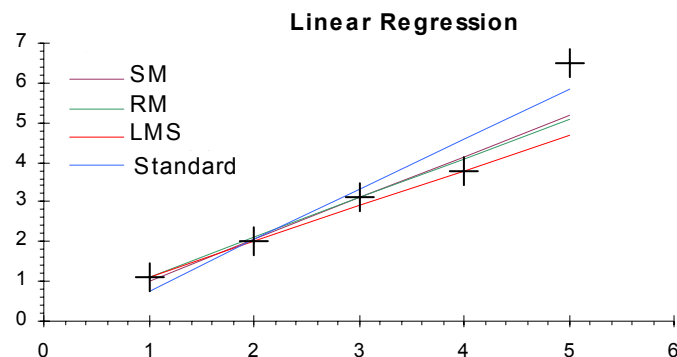
Use CTRL+SHIFT+ENTER to paste it.

Example: Suppose you have sampled 5 experimental values (xi, yi), with a (suspected) large error in the last value 6.5.

x	y
1	1.1
2	2
3	3.1
4	3.8
5	6.5

In the graph are shown the regression lines obtained with all robust methods in comparison with the standard OLS regression.

As we can see all the lines SM, RM, LMS (Robust Methods) minimize the influence of the value (5, 6.5)



<sup>6</sup> The routines for robust linear regression were developed by Alfredo Álvarez Valdivia. They appear in this collection thanks to its courtesy



## Linear Regression Min-Max

### RegLinMM(x, y)

This function performs the linear regression with the Min-Max criterion (also called Chebychev approximation) of a discrete dataset (x, y)

The parameter "x" is a (n x 1) vector of the independent variable,  
The parameter "y" is a (n x 1) vector of the dependent variable

The function returns the coefficients [a0, a1] of the max-min linear regression

$$\tilde{y} = a_0 + a_1 x$$

As known, those coefficients minimize the max absolute error for the given dataset

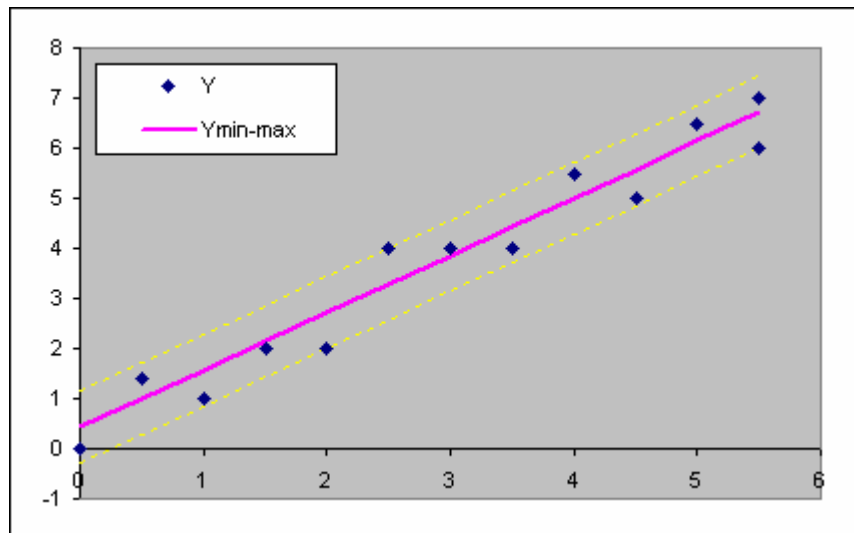
$$E = \max | \tilde{y}(x_i) - y_i |$$

Example. Find the better fitting line that minimize the absolute error

	A	B	C	D	E	F	G
1	<b>x</b>	<b>y</b>	<b>Ymin-max</b>	<b>Error</b>			
2	0	0	0.42857	0.42857		<b>Coefficients</b>	
3	0.5	1.4	1.00000	-0.40000		<b>a0</b>	<b>a1</b>
4	1	1	1.57143	0.57143		0.428571	1.142857
5	1.5	2	2.14286	0.14286			
6	2	2	2.71429	0.71429		{=RegLinMM(A2:A14;B2:B14)}	
7	2.5	4	3.28571	-0.71429			
8	3	4	3.85714	-0.14286		<b>ErrMax =</b>	0.7143
9	3.5	4	4.42857	0.42857		{=MAX(ABS(D2:D14))}	
10	4	5.5	5.00000	-0.50000		=\$E\$2+\$F\$4*A14	
11	4.5	5	5.57143	0.57143		=\$E\$2+\$F\$4*A14	
12	5	6.5	6.14286	-0.35714		=\$E\$2+\$F\$4*A14	
13	5.5	6	6.71429	0.71429		=\$E\$2+\$F\$4*A14	
14	5.5	7	6.71429	-0.28571			

The liner regression is  $y \cong 0.428 + 1.142 x$   
with an error max  $E_{\max} \cong \pm 0.7$

The scatter plot shows the lineare regression approximation



As we can see, all the points lie in the plane strips of  $\pm E_{\max}$  around the min-max line (pink line). ( $E_{\max} \cong 0.7$  in this example)

## Certification Results for Linear Regression

XNUMBERS addin is not a specific statistical package. But it contains a few useful functions for linear regression and univariate summary statistic showing interesting performance. Here, we report the NIST StRD<sup>7</sup> test for Linear Regression Coefficients for the the following functions:

<b>RegLin_Coeff()</b>	XNUMBERS function with standard double precision
<b>xRegLin_Coeff()</b>	XNUMBERS function with multiprecision
<b>LINEST</b>	EXCEL built-in function

Let's apply each of the above function to the NIST/ITL Longley test, a multivariate regression with 6 predictor variables and 16 data.

```
NIST/ITL StRD
Dataset Name: Longley (Longley.dat)
Data:
    1 Response Variable (y)
    6 Predictor Variable (x)
    16 Observations
    Higher Level of Difficulty
    Observed Data

Model:
    Polynomial Class
    7 Parameters (B0,B1,...,B7)
    y = b0 + b1*x1 + b2*x2 + b3*x3 + b4*x4 + b5*x5 + b6*x6
```

Test row data are:

	raw data						
	y	x1	x2	x3	x4	x5	x6
1	60323	83.0	234289	2356	1590	107608	1947
2	61122	88.5	259426	2325	1456	108632	1948
3	60171	88.2	258054	3682	1616	109773	1949
4	61187	89.5	284599	3351	1650	110929	1950
5	63221	96.2	328975	2099	3099	112075	1951
6	63639	98.1	346999	1932	3594	113270	1952
7	64989	99.0	365385	1870	3547	115094	1953
8	63761	100.0	363112	3578	3350	116219	1954
9	66019	101.2	397469	2904	3048	117388	1955
10	67857	104.6	419180	2822	2857	118734	1956
11	68169	108.4	442769	2936	2798	120445	1957
12	66513	110.8	444546	4681	2637	121950	1958
13	68655	112.6	482704	3813	2552	123366	1959
14	69564	114.2	502601	3931	2514	125368	1960
15	69331	115.7	518173	4806	2572	127852	1961
16	70551	116.9	554894	4007	2827	130081	1962

Now let's calculate the linear regression coefficients, and compare the result with their certified values. In order to show their accuracy, we calculate also the LRE for each result

The NIST StRD certified coefficients are:

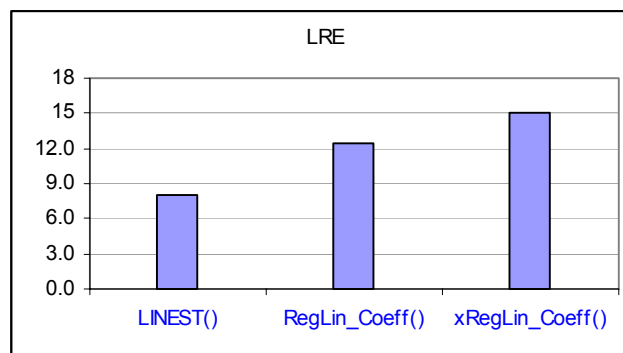
<sup>7</sup> The Statistical Engineering and Mathematical and Computational Sciences Divisions of the Information Technology Laboratory (National Institute of Standards and Technology) has released a number of benchmark datasets for assessing the numerical accuracy of statistical software. The Statistical Reference Datasets (StRD) were designed explicitly to assist researchers in benchmarking statistical software packages

NIST/ITL StRD	
Param.	Certified Values
b0	-3482258.63459582
b1	15.0618722713733
b2	-0.0358191792925910
b3	-2.02022980381683
b4	-1.03322686717359
b5	-0.0511041056535807
b6	1829.15146461355

While the calculated results are shown in the following table (the xRegLin\_Coeff output has been converted into double precision).

Param.	LINEST()		RegLin_Coeff()		xRegLin_Coeff()	
	Estimate	LRE	Estimate	LRE	Rounded Estimate	LRE
b0	-3482258.65389031	8.3	-3482258.63459501	12.6	-3482258.63459582	15
b1	15.0618726770786	7.6	15.0618722713460	11.7	15.0618722713733	15
b2	-0.0358191798902255	7.8	-0.0358191792925737	12.3	-0.0358191792925910	15
b3	-2.02022981272773	8.4	-2.02022980381635	12.6	-2.02022980381683	15
b4	-1.03322686974925	8.6	-1.03322686717341	12.8	-1.03322686717359	15
b5	-0.0511041036005626	7.4	-0.0511041056535569	12.3	-0.0511041056535807	15
b6	1829.15147447748	8.3	1829.15146461313	12.6	1829.15146461355	15

Taking the average of LRE, we obtaining the following graph of the general accuracy



As we can see both functions for linear regression give very accurate result. The multiprecision version xRegLin is the top but, of course, is also much more slow then the correspondent version in 32-bit precision.

## Linear Regression - General Accuracy

The NIST StRD Statistical Reference Datasets include several linear regression problem tests in each of three difficulty levels: low, average, and high. These benchmarks were specifically designed so that reliable algorithms implemented in double precision would produce acceptable results for all four suites.

Repeating the calculus for each linear regression StRD datasets we obtain the following table showing the general accuracy performance.

## Xnumbers Tutorial

Method	Accuracy
1) <a href="#">LINEST</a>	<a href="#">9.7</a>
2) <a href="#">RegLin_Coeff()</a>	<a href="#">11.5</a>
3) <a href="#">xRegLin_Coeff()</a>	<a href="#">15</a>

NIST StRD Dataset Properties for Linear Regression								
Name	Level of difficulty	Model of class	Param.	variables	Points	(1) LRE	(2) LRE	(3) LRE
Norris	low	Linear	2	1	36	13.5	14.7	15
Pontius	low	Quadratic	3	1	40	12.5	14.3	15
NoInt1	medium	Linear	1	1	11	15	15	15
NoInt2	medium	Linear	1	1	3	15	15	15
Filip	high	Polynomial	11	1	82	0	0	15
Longley	high	Multilinear	7	6	16	8	12.3	15
Wampler1	high	Polynomial	6	1	21	8.1	11.7	15
Wampler2	high	Polynomial	6	1	21	10.3	13.5	15
Wampler3	high	Polynomial	6	1	21	8.1	11.5	15
Wampler4	high	Polynomial	6	1	21	8.1	10	15
Wampler5	high	Polynomial	6	1	21	8.1	8.9	15

This table shows the high accuracy of the regression routine of Xnumbers. Of course all that has a cost: the multiprecision computation is much slower than the standard one. The multiprecision should be used only when needed. For example, the Filippelli test needs the multiprecision computing because, in standard precision, the result is totally wrong

## Transcendental Functions

### Logarithm natural (Napier's)

---

#### xLn(x, [Digit\_Max])

Returns the natural logarithm (or Napier's) , in base "e"  
The argument may be either normal or extended number.  
Example:

xLn(30) = 3.4011973816621553754132366916

### Logarithm for any base

---

#### xLog(x, [base], [Digit\_Max])

Returns the logarithm for any base (default 10)

$$y = \log_{base}(x)$$

The argument may be either normal or extended number.  
Example

xlog(30) = 1.47712125471966243729502790325

### Exponential

---

#### xexp(x, [Digit\_Max])

Returns the exponential of x in base "e"       $xexp(x) = e^x$

Example

$$e^{10} = xexp(10) = 22026.4657948067165169579006452$$

$$e^{1000} = xexp(1000) = 1.97007111401704699388887935224E+434$$

Note the exponent 434 of the second result. Such kind of numbers can be managed only with extended precision functions because they are outside the standard limits of double precision.

### Exponential for any base

---

#### xexpbase(a, x, [Digit\_Max])

Returns the exponential of x any in base       $xexpbase(a,x) = a^x$   
The arguments "a" and "x" may be either normal or extended numbers, with a > 0.  
Example.

$2^{1.234} = \text{xexpbase}(2, 1.234) = 2.3521825005819296401155858555$   
 $0.365^{-0.54} = \text{xexpbase}(0.365, -0.54) = 1.72330382988412269578819213881$

### Constant “e”

---

#### **xe([Digit\_Max])**

Returns Euler's constant "e", the base of the natural logarithm.  
The optional parameter Digit\_Max, from 1 to 415, sets the number of significant digits (default 30).

`xe()` = 2.71828182845904523536028747135  
`xe(60)` = 2.71828182845904523536028747135266249775724709369995957496696

### Constant Ln(2)

---

#### **xLn2([Digit\_Max])**

Returns the constant Ln(2).  
The optional parameter Digit\_Max, from 1 to 415, sets the number of significant digits (default 30).

### Constant Ln(10)

---

#### **xLn10([Digit\_Max])**

Returns the constant Ln(10).  
The optional parameter Digit\_Max, from 1 to 415, sets the number of significant digits (default 30).

### Hyperbolic Sine

---

#### **xsinh(x, [Digit\_Max])**

Returns the hyperbolic sine of x in multiprecision arithmetic

$$\sinh = \frac{e^x - e^{-x}}{2}$$

### Hyperbolic ArSine

---

#### **xasinh(x, [Digit\_Max])**

Returns the hyperbolic arsine of x in multiprecision arithmetic

$$\operatorname{asinh}(x) = \ln\left(x + \sqrt{x^2 + 1}\right)$$

## Hyperbolic Cosine

---

### **xcosh(x, [Digit\_Max])**

Returns the hyperbolic cosine of x in multiprecision arithmetic

$$\cosh(x) = \frac{e^x + e^{-x}}{2}$$

## Hyperbolic ArCosine

---

### **xacosh(x, [Digit\_Max])**

Returns the hyperbolic Arcosine of x in multiprecision arithmetic  
The argument x, normal or extended, must be  $x > 1$

$$\operatorname{acosh} = \ln\left(x + \sqrt{x^2 - 1}\right), \quad x > 1$$

## Hyperbolic Tangent

---

### **xtanh(x, [Digit\_Max])**

Returns the hyperbolic tangent of x in multiprecision arithmetic

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

## Hyperbolic ArTangent

---

### **xatanh(x, [Digit\_Max])**

Returns the hyperbolic artangent of x in multiprecision arithmetic  
The argument x, normal or extended, must be  $|x| < 1$

$$\operatorname{atanh}(x) = \frac{1}{2} \ln\left(\frac{1+x}{1-x}\right), \quad |x| < 1$$



## Euler's constant gamma

---

**=xeu( [Digits\_Max] )**

Returns the Euler-Mascheroni constant gamma  
(The same as Gamma constan returnde by xGm function)

Example

xeu() = 0.57721566490153286060651209008

xeu(60) = 0.57721566490153286060651209008240243104215933593992359880576

## Trigonometric Functions

### Sin

#### **xsin(a, [Digit\_Max])**

Returns the sine of the angle  $a$        $\text{xsin}(a) = \sin(a)$   
 The argument  $a$ , in radians, may be either a normal or an extended number.

$$\text{xsin}(1.5) = 0.997494986604054430941723371141$$

### Cos

#### **xcos(a, [Digit\_Max])**

Returns the cosine of the angle  $a$        $\text{xcos}(a) = \cos(a)$   
 The argument  $a$ , in radians, may be either a normal or an extended number.

$$\text{xcos}(1.5) = 7.07372016677029100881898514342\text{E-}2$$

### Computation effect of $\cos(\pi/2)$

Example: compute  $\cos(89,99999995^\circ)$  with the standard built-in function COS function

$$\text{COS}(89.99999995) = \text{COS}(1.570796326) = 7.94896654250123\text{E-}10$$

The correct answer, accurate to 15 digits, is  $7.94896619231321\text{E-}10$   
 As we can see, only 7 digits are corrected. The remaining 8 digits are meaningless.  
 On the contrary, with the multiprecision function  $\text{xcos}(x)$  we have the correct result with all its significant digits.

$$\text{xcos}(1.570796326) = 7.94896619231321\text{E-}10$$

The table below shows the computation effect when  $a$  approaches  $\pi/2$

angle $\alpha$	$\alpha$ (deg)	$\text{xcos}(\alpha)$	$\text{COS}(\alpha)$ built-in	Err %
1.57	89.95437383553930	7.96326710733325E-4	7.96326710733263E-04	7.75E-14
1.570	89.95437383553930	7.96326710733325E-4	7.96326710733263E-04	7.75E-14
1.5707	89.99448088119850	9.63267947476522E-5	9.63267947476672E-05	-1.55E-13
1.57079	89.99963750135470	6.32679489657702E-6	6.32679489666849E-06	-1.45E-11
1.570796	89.99998127603180	3.26794896619225E-7	3.26794896538163E-07	2.48E-10
1.5707963	89.99999846476560	2.67948966192313E-8	2.67948965850537E-08	1.28E-09
1.57079632	89.99999961068120	6.79489661923132E-9	6.79489670660314E-09	-1.29E-08
1.570796326	89.99999995445590	7.94896619231321E-10	7.94896654250123E-10	-4.41E-08
1.5707963267	89.99999999456290	9.48966192313216E-11	9.48965963318629E-11	2.41E-07
1.57079632679	89.99999999971950	4.89661923132169E-12	4.89658888522954E-12	6.20E-06

As we can see, the accuracy of the standard function COS decreases when the angle approaches the right angle. On the contrary, the  $\text{xcos}$  function keeps its accuracy.

## Tan

---

### **x<sub>tan</sub>(a, [Digit\_Max])**

Returns the tangent of a  $x_{\tan}(a) = \tan(a)$

The argument a, in radians, may be either a normal or an extended number.

## Arcsine

---

### **x<sub>asin</sub>(a, [Digit\_Max])**

Returns the arcsine of a  $x_{\sin}(a) = \arcsin(a)$

The arcsine is defined between  $-\pi/2$  and  $\pi/2$

The argument a, where  $|a| \leq 1$ , may be either a normal or an extended number.

## Arccosine

---

### **x<sub>acos</sub>(a, [Digit\_Max])**

Returns the arccosine of a  $x_{\cos}(a) = \arccos(a)$

The arccosine is defined between 0 and  $\pi$

The argument a, where  $|a| \leq 1$ , may be either a normal or an extended number.

## Arctan

---

### **x<sub>atan</sub>(a, [Digit\_Max])**

Returns the arctan of a  $x_{\tan}(a) = \arctan(a)$

The arctan(a) is defined between

$$-\pi/2 < \arctan(a) < \pi/2$$

## Constant $\pi$

---

These functions return the following multiples of  $\pi$

<b>x<sub>pi</sub>([Digit_Max])</b>	<b>x<sub>pi</sub> = <math>\pi</math></b>
<b>x<sub>pi2</sub>([Digit_Max])</b>	<b>x<sub>pi2</sub> = <math>\pi/2</math></b>
<b>x<sub>pi4</sub>([Digit_Max])</b>	<b>x<sub>pi4</sub> = <math>\pi/4</math></b>
<b>x<sub>2pi</sub>([Digit_Max])</b>	<b>x<sub>2pi</sub> = <math>2\pi</math></b>

The optional parameter Digit\_Max, from 1 to 415, sets the number of significant digits (default 30).

### Complement of right angle

---

#### **xanglecompl(a, [Digit\_Max])**

Returns the complement of angle a to the right angle

$$\text{xanglecompl}(\alpha) = \pi/2 - \alpha$$

where  $0 \leq \alpha \leq \pi/2$ .

Example:

`xanglecompl(1.4)` = 0.17079632679489661923132169163

For angles not too near the right angle this function is like the ordinary subtraction. The use of this function is computing the difference without loss of significant digits when the angle is very close to the right angle. For example, computing in Excel the following difference:

$$=(\text{PI}()/2 - 1.570796) = 1.57079632679490 - 1.570796 = 0,00000032679490$$

we have a loss of 7 significant digits, even though the computation has been made with 15 significant digits. On the contrary, if we use:

`xanglecompl(1,570796 , 15)` = 3,26794896619231E-7

we get the full precision with 15 significant digits. The "lost" digits are automatically replaced

## Polynomial Rootfinder

The roots of polynomials are of interest to more than just mathematicians. They play a central role in applied sciences including mechanical and electrical engineering where they are used in solving a variety of design problems.

All rootfinder routines are largely revised in this version.

Didactical routine like: Lin-Bairstow's, Newton-Raphson's and Halley's method, are not still supported. They will migrate in another workbook. They are substitute by more robust routines based on the following polynomial rootfinder algorithms.

<b>RootFinderJT</b>	Jenkins and Traub algorithm (translated in VB from original FORTRAN 77)
<b>RootFinderGN</b>	Generalized Newton-Raphson method
<b>RootFinderDK</b>	Durand, Kerner algorithm This methods was been developed by Ehrlich (1967) and Aberth (1973). So is also called with these names.
<b>RootfinderRF</b>	Ruffini's method for real integer roots. It uses the Rutishauser' s QD algorithm for roots bracketing.

All these algorithms are able to find, in a few seconds all roots of a dense polynomial up to 15<sup>th</sup> - 20<sup>th</sup> degree, with real or complex roots, in standard double precision or multi-precision. It is remarkable that sometimes the results have shown in exact way, even if the computation is intrinsically approximated. All these algorithms start with random initial guess. Therefore, if your computation is not converging, don't mind! Re-try again.

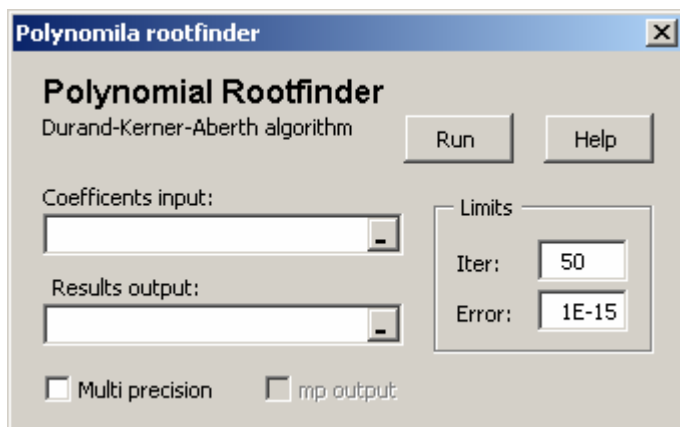
We have to point out that despite the effort dedicated to this problem and the large class of solution methods, any computer algorithm using finite precision is destined to fail for polynomials with sufficiently high degree. Pathological polynomials having tightly clustered roots or very large range are also very difficult to solve. Nevertheless, for polynomials encountered in practical use the above algorithms can find all the roots with good global accuracy.

The characteristics of each rootfinder are synthesized in the following table

Macro	Roots	Coefficients	Arithmetic
RootfinderJT	Complex	Real	Standard
RootfinderGN	Complex	Real	Multiprecision
RootfinderDK	Complex	Complex	Multiprecision
RootfinderRF	Real, integer	Real, integer	Multiprecision

## Input parameters

The input interface has been revised. It is more simple and straight.



**Coefficients input:** is the array containing the polynomial coefficients – from top to bottom – with increasing degree. May be also a single cell containing the polynomial symbolic formula such as:

$-120+274x-225x^2+85x^3-15x^4+x^5$

RootfinderDK can also accept complex coefficients. In that case the input is an (n x 2) array. Examples of possible input ranges are (thick black box):

degree	coef. r	coef. i	degree	coef.	degree	0	1	2	3
0	-150	-50	0	49130	coef.	234	-23	8	1
1	325	105	1	-9883	polynomial $x^{16}-6817x^8+1679616$				
2	-249	-74	2	878					
3	88	21	3	-45					
4	-15	-2	4	-15					
5	1	0	5	1					

### Note

The symbolic notation is more adapt for sparse polynomials.

Real coefficients can be put in horizontal or vertical vector. Complex coefficients, only in vertical vectors

**Results Output:** It is the upper left corner of the output area. If blank, the routine assumes the cell nearest the given coefficients range.

**Error:** Sets the relative roots accuracy. The algorithm terminates when the relative difference between two iterations is less then this value.

**Iter:** The algorithm stops when the iterations counter reaches this value.

**Multi-Precision:** Enable/disable the multi-precision arithmetic

**MP-out:** If checked, the results are written in multi-precision, otherwise they are converted in standard double

## Printing Results

The rootfinder macros write their results in the following simplified layout

The root list and their estimated relative errors are written in a table starting from the left upper cell indicated in the input window. In the right-bottom cell is written the total elaboration time in seconds

degree	coeff.	Real	Imm	Rel. Err.
0	-125	2	1	3.78E-08
1	225	2	-1	1.13E-07
2	-170	2	1	1.86E-07
3	66	2	-1	5.03E-08
4	-13	5	0	2.00E-18
5	1			
				0.046875
				elab. time (sec)

Note: we have formatted the table only for clarity. The macros do not perform this task. You do it!

## Integer Rootfinder output

Integer Rootfinder outputs all integer roots of the polynomial (if any) at the left and the coefficients of the remainder polynomial (deflated polynomial) at the right

B	C	D	E
coeff.		Int. Roots	Poly Rem.
-8704		-2	34
11904		2	-38
-6280		8	23
-328		8	-4
1510			1
-582			
147			
-20			
1			

This result means that the given polynomial

$$x^8 - 20x^7 + 147x^6 - 582x^5 + 1510x^4 - 328x^3 - 6280x^2 + 11904x - 8704$$

can be factorized as

$$(x+2)(x-2)(x+8)^2(x^4 - 4x^3 + 23x^2 - 38x + 34)$$

## How to use rootfinder macros

Using polynomial rootfinder macros is simpler than before. Simply select the coefficients polynomial and start the rootfinder that you prefer. All input fields are filled and the only work that you have to do – in the most cases - is to press "Run".

Now start the RootfinderJT . The input coefficients field is filled with C3:C11 and the output cell is filled with the cell E3. Press "Run" and wait.

	A	B	C	D	E	F	G	H	I	J
1										
2			<b>degree</b>	<b>coeff.</b>						
3			0	40320						
4			1	-109584						
5			2	118124						
6			3	-67284						
7			4	22449						
8			5	-4536						
9			6	546						
10			7	-36						
11			8	1						
12										
13										
14										
15										
16										
17										
18										
19										

**Polynomial Rootfinder**  
 Jenkins and Traub algorithm
 

Run
 Help

 Coefficients input:  
 \$C\$3:\$C\$11
 Results output:  
 \$E\$3
 Limits
 Iter: 100
 Error: 1E-15
 ☐ Multi precision
 ☐ mp output

Press "run" and - after a while - the routine ends and the roots will be displayed at the right, like in the following figure

A	B	C	D	E	F	G	
		<b>degree</b>	<b>coeff.</b>		<b>Real</b>	<b>Imm</b>	<b>Rel. Err.</b>
		0	40320		1	0	1.98E-19
		1	-109584		2	0	7.02E-13
		2	118124		3	0	2.86E-12
		3	-67284		4	0	4.32E-12
		4	22449		5	0	5.19E-12
		5	-4536		6	0	2.23E-12
		6	546		7	0	1.23E-12
		7	-36		8	0	2.63E-13
		8	1				
							0

**Sparse polynomials.** We can pass to the rootfinder macros also symbolic polynomial string, (that it is the faster way for sparse high degree polynomials). Let's see this example

Find all roots of the following 16<sup>th</sup> degree polynomial

$$x^{16} - 6817x^8 + 1679616$$

Write this string in a cell, select it and start a rootfinder macro



A2		fx x^16-6817x^8+1679616							
	A	B	C	D	E	F	G	H	I
1									
2	x^16-6817x^8+1679616								
3									
4	Real	Imm	Rel. Err.						
5	-3	0	3.022E-24						
6	-2.12132	2.1213203	6.684E-17						
7	-2.12132	-2.12132	6.684E-17						
8	-2	0	7.744E-23						
9	-1.414214	1.4142136	3.606E-17						
10	-1.414214	-1.414214	3.606E-17						
11	0	2	7.744E-23						
12	0	3	3.022E-24						
13	0	-2	7.744E-23						
14	0	-3	3.022E-24						
15	1.4142136	1.4142136	3.606E-17						
16	1.4142136	-1.414214	3.606E-17						
17	2	0	7.744E-23						
18	2.1213203	2.1213203	6.684E-17						
19	2.1213203	-2.12132	6.684E-17						
20	3	0	3.022E-24						

**Polynomial rootfinder**

Durand-Kerner-Aberth algorithm

Run Help

Coefficients input:

Results output:

Limits  
 Iter:   
 Error:

☐ Multi precision ☐ mp output

In this case we have used the Durand-Kerner algorithm obtaining a very high accuracy (practically the highest accuracy in standard double precision)

## Root Error Estimation

The third column produced by the rootfinder macros is an estimation of the relative root error, defined as:

$$Er_i = \frac{|\tilde{x} - x_i|}{|x_i|}$$

where  $\tilde{x}$  is the true unknown root and  $x_i$  is the approximate root given by the rootfinder

We have to say that this number should be regarded as an estimation of “goodness” of the root found; small values (for example 1E-9 , 1E-12 ) indicate a great precision of the root. On the contrary, high values (for examples 1E-3 , 1E-5) indicates “difficult” roots that require an extra investigation.

For example assume to find the root of the following 6<sup>th</sup> degree polynomial

Coef.	Real	Imm	Rel. Err.
1.158727752	1.0000001	0	1.85E-07
-6.784680492	1.0099996	0	4.13E-07
16.55167774	1.0200008	0	8.17E-07
-21.534225	1.0299993	0	8.98E-07
15.7585	1.0400003	0	4.01E-07
-6.15	1.05	0	7.94E-08
1			

roots relative error.  
Its an index of "goodness"

**Clustering effect:** In this case, the accuracy is enough good, but quite lower than the previous example. The reason is that the roots:

– 1, 1.01, 1.02, 1.03, 1.04, 1.05

are very close each other (0.1% of difference)

**Complex polynomials.** The macro RootfinderKD can solve also complex polynomials  
Example: find the roots of the following polynomial with complex coefficients

$$P(z) = (-12 + 4i) + 4z + (15 - 5i)z^2 - 5z^3 + (-3 + i)z^4 + z^5$$

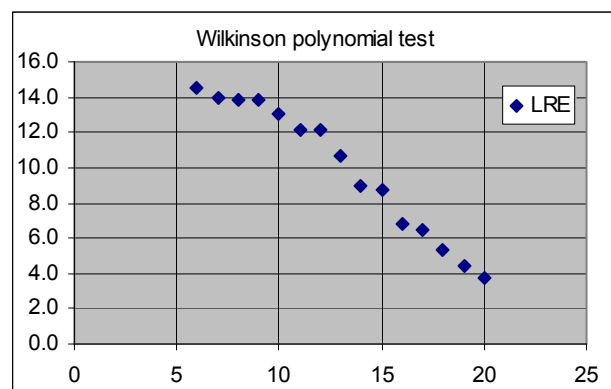
Select both real and imaginary coefficients columns and start the macro RootfinderKD

	A	B	C	D	E	F
1	coef re	coef im		Real	Imm	Rel. Err.
2	-12	4		-2	0	8.172E-18
3	4	0		-1	0	4.042E-17
4	15	-5		1	0	7.454E-17
5	-5	0		2	0	2.946E-17
6	-3	1		3	-1	4.757E-18
7	1	0				
8						0.0078125

The roots are  $z = \pm 1$ ,  $z = \pm 2$ ,  $z = 3 - j$

A polynomial of n degree, having as roots the first integer n numbers, belongs to the Wilkinson class that, as known, is hill-conditioned. This dense polynomial is usually used as standard reference for polynomial rootfinder algorithms. We have tabulated the LRE (log relative error) obtained with all the rootfinder macros.

As we can see, for a Wilkinson polynomial of 20<sup>th</sup> degree, we have exact about four significant digits (0.1% accuracy)



But all polynomials are so hard to solve? Fortunately not. Many polynomials with higher degree, can be solved with good accuracy

For example, if we try to get all real roots of the 16<sup>th</sup> degree Legendre's polynomial

$$6435-875160x^2+19399380x^4-162954792x^6+669278610x^8-1487285800x^{10}+1825305300x^{12}-1163381400x^{14}+300540195x^{16}$$

We have a general accuracy of more than 13 digits

Legendre polyn. Coeff.	Real	Imm	Rel. Err.
6435	-0.989400934991646	0	2.9585E-17
0	-0.944575023073157	0	1.6352E-13
-875160	-0.865631202387904	0	3.673E-14
0	-0.755404408355024	0	1.1559E-14
19399380	-0.617876244402639	0	1.0779E-14
0	-0.458016777657228	0	7.4625E-16
-162954792	-0.281603550779259	0	1.9313E-16
0	-0.095012509837637	0	8.5038E-18
669278610	0.095012509837637	0	8.5038E-18
0	0.281603550779259	0	1.4485E-16
-1487285800	0.458016777657228	0	1.5356E-15
0	0.617876244402640	0	5.0196E-15
1825305300	0.755404408354981	0	2.5615E-14
0	0.865631202387767	0	4.6793E-14
-1163381400	0.944575023073325	0	8.4139E-14
0	0.989400934991655	0	5.5583E-15
300540195			
		Time =	0.34375

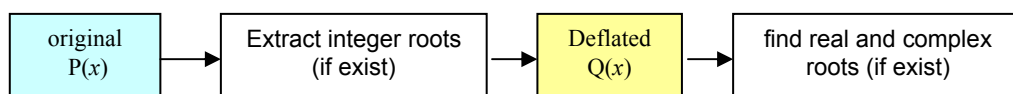
(remember that the higher degree coefficients are at bottom)

In the last column are the estimation errors given by the rootfinder DK. They are slight different from the true roots errors, but we have to remember that this column must be regard as an index of the root approximation: low error values mean a good accuracy, higher errors could mean poor approximation (but not always!)

## Integer roots

In applied science it's rarely to come across polynomials having exact integer roots. Nevertheless, they are frequent in math, didactical examples and algorithm testing . Xnumbers has a dedicated special macro for finding the integer real roots of a polynomial. It uses the Ruffini's method with the QD algorithm for roots isolation. This method is generally less efficient then JT or DK but it can gain in accuracy. The roots found with this method have no round-off errors so the deflated polynomial is exact. Therefore, in that case, the process root-finding-deflating is without errors.

For polynomial having a mix of integer real roots, complex roots and real roots the method returns the integer roots and the coefficients of the deflated polynomial that can be solved with the aid of the general purpose macros: DK, GN or JT. Because the deflated polynomial has a lower degree, the roots accuracy will be generally higher than if we solve directly the given polynomials.



## Xnumbers Tutorial

Let's see how it works practically

Assume to have the following polynomial

degree	coeff
a0	8678880
a1	-13381116
a2	8844928
a3	-3279447
a4	746825
a5	-107049
a6	9437
a7	-468
a8	10

The exact roots are:

integer	real	complex
5, 6, 7, 8, 9	2.8	$4.5 \pm i 0.5$

If we try to solve this 8<sup>th</sup> degree polynomial with a general rootfinder, probably the best accuracy that we can obtain is about 1e-10, that it is a good result but we can do better if we extract the integer roots before and then, solving for the remaining roots

Extract the integer roots and deflated polynomial

	A	B	C	D	E	F	G	H	I	J	K
1	degree	coeff		Int. Roots	Poly Rem.	Polynomial rootfinder					
2	a0	8678880		5	-574	<b>Polynomial Integer Rootfinder</b> Ruffini's method <span>Run</span> <span>Help</span> Coefficients input: <input type="text" value="\$B\$2:\$B\$10"/> <span>-</span> Results output: <input type="text" value="\$D\$2"/> <span>-</span> Limits Iter: <input type="text" value="1000"/> <input type="checkbox"/> Multi precision					
3	a1	-13381116		6	457						
4	a2	8844928		7	-118						
5	a3	-3279447		8	10						
6	a4	746825		9							
7	a5	-107049									
8	a6	9437									
9	a7	-468									
10	a8	10									
11				0.03125							
12											
13											
14											
15											
16											

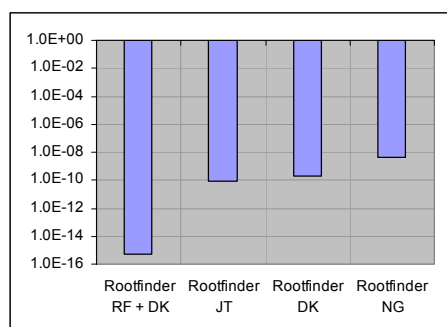
The original polynomial is now cracked into the following factors

$$(x-5)(x-6)(x-7)(x-8)(10x^3 - 118x^2 + 457x - 574)$$

Now let's find the roots of the following 3<sup>rd</sup> degree polynomial by, for example, the general JT rootfinder. We obtain:

Re	Im	Rel. Err.
2.8	0	1.14E-17
4.5	0.5	1.44E-15
4.5	-0.5	1.44E-15

The general accuracy is better than 1e-14, thousand times than the direct method. Clearly is a good thing to keep attention to the integer roots (when there are).



Global roots accuracy versus the solving methods:

Rootfinder RF + DK  
Rootfinder JT  
Rootfinder DK  
Rootfinder NG

## Xnumbers Tutorial

The multiprecision should be used when the coefficients exceed 15 digits (remember that the coefficients must be exact in order to extract the exact integer roots)

Let's see the following 18<sup>th</sup> degree polynomial having the roots

Coefficients
-612914519230813800000
91181999821816015800
-5186948337826516202
137665995531841931
-1622627967498318
6214402509219
-11208193158
10605849
-5122
1

### Polynomial roots

integer	real	complex
25, 27, 29, 31, 1000, 1001, 1002, 1003, 1004	none	none

Note that same coefficients have 16 - 18 significant digits and they must be inserted as x-numbers, (that is as string) in order to preserve the original precision. We have also to set the multiprecision check-box in the macro RootfinderRF

The screenshot shows a spreadsheet with columns A through J. Column A contains degrees (a0 to a9), column B contains coefficients (as strings), and column D contains integer roots (25, 27, 29, 31, 1000, 1001, 1002, 1003, 1004). A dialog box titled 'Polynomial Integer Rootfinder' is open, showing 'Ruffini's method' selected, 'Coefficients input' as '\$B\$43:\$B\$52', 'Results output' as '\$D\$43', and 'Limits' set to 'Iter: 1000'. The 'Multi precision' checkbox is checked and circled in red.

Note that this is a so called clustered polynomial because some of its integer roots (1000, 1001, 1002, 1003, 1004) are very close each other (difference less than 1%). This situation is quite difficult for many algorithms and the accuracy is generally quite poor. On the contrary, the Ruffini's method works very fine in that case.

## Central Polynomial

We call "central normalized polynomial" a polynomial having the center B of his roots equal to point (0 , 0).

Given a generic normalized polynomial ( $a_n=1$ )

$$P(z) = a_n z^n + a_{n-1} z^{n-1} + \dots a_3 z^3 + a_2 z^2 + a_1 z + a_0$$

For real coefficients, the roots are symmetric to the y-axis, so the  $B_y = 0$ .  
While for  $B_x$  we have:

$$x_c = \frac{\sum x_i}{n} = -\frac{a_{n-1}}{n}$$

So, the central condition implies:

$$x_c = 0 \Leftrightarrow a_{n-1} = 0$$

Any generic polynomial can be transformed in "central" by the following translation:

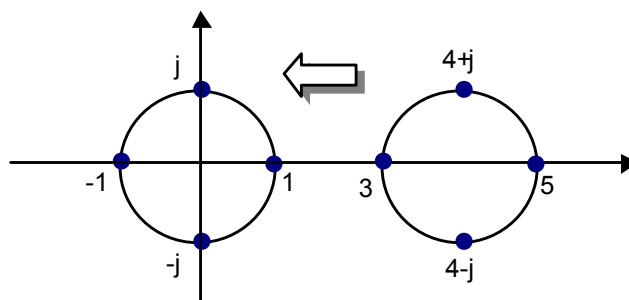
$$z = s + x_c$$

Example:

$$z^4 - 16 z^3 + 96 z^2 - 256 z + 255 \quad \xrightarrow{z = s + 4} \quad s^4 - 1$$

As we can see, transforming a generic polynomial into a center polynomial may reduce the complexity and the magnitude of coefficients. This is very important to avoid the overflow during numeric computing and also the convergence of the iterative rootfinder methods can be greatly improved.

The graph below shows the transformation effect. All the roots are shifted to the origin



## Coefficients Transformation

Given a polynomial

$$P(z) = a_n z^n + a_{n-1} z^{n-1} + \dots a_3 z^3 + a_2 z^2 + a_1 z + a_0$$

Setting the variable substitution:

$$z = x + x_c$$

where

$$x_c = -\frac{a_{n-1}}{n}$$

The above translation involves the transformation of all original coefficients. Indicated the central polynomial as:

$$b_n x^n + b_{n-2} x^{n-2} + \dots b_2 x^2 + b_1 x + b_0$$

Then, the coefficients b can be given by the following formulas:

$$b_k = \frac{P^{(k)}(x_c)}{k!} \quad k = 0 \dots n$$

We can avoid the computation of the n-th order derivatives and the computation of factorial by the following the iterative method:

Starting with

$$P_0(z) = P(z)$$

For  $k = 0, 1 \dots n$

$$b_k = P_k(x_c)$$

$$P_{k+1} = \frac{1}{k+1} \frac{dP_k}{dz}$$

## Circle of the Roots

We define "circle of the roots" the smallest circle that contains all the roots of a polynomial. The radius of this circle is:

$$R = \max_{i=1 \dots n} (|z_i|)$$

If the polynomial is "central", a good estimation for R is:

$$R^* = 1.1 \cdot \max_{i=2 \dots n} \left( |b_{n-i}|^{\frac{1}{i}} \right)$$

That is:

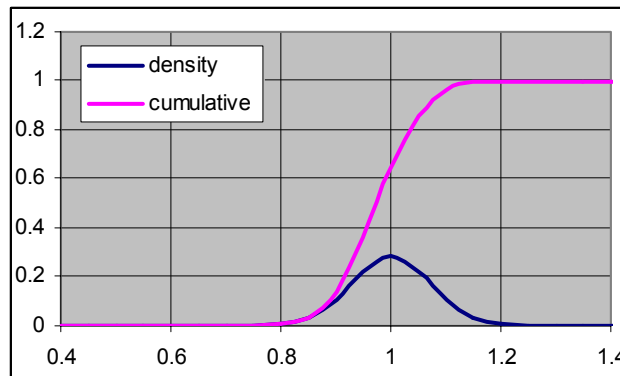
$$R^* = 1.1 \cdot \max \left( |b_0|^{\frac{1}{n}}, |b_1|^{\frac{1}{n-1}}, |b_2|^{\frac{1}{n-2}}, \dots, |b_{n-3}|^{\frac{1}{3}}, |b_{n-2}|^{\frac{1}{2}} \right)$$

Of course the "true" R is not exactly the R\*, but is distributed around R\* with a statistical distribution. If we define the stochastic variable:

$$t = \frac{R}{R^*}$$

We can also define p(t) and F(t) respectively the Probability Density and the Probability cumulative function of statistical distribution of "t".

The graph below shows an example of statistical distribution given from a sample of a few hundred random polynomials, from 3° to 6° degree.



As we can see, for  $t = 1$  the probability is about 50%. Thus, the probability to find all the roots in a circle with radius equal to  $R^*$  is about 50%.

The probability becomes more than 99% for a radius of  $1.2 R^*$

This result helps to restrict the searching area of the polynomial roots.



## Polynomial Functions

### Polynomial evaluation

**=POLYN(z, Coefficients, [DgtMax])**

Computes the polynomial at the value z.

$$P(z) = a_0 + a_1 z + a_2 z^2 + \dots a_n z^n$$

The parameter Coefficients is the (n+1) column vector containing the polynomial coefficients from the lowest to the highest order.

This function accept also complex coefficients. In that case the parameter Coefficients is an (n+1 x 2) array.

The optional parameter DgtMax set the number of the precision digits. If omitted, the function works in the faster double precision.

This function works also for complex arguments. In that case, z must be a complex number (two adjacent cells) and the function returns two values. To see both real and imaginary part, select two cells and give the CTRL+SHIFT+ENTER key sequence. If you press only ENTER, the function returns only its real part.

Example: compute the following real polynomial

$$P(z) = 2z^4 + z^3 - 5z^2 + 2z + 4$$

for  $z = 4 - 2i$

	A	B	C	D	E
1	<b>degree</b>	<b>coeff</b>		<b>re</b>	<b>im</b>
2	a0	4	z =	4	-2
3	a1	2			
4	a2	-5		<b>re</b>	<b>im</b>
5	a3	1	P(z) =	-256	-780
6	a4	2			
7					
8	{=POLYN(D2:E2;B2:B6)}				

Otherwise, if you want to compute a real polynomial for a real argument, e.g.  $z = 10$  - simply pass a single value

	A	B	C	D
1	<b>degree</b>	<b>coeff</b>		
2	a0	4	z =	10
3	a1	2		
4	a2	-5		
5	a3	1	P(z) =	20524
6	a4	2		
7				
8	{=POLYN(D2;B2:B6)}			
9				

## Xnumbers Tutorial

Example: compute the following complex polynomial

$$P(z) = 2z^4 + (1-i)z^3 - 5z^2 + (2-i)z + (4-5i)$$

for  $z = 4 - 2i$

	A	B	C	D	E	F
1	<b>degree</b>	<b>re</b>	<b>im</b>		<b>re</b>	<b>im</b>
2	a0	4	5	<b>z =</b>	4	-2
3	a1	2	-1			
4	a2	-5	0		<b>re</b>	<b>im</b>
5	a3	1	-1	<b>P(z) =</b>	-346	-795
6	a4	2	0			
7						
8		{=POLYN(E2:F2;B2:C6)}				

## Polynomial derivatives

---

**=DPOLYN(z, Coefficients, Order, [DgtMax])**

Computes the polynomial derivative at the value z.

$$P(z) = a_0 + a_1 z + a_2 z^2 + \dots a_n z^n$$

$$D_j(z) = \frac{d^j P(z)}{dz^j}$$

The parameter "Coefficients" is the (n+1) vector containing the polynomial coefficients from the lowest to the highest order.

This function accept also complex coefficients. In that case the parameter Coefficients is an (n+1 x 2) array.

The parameter "Order" sets the order of the derivative.

The optional parameter "DgtMax" set the number of the precision digits. If omitted, the function works in the faster double precision.

This function works also for complex arguments. In that case, z must be a complex number (two adjacent cells) and the function returns two values. To see both real and imaginary part, select two cells and give the CTRL+SHIFT+ENTER key sequence. If you press only ENTER, the function returns only its real part.

Example. Compute the derivatives of the following polynomial

$$P(z) = 3 + 2z + z^2 + z^3$$

For z= 3, we have:

	A	B	C	D	E	F	G
1	<b>degree</b>	<b>coeff</b>	<b>z =</b>	3			
2	a0	3	<b>P(z) =</b>	45	=DPOLYN(D1;\$B\$2:\$B\$5;0)		
3	a1	2	<b>P'(z) =</b>	35	=DPOLYN(D1;\$B\$2:\$B\$5;1)		
4	a2	1	<b>P''(z) =</b>	20	=DPOLYN(D1;\$B\$2:\$B\$5;2)		
5	a3	1	<b>P'''(z) =</b>	6	=DPOLYN(D1;\$B\$2:\$B\$5;3)		
6							

Example: calculate the 2nd derivative of the following complex polynomial at the point  $z = 4 - 2i$

$$P(z) = 2z^4 + (1-i)z^3 - 5z^2 + (2-i)z + (4-5i)$$

## Xnumbers Tutorial

	A	B	C	D	E	F
1	degree	re	im		re	im
2	a0	4	5	z =	4	-2
3	a1	2	-1			
4	a2	-5	0			
5	a3	1	-1	P''(z) =	290	-420
6	a4	2	0			
7						
8		{=DPOLYN(E2:F2;B2:C6;2)}				

With DPOLYN and POLYN it is very easy to implement, for example, the Newton's algorithm for finding the polynomial root with high precision

Example: find the real root of the following polynomial with Newton's algorithm

$$x^7 - 5x^6 + 64x^3 - 8000$$

The popular iterative Newton's formula is

$$x_{i+1} = x_i + \frac{p(x_i)}{p'(x_i)}$$

Starting from the point  $x = 10$ . Note that we cannot use the handy  $x = 0$ , because the derivative is zero

	A	B	C
1	<b>Polynomial root with Newton's method</b>		
2	$x^7 - 5x^6 + 64x^3 - 8000$		
3		=xsub(A5;xdiv(B5;C5))	=POLYN(A5;\$A\$2;30)
4			=DPOLYN(A5;\$A\$2;1;30)
5	x	p(x)	p'(x)
6	10	5056000	4019200
7	8.74203821656050955414012738854	1705019.38438633416493723834182	1607389.0879659353395148101859
8	7.68129978231871391768544159458	571754.68154068504090533718409	646931.960760449058718442189118
9	6.79750563359771918987884221808	189425.591769292484251118666596	264041.490046275503225345732376
10	6.0800972000364101396611583227	60951.5397999926501692109253746	111465.394901013712483397689024
11	5.53327690792583751912739957974	18147.9328785359838442503913358	51175.8040388466790518453855236
12	5.17865750879025778577387437683	4334.6352887553952480011017547	28430.483345306655462998174874
13	5.02619315438205307993125072528	548.68864761545931439992744713	21477.1425020876344296509538314
14	5.00064559154579004621392666485	13.19442476322699990934397961	20450.4610236244889342203264813
15	5.00000040194600622103029012186	0.00820975036142442331636681	20425.0158447161401449447377224
16	5.00000000000015590492153377256	0.00000000318435802232778355	20425.0000000061457720068620024
17	5.0000000000000000000000000002345	0.000000000000000000000047892	20425.0000000000000000000000924
18	5	0	20425

The exact digits caught by the algorithm, are shown in blue. Note the impressive acceleration. Try this example with 60 and more digits if you like.

## Polynomial coefficients

**=PolyTerms(Polynomial)**

Returns the vector of the polynomial coefficients

The argument is a polynomial string like "1-3x+5x^2 +x^5" in any order.

Example

	E	F	G	H	I	J	K
11							
12		<b>24-5x+3x^2+x^3</b>	24	-5	3	1	
13							
14		You can paste both vertical and horizontal vector	24				
15			-5				
16			3				
17			1				
18							

Note the braces { } in the formula. This indicates that the function return a vector. We must select the range before enter the function with "shift+ctrl+enter".

## Polynomial writing

### =PolyWrite(Coefficients, [variable])

It returns the polynomial string from its coefficients.

The first argument may be a (1 x n) vector or an (2 x n) array. In the last case, the first row indicates the coefficient position and the second row contains the correspondent coefficient value.

The second optional argument specifies the variable string (default is "x").

	A	B	C	D
10	0	1	6	
11	1000	200	1	
12				
13	<b>1000+200x+x^6</b>			
14				
15				

	A	B	C	D	E
18	-121	56	-12	3	1
19					
20	<b>-121+56t-12t^2+3t^3+t^4</b>				
21					
22					

Note that the second argument "t" must be insert as string, that is between quotes "..."

## Polynomial addition

### =PolyAdd(Poly1, Poly2)

Performs the addition of two polynomials.

The arguments are monovariate polynomial strings.

Example:

`PolyAdd("1-3x" , "-2-x+x^2") = "-1-4x+x^2" .`

## Polynomial multiplication

### =PolyMult(Poly1, Poly2)

Performs the multiplication of two polynomials

The arguments are monovariate polynomial strings.

## Xnumbers Tutorial

Example:

```
PolyMult("1-3x" , "-2+5x+x^2") = "-2+11x-14x^2-3x^3" .
```

$$(1-3x)(-2+5x+x^2) = -2+11x-14x^2-3x^3$$

## Polynomial subtraction

---

**=PolySub(Poly1, Poly2)**

Returns the difference of two polynomials  
The arguments are monovariate polynomial strings.

Example:

```
PolySub("1-3x" , "-2+5x+x^2") = "3-8x-x^2" .
```

## Polynomial division quotient

---

**=PolyDiv(Poly1, Poly2)**

Returns the quotient of two polynomials  
The arguments are monovariate polynomial strings.

Example:

```
PolyDiv("x^4-1" , "x^2-x-1") = "2+x+x^2" .
```

In fact:

$$x^4-1 = (x^2-x-1)(2+x+x^2) + 1+3x$$

## Polynomial division remainder

---

**=PolyRem(Poly1, Poly2)**

Returns the remainder of two polynomials  
The arguments are monovariate polynomial strings.

## Hermite's and Cebychev's polynomials

By the basic operations we can build any other polynomial.

Example: Calculate the first 9 Cebychev's and Hermite's polynomials

Cebysev's polynomials can be obtained by the iterative formula	Hermite's polynomials can be obtained by the iterative formula
$T_0 = 1$ , $T_1 = x$ $T_{n+1} = 2x \cdot T_n - T_{n-1}$	$H_0 = 1$ , $H_1 = x$ $H_{n+1} = 2x \cdot H_n - 2n \cdot H_{n-1}$

The two iterative formulas can be arrange as:

`=polysub(PolyMult("2x", Tn), Tn-1)`

`=polysub(PolyMult("2x", Hn), PolyMult(2*n, Hn-1))`

These functions are inserted from the cell B4 to B9 and C5 to C9

	A	B	C
1	<b>n</b>	<b>Hermite polynomials</b>	<b>Cebysev polynomials</b>
2	0	1	1
3	1	2x	x
4	2	-2+4x <sup>2</sup>	-1+2x <sup>2</sup>
5	3	-12x+8x <sup>3</sup>	-3x+4x <sup>3</sup>
6	4	12-48x <sup>2</sup> +16x <sup>4</sup>	1-8x <sup>2</sup> +8x <sup>4</sup>
7	5	120x-160x <sup>3</sup> +32x <sup>5</sup>	5x-20x <sup>3</sup> +16x <sup>5</sup>
8	6	-120+720x <sup>2</sup> -480x <sup>4</sup> +64x <sup>6</sup>	-1+18x <sup>2</sup> -48x <sup>4</sup> +32x <sup>6</sup>
9	7	-1680x+3360x <sup>3</sup> -1344x <sup>5</sup> +128x <sup>7</sup>	-7x+56x <sup>3</sup> -112x <sup>5</sup> +64x <sup>7</sup>
10	8	1680-13440x <sup>2</sup> +13440x <sup>4</sup> -3584x <sup>6</sup> +256x <sup>8</sup>	1-32x <sup>2</sup> +160x <sup>4</sup> -256x <sup>6</sup> +128x <sup>8</sup>
11	9	30240x-80640x <sup>3</sup> +48384x <sup>5</sup> -9216x <sup>7</sup> +512x <sup>9</sup>	9x-120x <sup>3</sup> +432x <sup>5</sup> -576x <sup>7</sup> +256x <sup>9</sup>
12			
13		<code>=polysub(PolyMult("2x",B10);PolyMult(2*A10;B9))</code>	<code>=polysub(PolyMult("2x",C10);C9)</code>

## Legendre's Polynomials

Legendre's polynomials can be obtained by the following well known iterative formula

$$P_n(x) = \frac{2n-1}{n} \cdot x \cdot P_{n-1}(x) - \frac{n-1}{n} \cdot P_{n-2}(x) \quad , \quad P_0 = 1 \quad , \quad P_1 = x$$

The first five polynomials are:

$$P_0 = 1 \quad , \quad P_1 = x \quad , \quad P_2 = \frac{1}{2}(3x^2 - 1) \quad , \quad P_3 = \frac{1}{2}(5x^3 - 3x) \quad , \quad P_4 = \frac{1}{8}(35x^4 - 30x^2 + 3)$$

The above formula is very popular, but from the point of view of numeric calculus has one disadvantage: its coefficients are decimal and this causes round-off errors leading inaccuracy for higher polynomial degree. It is convenient to rearrange the iterative formula to avoid fractional coefficients.

Let's assume that a Legendre's polynomial can be written as

## Xnumbers Tutorial

$$P_n(x) = \frac{1}{k_n} L_n(x) \quad (1a)$$

Where  $k_n$  is an integer number and  $L_n(x)$  is a polynomial having integer coefficients  
The Legendre's polynomial  $P_n(x)$  is completely defined by the couple of  $(k_n, L_n(x))$

Starting with

$$\begin{aligned} k_0 &= 1 & L_0 &= 1 \\ k_1 &= 1 & L_1 &= x \end{aligned}$$

We can show that the following iterative process, with  $n \geq 2$ , gives the couples  $(k_n, L_n(x))$

$$\begin{aligned} U_n(x) &= k_{n-2} \cdot (2n-1) \cdot x \\ a_n &= k_{n-1} \cdot (n-1) \\ V_n(x) &= U_n(x) \cdot L_{n-1}(x) - a_n \cdot L_{n-2}(x) \\ b_n &= n \cdot k_{n-1} \cdot k_{n-2} \\ c_n &= \text{GCD}(b_n, \text{coef}(V_n)) \end{aligned}$$

Where the *coef* operator returns the coefficients vector of the polynomial  $V_n(x)$ , and the GCD is the greatest common divisor.

Simplifying, we get, finally the couple  $(k_n, L_n(x))$

$$\begin{aligned} k_n &= \frac{b_n}{c_n} \\ L_n(x) &= \frac{1}{c_n} V_n(x) \end{aligned}$$

This iterative algorithm, working only with integer values, is adapted to build Legendre's polynomials with high degree.

Let's see how to arrange a worksheet for finding Legendre's polynomial

In the first column we insert the degree  $n$ , beginning from 0 to 2, for the moment  
In the last two columns "k" and "L(x)" we have added the starting values.

	A	B	C	D	E	F	G	H
1	<b>Legendre's Polynomials</b>							
2				=xMCD(polyterms(D6);E6)			=E6/F6	
3	<b>n</b>	<b>U(x)</b>	<b>a</b>	<b>V(x)</b>	<b>b</b>	<b>c</b>	<b>k</b>	<b>L(x)</b>
4	0						1	1
5	1						1	x
6	2	3x	1	-1+3x^2	2	1	2	-1+3x^2
7								
8								
9								

The row 6 contains all the functions that the process needs.

In particular we note:

The function polyterms(D6) gives the coefficients vectors [-1, 0, 3] of  $V(x) = -1+3x^2$

The function xMCD returns the greatest common divisor of [-1, 0, 3, 2]  $\Rightarrow 1$



## Xnumbers Tutorial

Select the row 6 and drag it down. We generate the Legendre's polynomial in the form (1a)

	A	B	C	D	E	F	G	H
1	<b>Legendre's Polynomials</b>							
2								
3	<b>n</b>	<b>U(x)</b>	<b>a</b>	<b>V(x)</b>	<b>b</b>	<b>c</b>	<b>k</b>	<b>L(x)</b>
4	0						1	1
5	1						1	x
6	2	3x	1	-1+3x^2	2	1	2	-1+3x^2
7	3	5x	4	-9x+15x^3	6	3	2	-3x+5x^3
8	4	14x	6	6-60x^2+70x^4	16	2	8	3-30x^2+35x^4
9	5	18x	32	150x-700x^3+630x^5	80	10	8	15x-70x^3+63x^5
10	6	88x	40	-120+2520x^2-7560x^4+5544x^6	384	24	16	-5+105x^2-315x^4+231x^6

Here is a table of Legendre's polynomials obtained with the above method

<b>n</b>	<b>k</b>	<b>L(x)</b>
0	1	1
1	1	x
2	2	-1+3x^2
3	2	-3x+5x^3
4	8	3-30x^2+35x^4
5	8	15x-70x^3+63x^5
6	16	-5+105x^2-315x^4+231x^6
7	16	-35x+315x^3-693x^5+429x^7
8	128	35-1260x^2+6930x^4-12012x^6+6435x^8
9	128	315x-4620x^3+18018x^5-25740x^7+12155x^9
10	256	-63+3465x^2-30030x^4+90090x^6-109395x^8+46189x^10
11	256	-693x+15015x^3-90090x^5+218790x^7-230945x^9+88179x^11
12	1024	231-18018x^2+225225x^4-1021020x^6+2078505x^8-1939938x^10+676039x^12
13	1024	3003x-90090x^3+765765x^5-2771340x^7+4849845x^9-4056234x^11+1300075x^13

We can also extract a table of Legendre's coefficients by the Polyterms() function

## Polynomial shift

**=PolyShift(Poly, x0)**

Performs the polynomial translation of  $x_0$ ,

The argument "Poly" can be the polynomial strings or the vector of polynomial coefficients.

This function returns the coefficient vector of the translated polynomial.

If you select one cell, the output will be a polynomial string

Example:

Given the polynomial:

$$188784918 - 47389623 x + 4952504 x^2 - 275809 x^3 + 8633 x^4 - 144 x^5 + x^6$$

substituting x with  $z+24$ , we have

$$-18 + 9z - 16z^2 - z^3 - 9z^4 + z^6$$

	A	B	C	D	E	F	G
1							
2	<b>degree</b>	<b>Coefficients</b>				<b>degree</b>	<b>Coefficients</b>
3	0	188784918				0	-18
4	1	-47389623		24		1	9
5	2	4952504				2	-16
6	3	-275809		Shift		3	-1
7	4	8633				4	-7
8	5	-144				5	0
9	6	1				6	1
10							
11							
12							

This function is useful for transforming polynomial for reducing the coefficients amplitude and improving the precision of rootfinder methods. In this example we work with coefficients of two maximum digits, instead of 9 digits. We note also that the second polynomial, having the second coefficient = 0, is centered. Its roots are the same of the given polynomial, translated of 24, but can be factorize much better. In fact, we have

$$(z^2 - z + 1)(z^2 + z + 2)(z^2 - 9)$$

## Polynomial center

**=PolyCenter(Coefficients)**

Returns the center of the polynomial roots circle

The argument specifies the vector of the polynomial coefficients in the following order:

$$[a_0, a_1, a_2 \dots a_n]$$

It can also be a polynomial string

if  $x_1, x_2, \dots, x_n$  are roots of polynomial the center **Bx** is defined as:

$$B_x = \frac{x_0 + x_1 + x_2 \dots + x_n}{n} = \frac{-a_{n-1}}{n}$$

## Polynomial roots radius

### =PolyRadius(Coefficients)

Returns the approximated radius of the polynomial roots circle.

The argument is the vector of the polynomial coefficients in the following order:

$$[a_0, a_1, a_2 \dots a_n]$$

It can also be a polynomial string

If  $z_i$  are the roots of a polynomial, the radius is defined as:

$$R = \max_{i=1 \dots n}(|z_i|)$$

The circle of root is very useful for locating all the roots of a polynomial. For example, given the following 9 degree polynomial.

degree	coefficients
a0	-3098250
a1	4116825
a2	-2427570
a3	916272
a4	-244674
a5	46934
a6	-6430
a7	608
a8	-36
a9	1

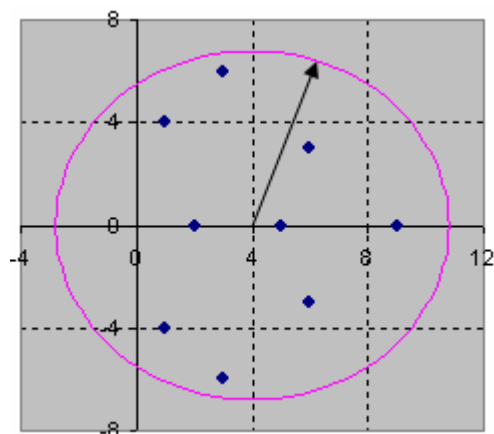
	A	B	C	D
1	degree	coefficients	radius	center
2	a0	-3098250	6.7882251	4
3	a1	4116825		
4	a2	-2427570	=PolyRadius(B2:B11)	
5	a3	916272	=PolyCenter(B2:B11)	
6	a4	-244674		
7	a5	46934		
8	a6	-6430		
9	a7	608		
10	a8	-36		
11	a9	1		

The center = 4 and the radius  $\cong 6.8$

We can draw the circle containing, with high probability, all polynomial roots

We know that the roots of this polynomial are:

x real	x imm
9	0
5	0
2	0
3	-6
3	6
1	-4
1	4
6	-3
6	3



We have to point out that this method is probabilistic. It means that the most part of the roots are found inside the circle but it is also possible to find some roots outside the circle with 1% of probability.

## Xnumbers Tutorial

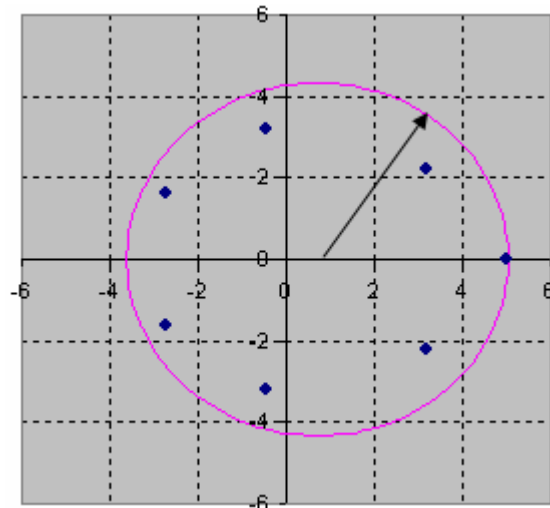
Example: compute the root circle of the polynomial:  $x^7-5x^6+64x^3-8000$

radius  $\cong 4.331$

center  $\cong 0.714$

The roots are:

x real	x imm
-2.7429701	1.6132552
-2.7429701	-1.6132552
-0.4369651	3.2182957
-0.4369651	-3.2182957
3.17993518	2.2060806
3.17993518	-2.2060806
5	0



## Polynomial building from roots

**=PolyBuild(Roots, [Variable])**

Builds a polynomial from its roots. Argument "Roots" is an (n x 2) array, contains the polynomial roots. It can be a vector for real roots.

This function returns the coefficient vector of the polynomial.

If you select one cell, the output will be a polynomial string

Complex roots for real polynomial:

	A	B	C
1	Xreal	Yimm	P(x)
2	1	-1	-4+6x-4x^2+x^3
3	1	1	
4	2	0	=PolyBuild(A2:B4)
5			

Multiple roots:

	A	B	C
1	Xreal	Yimm	P(x)
2	-1	0	1+4x+6x^2+4x^3+x^4
3	-1	0	
4	-1	0	=PolyBuild(A2:B4)
5	-1	0	
6			

Complex roots for complex polynomial

If the complex roots are not symmetrical, the polynomial has both real and imaginary part. This function returns both, simply as a vector (2 x 1).

	A	B	C
1	Xreal	Yimm	P(x)
2	-1	0	x+3x^2+3x^3+x^4
3	-1	0	-1-3x-3x^2-x^3
4	-1	0	
5	0	1	{=PolyBuild(A2:B4)}
6			

Zero roots

If you want a polynomial with multiple zero roots, simply repeat many couple [0, 0] as they need.

	A	B	C
1	Xreal	Yimm	P(x)
2	0	0	z^2+2z^3+z^4
3	0	0	
4	-1	0	=PolyBuild(A2:B4; "z")
5	-1	0	

This function can return the vector of polynomial coefficients if you select more than two vertical cells. It is very useful for higher degree polynomial

## Xnumbers Tutorial

	A	B	C	D	E
1	<b>x real</b>	<b>x imm</b>		<b>degree</b>	<b>coefficients</b>
2	9	0		a0	-3098250
3	5	0		a1	4116825
4	2	0		a2	-2427570
5	3	-6		a3	916272
6	3	6		a4	-244674
7	1	-4		a5	46934
8	1	4		a6	-6430
9	6	-3		a7	608
10	6	3		a8	-36
11	{=PolyBuild(A2:B10)}			a9	1
12					

In this example we get the 10 coefficients of the 9<sup>th</sup> degree polynomial having the 9 roots in the range A2:B10

	A	B	C	D	E
1	<b>x real</b>	<b>x imm</b>	<b>degree</b>	<b>coeff. re.</b>	<b>coeff im.</b>
2	9	0	a0	540	-405
3	5	0	a1	-393	351
4	2	1	a2	127	-79
5	3	-6	a3	-19	5
6			a4	1	0
7					

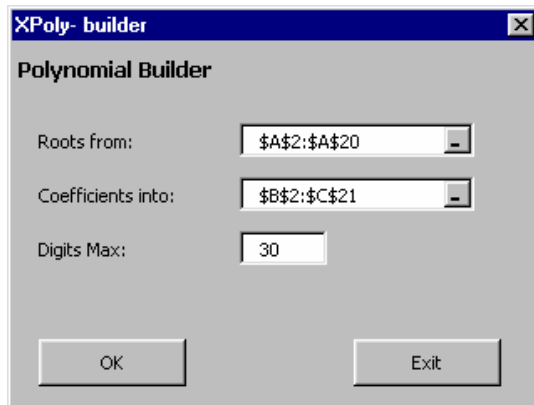
If complex roots are not conjugate, the polynomial has complex coefficients. This function can return also the imaginary column of the coefficients, simply selecting two columns

## Polynomial building with multi-precision

### PolyBuildCfx()

This macro generate the polynomial coefficients from the given roots.

This macro works like the function PolyBuild except that it works in multi-precision. It is very useful for high degree polynomial, when the coefficients become longer than 15 digits.



For using this macro select the range that contains the roots.

Then, start the macro. Choose the digits precision (default=30) and the range you want to paste the coefficients (default is the range at the right side of the roots range selected).

In the following table we have calculated the coefficient of the polynomial having as roots the first 19 integer numbers. That is:

$$x_1 = 1, x_2 = 2, x_3 = 3, \dots, x_{19} = 19$$

Roots	PolybuildCfx (30 digits)	PolyBuild	Diff.
1	-121645100408832000	-121645100408832000	0
2	431565146817638400	431565146817638000	400
3	-668609730341153280	-668609730341153000	-280
4	610116075740491776	610116075740492000	-224
5	-371384787345228000	-371384787345228000	0
6	161429736530118960	161429736530119000	-40
7	-52260903362512720	-52260903362512700	-20
8	12953636989943896	12953636989943900	-4
9	-2503858755467550	-2503858755467550	0
10	381922055502195	381922055502195	0
11	-46280647751910	-46280647751910	0
12	4465226757381	4465226757381	0
13	-342252511900	-342252511900	0
14	20692933630	20692933630	0
15	-973941900	-973941900	0
16	34916946	34916946	0
17	-920550	-920550	0
18	16815	16815	0
19	-190	-190	0
	1	1	0

As we can see there are a little difference (digits in red) between the exact coefficients computed by this macro PolyBuildCfx (multiprecision arithmetic with 30 digits) and those returned by the function PolyBuild (standard double precision 32-bit).

## Polynomial solving

### =PolySolve (Polynomial)

This function returns the roots of a given real polynomial using the Jenkins-Traub algorithm.

$$a_0 + a_1x + a_2x^2 + \dots a_nx^n$$

The arguments can be a monovariate polynomial strings like "X^2+3x+2" or a vector of coefficients

This function returns an (n x 2) array.

It uses the same algorithm of the RootfinderJT macro. It works fine with low-moderate degree polynomials, typically from 2° till 10° degree. For higher degree it is more convenient to use the macro.

Example. Find all roots of the given 10 degree polynomial

	A	B	C	D	E
1	<b>Degree</b>	<b>Coefficients</b>		<b>REAL</b>	<b>IMM</b>
2	a0	3628800		1	0
3	a1	-10628640		2	0
4	a2	12753576		3	0
5	a3	-8409500		4	0
6	a4	3416930		5	0
7	a5	-902055		6	0
8	a6	157773		7	0
9	a7	-18150		8	0
10	a8	1320		9	0
11	a9	-55		10	0
12	a10	1			
13					
14		=PolySolveJT(B2:B12)			
15					

## Integer polynomial

### =PolyInt(Polynomial)

This function returns a polynomial with integer coefficients having the same roots of the given polynomial. This transformation is also known as "denormalization" and can be useful when the coefficients of the normalized polynomial are decimal.

Example: Eliminate decimal coefficients from the following polynomial:

$$-0.44 + 2.82x - 3.3x^2 + x^3$$

To eliminate decimal coefficients we denormalize the polynomial

$$-22 + 141x - 165x^2 + 50x^3 = \text{PolyInt}("-0.44 + 2.82x - 3.3x^2 + x^3")$$

Take care with the denormalization because the coefficients became larger and the computation can lose accuracy. See the example below

## Xnumbers Tutorial

The following polynomials have the same root  $x = 11/10$ :

$$P_b(x) = -2.4024 + 10.1524x - 17.1x^2 + 14.35x^3 - 6x^4 + x^5$$

$$P_a(x) = -6006 + 25381x - 42750x^2 + 35875x^3 - 15000x^4 + 2500x^5$$

If we compute both polynomials for  $x = 11/10$ , with standard double precision we get:

$$P_a(1.1) = -2.664E-15$$

$$P_b(1.1) = 4.547E-12$$

As we can see, the first value, obtained by the decimal polynomial, is 1000 times more precise than the one obtained by the integer polynomial

## Polynomial interpolation

---

**=PolyInterp (x, xi, yi, [DgtMax])**

**=PolyInterpCf (xi, yi, [DgtMax])**

These functions perform the polynomial interpolation

The first function performs the interpolation of a given set of points (xi,yi), and returns the value at the point x. If the parameter x is literal, like "x", the function returns the interpolation polynomial expression.

Input parameters xi and yi are vectors.

The optional parameter DgtMax sets the max digits in multiprecision arithmetic. If omitted or zero, the functions work in faster standard double precision.

The second function returns an array containing the coefficients of polynomial interpolation

These functions use the following popular Newton's formula:

$$p(x) = y_1 + \sum_{m=1}^n \left( D(x_1, \dots, x_m) \prod_{j=1}^{m-1} (x - x_j) \right)$$

Where D are the "divided differences", given by the following recursive formulas:

$$D(x_1, x_2) = \frac{y_1 - y_2}{x_1 - x_2}, \quad D(x_2, x_3) = \frac{y_2 - y_3}{x_2 - x_3}, \dots$$

$$D(x_1, x_2, x_3) = \frac{D(x_1, x_2) - D(x_2, x_3)}{x_1 - x_3}, \dots$$

$$D(x_1, \dots, x_m) = \frac{D(x_1, \dots, x_{m-1}) - D(x_2, \dots, x_m)}{x_1 - x_m}$$

## Sub-tabulation

Interpolation method is very useful to generate a sub-tabulation from a given table.

Usually interpolation is used when we have few exact knots, and we want to approximate the value between two consecutive knots. On the contrary, when we have many values affected by relevant random errors (experimental samples) is better to



## Xnumbers Tutorial

use the regression. The main differences is that interpolation curve always crosses for all knots, the regression line may not cross for any given knots.

Example:

x	y
0	0.5
0.5	0.7
1	1.2
1.5	1.2
2	1.3
2.5	2.2

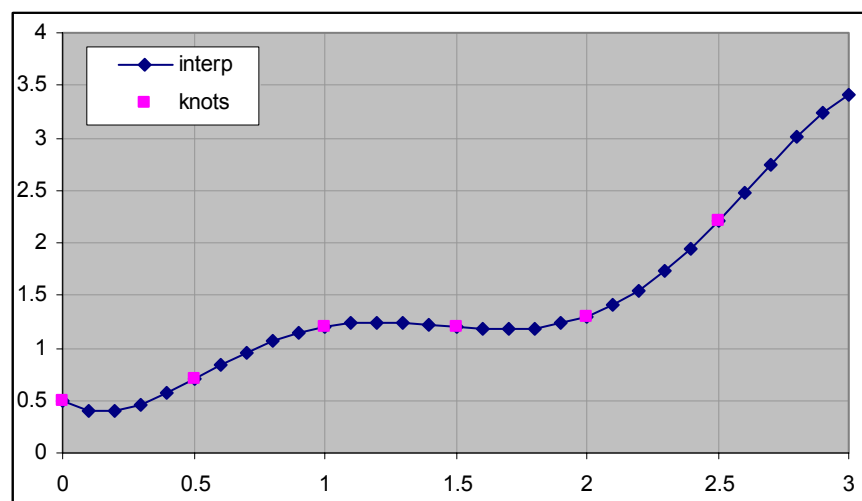
Sub-tabulation problem.

Given the following table we want to generate a new table with step = 0.1 and for  $0 \leq x \leq 3$

	A	B	C	D	E
1	x	y		x	y
2	0	0.5		0	0.5
3	0.5	0.7		0.1	0.597363
4	1	1.2		0.2	0.595622
5	1.5	1.2		0.3	0.462458
6	2	1.3		0.4	0.571117
7	2.5	2.2		0.5	0.7
8				0.6	0.832243
9					=PolyInterp(D2;\$A\$2:\$A\$7;\$B\$2:\$B\$7)
10				0.8	1.060538

In the cell E2 insert the function PolyInterp as in figure. Select the cell E2 and drag it down to fill all the cells that you need.

The following graph shows the interpolate points (blue) and the given knots (pink)



### Interpolation Polynomial string

We can obtain the interpolation polynomial expression, simply passing a generic letter (Ex: "x") to the argument x. We get:

```
=PolyInterp("x";A2:A7;B2:B7) = 0.5-x+6.93333333333333x^2-6.9x^3+2.66666666666667x^4-0.346666666666667x^5
```

If we do not want decimal values, use the function **PolyInt()**. We get:

```
75-150x+1040x^2-1035x^3+400x^4-52x^5
```

Remember that this polynomial is not the same of the above interpolation polynomial. We must divide it for an adapt coefficient, that can be computed dividing a coefficient of the second polynomial (e.g: 75) for the corresponding coefficient of the first one (e.g: 0.5). We get

## Xnumbers Tutorial

$$M = 75 / 0.5 = 150$$

So the final interpolation polynomial can be written as:

$$P(x) = 1/150 * (75 - 150x + 1040x^2 - 1035x^3 + 400x^4 - 52x^5)$$

## Polynomial System of 2<sup>nd</sup> degree

### =SYSPOLY2(Poly1, Poly2)

Solves a system of two 2<sup>nd</sup> degree polynomials.

$$\begin{cases} a_{11}x^2 + a_{12}xy + a_{13}y^2 + a_{14}x + a_{15}y + a_{10} = 0 \\ a_{21}x^2 + a_{22}xy + a_{23}y^2 + a_{24}x + a_{25}y + a_{20} = 0 \end{cases}$$

It returns a (4 x 4) array containing the four solutions.

The parameters *Poly1* and *Poly2* can be coefficients vectors or polynomials strings

The coefficients must be passed in the same order of the above equation.

Polynomial strings, on the contrary, can be written in any order. Examples of 2<sup>nd</sup> degree x-y polynomials strings are:

$$13 + x + y^2 - y + x^2 + 2x * y$$

$$x^2 + y^2 - 10$$

$$4x^2 + 8x * y + y^2 + 2x - 2$$

Note: the product symbol “\*” can be omitted except for the x\*y mixed term

A 2<sup>nd</sup> degree system can have up to four solutions. It can also have no solution (impossible) or even infinite solutions (undetermined). The function returns #N/D if a solution is missing

**Example:** solve the following system

$$\begin{cases} x^2 + 2xy + y^2 + x - y = 0 \\ x^2 + y^2 - 10 = 0 \end{cases}$$

Using SYSPOLY2 the solutions – real or complex – can be obtained in a very quick way

Real solutions represent the intersection point of the curve poly1 and poly2.

They are:  $P_1 = (-3, 1)$  ,  $P_2 = (-1, 3)$

	A	B	C	D
1				
2	Poly1 : x^2+2x*y+y^2+x-y			
3	Poly2 : x^2+y^2-10			
4				
5	X real	X im	Y real	Y im
6	2.5	1.1180340	-2.5	1.1180340
7	2.5	-1.1180340	-2.5	-1.1180340
8	-3	-0	1	0
9	-1	-0	3	0
10	{=SYSPOLY2(B2:B3)}			

The system has also two complex solutions that have not a geometrical representation

$$P_3 = (2.5 + j 1.118034, -2.5 + j 1.118034) , P_4 = (2.5 - j 1.118034, -2.5 - j 1.118034)$$

The degree of the given system is 4

## Xnumbers Tutorial

**Example:** solve the following system

$$\begin{cases} xy - 1 = 0 \\ 2xy + y^2 + x + y - 1 = 0 \end{cases}$$

The apparent degree of the system is  $2 \times 2 = 4$

	A	B	C	D	E	F	G	H	I	J	K
1	$x^2$	$xy$	$y^2$	$x$	$y$	$c$		X real	X im	Y real	Y im
2	0	1	0	0	0	-1		8.5E-18	-1	8.5E-18	1
3	0	2	1	1	1	-1		8.5E-18	1	8.5E-18	-1
4								-1	0	-1	0
5		{=SYSPOLY2(A2:F2;A3:F3)}						#N/D	#N/D	#N/D	#N/D
6											

As we can see, the function SYSPOLY2 returns only three solutions: one real and two complex.

$$P_1 = (-1, -1), \quad P_2 = (-j, j), \quad P_3 = (j, -j)$$

Thus, the actual system degree is 3.

## Bivariate Polynomial

**=POLYN2(Polynomial, x, y, [DgtMax])**

Returns the bivariate polynomial value, real or complex, at the point x, y.

The parameter "Polynomial" is an expression strings. Valid examples of x y polynomial strings are:

$13 + x + y^2 - y + x^2 + 2x \cdot y$  ,  $x^2 + y^2 - 10$  ,  $8x \cdot y + y^2 + 2x - 2$  ,  $10 + 4x^6 + x^2 \cdot y^2$

Note: the product symbol "\*" can be omitted except for the  $x \cdot y$  mixed terms

The third optional parameter is used for multi precision computing. If you set any number from 1 to 200, the computation is performed in multiprecision.

The variables x, y can be real or complex. The function can return real or complex numbers. Select two cells if you want to see the imaginary part and give the CTRL+SHIFT+ENTER sequence

Example: Compute the polynomial

$$P = x^2 + 2xy + y^2 + x - y$$

at the point

$$x = (2.5 + j \ 1.11803398874989)$$

$$y = (-2.5 + j \ 1.11803398874989)$$

And verify that it is a good approximation of the polynomial root

	A	B	C
1			
2	P(x, y) = $x^2 + 2x \cdot y + y^2 + x - y$		
3			
4		real	im
5	x =	2.5	1.118033989
6	y =	-2.5	1.118033989
7	P =	-3.90799E-14	6.66134E-15
8			
9	{=POLYN2(B2;B5:C5;B6:C6)}		

## Partial fraction decomposition

---

Partial fraction decomposition is the process of rewriting a rational expression as the sum of a quotient polynomial plus partial fractions. If the rational expression is proper - thus the degree of numerator polynomial is lower than the denominator - the quotient will be zero and it remains only the partial fractions terms. A polynomial with real coefficients can be factored into a product of powers of linear and quadratic factors: the linear factors are taken by real roots while the quadratic factors are taken by complex roots.

$$\frac{N(x)}{D(x)} = Q(x) + \frac{R(x)}{D(x)} = Q(x) + \sum F_i$$

where each  $F_i$  is a fractions of the form

$$\frac{A_1}{x+p} + \frac{A_2}{(x+p)^2} + \dots + \frac{A_m}{(x+p)^m}$$

or

$$\frac{B_1x+C_1}{x^2+bx+c} + \frac{B_2x+C_2}{(x^2+bx+c)^2} + \dots + \frac{B_mx+C_m}{(x^2+bx+c)^m}$$

being  $m$  is the multiplicity of the correspondent root

The denominators is determined from the poles, thus the roots of the denominator  $D(x)$ . In fact,  $p$  is just a real root of  $D(x)$ , while the quadratic factor can be obtained from the complex root using the following relation

$$\alpha \pm i\beta \Rightarrow b = -2\alpha, \quad c = \alpha^2 + \beta^2 \quad (1)$$

Many calculators and computer algebra systems, are able to factor polynomials and split rational functions into partial fractions. But also in Excel a solution can be arranged with the aid of Xnumbers functions. Let's see

**Real single poles.** Find the fraction decomposition of the following rational fraction

$$\frac{N(x)}{D(x)} = \frac{3x^3 + 276x^2 - 1433x + 1794}{x^4 - 15x^3 + 65x^2 - 105x + 54}$$

First of all, we try to find the roots of the denominator using, for example, the function polysolve. We find that the roots are  $p_i = [1, 2, 3, 9]$ . They are all real with unitary multiplicity, therefore the fraction expansion will be

$$\frac{N(x)}{D(x)} = \frac{A_1}{x+p_1} + \frac{A_2}{x+p_2} + \frac{A_3}{x+p_3} + \frac{A_4}{x+p_4}$$

where  $p_i$  are the roots and  $A_i$  are unknown

Several methods exist for solving the fraction coefficients  $A_i$ . One of the most straight and elegant is the Heaviside's formula that, for a real single root, is simple:

$$A_i = \frac{N(p_i)}{D'(p_i)}$$

where  $D'$  is the derivative of  $D$

A possible arrangement in Excel is the following

	A	B	C	D	E	F	G	H	I	J
2		coefficients		Poles						
3		N(x)	D(x)	re	im		N(x)	D'(x)	A i	
4		1794	54	1	0		640	-16	-40	
5		-1433	-105	2	0		56	7	8	
6		276	65	3	0		60	-12	-5	
7		3	-15	9	0		13440	336	40	
8			1							
9										
10										
11										
12										

{=polysolve(C4:C8)}

=polyn(D7,\$B\$4:\$B\$7)

=G7/H7

=dpolyn(D7,\$C\$4:\$C\$8,1)

Therefore, the requested decomposition is

$$\frac{3x^3 + 276x^2 - 1433x + 1794}{x^4 - 15x^3 + 65x^2 - 105x + 54} = -\frac{40}{x+1} + \frac{8}{x+2} - \frac{5}{x+3} + \frac{40}{x+9}$$

You can prove yourself that this expression is an identity, thus always true for every x, except the poles.

**Complex single poles.** Find the fraction decomposition of the following rational fraction

$$\frac{N(x)}{D(x)} = \frac{-x^3 - 21x^2 + 52x + 123}{x^4 - 2x^3 - 29x^2 - 42x + 650}$$

First of all, we try to find the roots of the denominator using, for example, the function polysolve. We find that the roots are  $p = \{ 5 \pm 2i, -4 \pm 3i \}$ . They are complex with unitary multiplicity, therefore the fraction expansion will be

$$\frac{N(x)}{D(x)} = \frac{B_1x + C_1}{x^2 + b_1x + c_1} + \frac{B_2x + C_2}{x^2 + b_2x + c_2}$$

where  $b_i$  and  $c_i$ , calculated by the (1), are  $b_1 = -10$ ,  $c_1 = 26$ ,  $b_2 = 8$ ,  $c_2 = 25$

The coefficients  $B_i$  and  $C_i$  are unknown. For solving them we used here the so called undetermined coefficients method

Renamed, for simplicity:

$$D_1(x) = x^2 + b_1x + c_1, \quad D_2(x) = x^2 + b_2x + c_2$$

The fraction expansion may be rewritten as

$$\frac{N(x)}{D(x)} = \frac{B_1x}{D_1(x)} + \frac{C_1}{D_1(x)} + \frac{B_2x}{D_2(x)} + \frac{C_2}{D_2(x)}$$

Giving 4 different values to x, the above relation provides 4 linear equations in  $B_1, C_1, B_2, C_2$ , that can be easily solved. We can choose any value that we want; for example  $x_i = \{ 0, 1, 2, 3 \}$  and we get the following linear system

0	1/26	0	1/25	x	=	123/650
1/17	1/17	1/34	1/34			9/34
1/5	1/10	2/45	1/45			3/10
3/5	1/5	3/58	1/58			63/290

Solving this linear system by any method that we like, we get the solution

$$[B_1, C_1, B_2, C_2] = [-2, 7, 1, -2]$$

Substituting these values, we have finally the fraction decomposition

$$\frac{-x^3 - 21x^2 + 52x + 123}{x^4 - 2x^3 - 29x^2 - 42x + 650} = \frac{-2x + 7}{x^2 - 10x + 26} + \frac{x - 2}{x^2 + 8x + 25}$$

You can prove yourself that this expression is an identity, thus always true for every x

In Excel a possible arrangement for solving this problem is a bit more complicated than the previous one. Let's see. First of all we compute the roots with the function Polysolve, then we compute the trinomials D1(x) and D2(x) by the formulas (1)

	B	C	D	E	F	G	H
1							
2	coefficients		Poles				
3	N(x)	D(x)	re	im		D1(x)	D2(x)
4	123	650	5	1		26	25
5	52	-42	5	-1		-10	8
6	-21	-29	-4	3		1	1
7	-1	-2	-4	-3			
8		1					
9							
10							

Formulas shown in the image:

- `=D4^2+E4^2` (in cell F4)
- `=-2*D4` (in cell F5)
- `=polysolve(C4:C8)` (in cell F7)

Then we compute the polynomials N, D, D1, D2 for each values of x by the function polyn. We get the 4x5 table at the right

	B	C	D	E	F	G	H	I	J	K	L	M	N
2	coefficients		Poles										
3	N(x)	D(x)	re	im		D1(x)	D2(x)		x	N	D	D1	D2
4	123	650	5	1		26	25		0	123	650	26	25
5	52	-42	5	-1		-10	8		1	153	578	17	34
6	-21	-29	-4	3		1	1		2	135	450	10	45
7	-1	-2	-4	-3					3	63	290	5	58
8		1											
9													
10													

Formula shown in the image: `=polyn(J7,$B$4:$B$7)` (in cell M7)

From the value-table we get the complete system matrix

	J	K	L	M	N	O	P	Q	R	S	T
3	x	N	D	D1	D2		x / D1	1 / D1	x / D2	1 / D2	N / D
4	0	123	650	26	25		0	0.0385	0	0.04	0.1892308
5	1	153	578	17	34		0.0588	0.0588	0.0294	0.0294	0.2647059
6	2	135	450	10	45		0.2	0.1	0.0444	0.0222	0.3
7	3	63	290	5	58		0.6	0.2	0.0517	0.0172	0.2172414
8											
9											
10											

Formulas shown in the image:

- `=J7/M7` (in cell P7)
- `=1/M7` (in cell Q7)
- `=J7/N7` (in cell R7)
- `=1/N7` (in cell S7)
- `=K7/L7` (in cell T7)

That can be solved by any method that you want. For example by matrix inversion

	O	P	Q	R	S	T	U	V	W
3		x / D1	1 / D1	x / D2	1 / D2	N / D			
4		0	0.0385	0	0.04	0.1892308		B1	-2
5		0.0588	0.0588	0.0294	0.0294	0.2647059		C1	7
6		0.2	0.1	0.0444	0.0222	0.3		B2	1
7		0.6	0.2	0.0517	0.0172	0.2172414		C2	-2
8									
9									
10									

Formula shown in the image: `=MMULT(MINVERSE(P4:S7),T4:T7)` (in cell U4)

## Orthogonal Polynomials

Orthogonal polynomials are a class of polynomials following the rule:

$$\int_a^b w(x) p_m(x) p_n(x) dx = \delta_{mn} c_n,$$

Where  $m$  and  $n$  are the degrees of the polynomials,  $w(x)$  is the weighting function, and  $c(n)$  is the weight.  $\delta_{mn}$  is the Kronecker's delta function being 1 if  $n = m$  and 0 otherwise.

The following table synthesizes the interval  $[a, b]$ , the  $w(x)$  functions and the relative weight  $c(n)$  for each polynomials family

polynomial	interval	$w(x)$	$c_n$
Chebyshev polynomial of the first kind	$[-1, 1]$	$(1 - x^2)^{-1/2}$	$\begin{cases} \pi & \text{for } n = 0 \\ \frac{1}{2}\pi & \text{otherwise} \end{cases}$
Chebyshev polynomial of the second kind	$[-1, 1]$	$\sqrt{1 - x^2}$	$\frac{1}{2}\pi$
Gegenbauer polynomial	$[-1, 1]$	$(1 - x^2)^{\alpha-1/2}$	$\begin{cases} \frac{2^{1-2\alpha}\pi\Gamma(n+2\alpha)}{n!(n+\alpha)[\Gamma(\alpha)]^2} & \text{for } \alpha \neq 0 \\ \frac{2\pi}{n^2} & \text{for } \alpha = 0. \end{cases}$
Hermite polynomial	$(-\infty, \infty)$	$e^{-x^2}$	$\sqrt{\pi} 2^n n!$
Jacobi polynomial	$(-1, 1)$	$(1 - x)^\alpha (1 + x)^\beta$	$h_n$
Laguerre polynomial	$[0, \infty)$	$e^{-x}$	1
generalized Laguerre polynomial	$[0, \infty)$	$x^k e^{-x}$	$\frac{(n+k)!}{n!}$
Legendre polynomial	$[-1, 1]$	1	$\frac{2}{2n+1}$

Where

$$h_n \equiv \frac{2^{\alpha+\beta+1}}{2n + \alpha + \beta + 1} \frac{\Gamma(n + \alpha + 1)\Gamma(n + \beta + 1)}{n!\Gamma(n + \alpha + \beta + 1)},$$

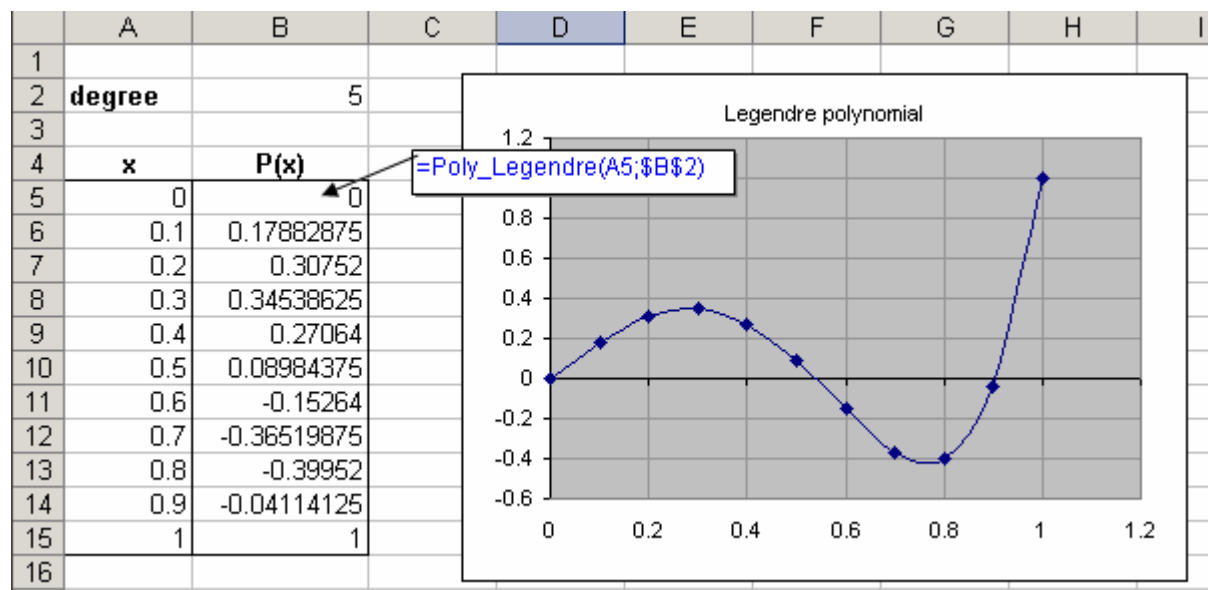
## Orthogonal Polynomials evaluation

This set of functions<sup>8</sup> calculate the orthogonal polynomials and their derivatives at the given point. They return two values: the first one is the polynomial value, the second is its 1st derivative. If you want to see both values select two adjacent cells and give the CTRL+SHIFT+ENTER sequence. If you give ENTER, you will get only the polynomial value

Function Poly_ChebyshevT(x, n)	Chebyshev polynomial of the first kind
Function Poly_ChebyshevU(x, n)	Chebyshev polynomial of the second kind
Function Poly_Gegenbauer(a, x, n)	Gegenbauer polynomial
Function Poly_Hermite(x, n)	Hermite polynomial
Function Poly_Jacobi(a, b, x, n)	Jacobi polynomial
Function Poly_Laguerre(x, n, m)	Laguerre generalized polynomial
Function Poly_Legendre(x, n)	Legendre polynomial

### Example:

Tabulate the Legendre polynomial of 6<sup>th</sup> degree, for  $0 \leq x \leq 1$ , with step  $h = 0.1$



As we can see we have insert **Poly\_Legendre** as a standard function, because in this exercise we do not need the derivative information

**Example.** Find the greatest zero of the 5<sup>th</sup> degree Legendre polynomial  
We can use the Newton-Raphson method, starting from  $x = 1$ , as shown in the following sheet arrangement

<sup>8</sup> Many thanks to Luis Isaac Ramos Garcia for his great contribution in developing this software



## Xnumbers Tutorial

	A	B	C	D	E
1					
2	degree =	5	{=Poly_Legendre(A5;\$B\$2)}		
3					
4	x	P(x)	P'(x)		
5	1	1	15		
6	0.9333333333	0.21336	8.887444444		
7	0.909326432	0.021964519	7.09107837		
8	0.906228946	0.000337416	6.873752439		
9	0.906179858	8.38968E-08	6.870334331		
10	0.906179846	5.16718E-15	6.870333481		
11	0.906179846	-7.60676E-17	6.870333481		
12					
13		=A5-B5/C5			
14					

Both polynomial and derivative are obtained from the [Poly\\_Legendre](#) simply selecting the range B5:C5 and pasting the function as array with CTRL+SHIFT+ENTER sequence

The other cells are filled simply by dragging down the range B5:C5

### Function Poly\_ChebyshevT(x, [n])

### Function Poly\_ChebyshevU(x, [n])

Evaluate the Chebyshev orthogonal polynomial of 1st and 2nd kind

Parameters:

- x (real) is the abscissa,
- n (integers) is the degree. Default n = 1

### Function Poly\_Gegenbauer(L, x, [n])

Evaluate the Gegenbauer orthogonal polynomial of 1st and 2nd kind

Parameters:

- x (real) is the abscissa,
- n (integers) is the degree. Default n = 1
- L (real) is the Gegenbauer factor and must be  $L < 1/2$

### Function Poly\_Hermite(x, [n])

Evaluate the Hermite orthogonal polynomial of 1st and 2nd kind

Parameters:

- x (real) is the abscissa,
- n (integers) is the degree. Default n = 1

### Function Poly\_Jacobi(a, b, x, [n])

Evaluate the Jacobi orthogonal polynomial of 1st and 2nd kind

Parameters:

- x (real) is the abscissa,
- n (integers) is the degree. Default n = 1

## Xnumbers Tutorial

a (real) is the power of  $(1-x)$  factor of the weighting function  
b (real) is the power of  $(1+x)$  factor of the weighting function

### Function Poly\_Laguerre(x, [n], [m])

Evaluate the Laguerre orthogonal polynomial of 1st and 2nd kind

Parameters:

x (real) is the abscissa,

n (integers) is the degree. Default n = 1

m (integer) is the number of generalized polynomial. Default m = 0

### Function Poly\_Legendre(x, [n])

Evaluate the Legendre orthogonal polynomial of 1st and 2nd kind

Parameters:

x (real) is the abscissa,

n (integers) is the degree. Default n = 1

## Weight of Orthogonal Polynomials

This set of functions calculate the weight  $c(n)$  for each orthogonal polynomial  $p(x, n)$

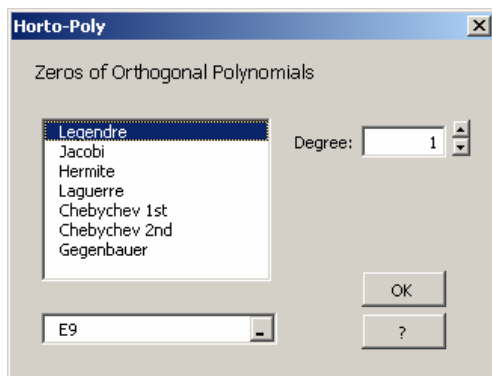
$$c_n \equiv \int_a^b w(x) [p_n(x)]^2 dx$$

Function Poly_Weight_ChebychevT(n)	Chebychev polynomial of the first kind
Function Poly_Weight_ChebychevU(n)	Chebychev polynomial of the second kind
Function Poly_Weight_Gegenbauer(n, l)	Gegenbauer polynomial
Function Poly_Weight_Hermite(n)	Hermite polynomial
Function Poly_Weight_Jacobi(n, a, b)	Jacobi polynomial
Function Poly_Weight_Laguerre(n, m)	Laguerre generalized polynomial
Function Poly_Weight_Legendre(n)	Legendre polynomial

If we divide each orthogonal polynomial family for the relative weight we have an orthonormal polynomial family

## Zeros of Orthogonal Polynomials

This macro finds all roots of the most common orthogonal polynomials  
Its use is very easy. Simply start the **Zero** macro from the menu  
"tools > Ortho-polynomials..."



Choose the family and the degree that you want and fill the optional parameters  
Then press OK

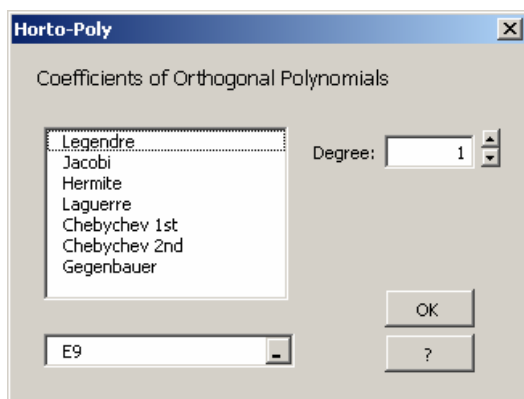
	A	B	C
1	<b>Zeros of Laguerre polynomials</b>		
2	<b>m =</b>	0	
3	<b>Degree =</b>	6	
4	<b>i</b>	<b>root</b>	<b>poly</b>
5	1	15.98287398	2.11164E-13
6	2	9.837467418	1.51048E-14
7	3	5.775143569	2.00361E-15
8	4	2.992736326	3.10805E-17
9	5	1.188932102	-3.25261E-18
10	6	0.222846604	6.95696E-18
11			

This is an example of output for a Laguerre polynomial of 6th degree (  $m = 0$  )

Note: formatting is added for clarity.  
The macro does not format

## Coefficients of Orthogonal Polynomials

This macro calculate the coefficients of the most common orthogonal polynomials  
Its use is very easy. Simply start the **Coeff** macro from the menu "*tools/Ortho-polynomials...*"



Choose the family and the degree that you want and fill the optional parameters.  
Then, press OK

This macro return also the polynomial weight

E	F	G
<b>Coeff. of Laguerre polynomials</b>		
<b>weight =</b>	1	
<b>Degree =</b>	4	
<b>kd =</b>	24	
i	coeff	
0	24	
1	-96	
2	72	
3	-16	
4	1	

This is an example of output for a Laguerre polynomial of 4th degree (m = 0)

The orthopolynomial can be written as

$$L_6(x) = \frac{1}{24}(x^4 - 16x^3 + 72x^2 - 96x + 24)$$

## Complex Arithmetic and Functions

Xnumbers provides a large collection of complex functions

Complex Addition	Complex Hyperbolic Sin
Complex Subtraction	Complex Hyperbolic Cos
Complex Multiplication	Complex Hyperbolic Tan
Complex Division	Complex Inverse Hyperbolic Cos
Polar Conversion	Complex Inverse Hyperbolic Sin
Rectangular Conversion	Complex Inverse Hyperbolic Tan
Complex absolute	Complex digamma
Complex power	Complex Exponential Integral
Complex Root	Complex Error Function
Complex Log	Complex Complem. Error Function
Complex Exp	Complex Gamma Function
Complex inv	Complex Logarith. Gamma Function
Complex negative	Complex Zeta Function
Complex conjugate	Complex Quadratic Equation
Complex Sin	Complex Expression Evaluation
Complex Cos	
Complex Tangent	
Complex Inverse Cos	
Complex Inverse Sin	
Complex Inverse Tan	

### How to insert a complex number

For definition a complex number is an ordered couple of numbers: (a,b)

In Excel a couple of numbers is represented by two vertical or horizontal adjacent cells, so the complex number (a, b) is a range of two cells. The figure below shows both vertical and horizontal representations:

(234 , 105) in range "B7:C7" and in range "B2:B3"  
 (-100 , 23) in the range "E7:F7" and in range "D2:D3"

H7		={=xcplxadd(B7:C7,E7:F7)}									
	A	B	C	D	E	F	G	H	I	J	K
1		A		B		C					
2	re =>	234		-100		134		{=xcplxadd(B2:B3;D2:D3)}			
3	im =>	105	+	23	=	128		{=xcplxadd(B7:C7;E7:F7)}			
4											
5		A		B		C					
6		re	im		re	im		re	im		
7		234	105	+	-100	23	=	134	128		
8											

Most of complex-functions return a complex number, which is an array of two values. For entering complex functions you must select two cells, insert the comple function and give the CTRL+SHIFT+ENTER keys sequence

If you press the ENTER key, the function returns only the real part of the complex number.

### Symbolic rectangular format

Xnumbers support the format "x+jy" only in expression strings passed to the function cplxeval. Except this case, you must always provides a complex number as a couple of real numbers (one or two cells).

The reason for this choice is that the rectangular format is more adapt for symbolic calculation while the array format is more convenient for numerical computation where, often, we have to manage long, decimal numbers

But, of course, you can convert a complex number (a,b) into its symbolic format "a+jb" by the Excel function COMPLEX, as shown in the following example

	A	B	C	D
1	real	imm	symbolic rectangular format	
2	0.523598776	-0.785398163	0.523598775598298-0.785398163397448i	=COMPLEX(A2;B2)
3	12	5	12+5i	=COMPLEX(A4;B4)

XNUMBERS has two sets of complex functions: for standard double precision (prefixed by "cplx") and for multiprecision (prefixed by "xcplx").

### Complex Addition

---

**xcplxadd(a, b, [Digit\_Max])**

**cplxadd(a, b)**

Performs the complex addition:

$$(a_1, a_2) + (b_1, b_2) = (a_1 + a_2, b_1 + b_2)$$

### Complex Subtraction

---

**xcplxsub(a, b, [Digit\_Max])**

**cplxsub(a, b)**

Performs the complex subtraction.

$$(a_1, a_2) - (b_1, b_2) = (a_1 - a_2, b_1 - b_2)$$

### Complex Multiplication

---

**xcplxmult(a, b, [Digit\_Max])**

**cplxmult(a, b)**

Performs the complex multiplication:

$$(a_1, a_2) * (b_1, b_2) = (a_1 b_1 - a_2 b_2, a_1 b_2 + a_2 b_1)$$

## Complex Division

---

**xcplxdiv(a, b, [Digit\_Max])**

**cplxdiv(a, b)**

Performs the complex division

$$\frac{(a_1, a_2)}{(b_1, b_2)} = \left( \frac{a_1 b_1 + a_2 b_2}{b_1^2 + b_2^2}, \frac{a_2 b_1 - a_1 b_2}{b_1^2 + b_2^2} \right)$$

## Polar Conversion

---

**xcplxpolar(z, [angle], [Digit\_Max])**

**cplxpolar(z, [angle])**

Converts a complex number from its rectangular form to the equivalent polar form. The optional parameter *angle* sets the angle unit (RAD, DEG) (default RAD).

$$(x, y) \Rightarrow (\rho, \theta)$$

Where

$$\rho = \sqrt{x^2 + y^2}$$

$$\theta = \operatorname{atan}\left(\frac{y}{x}\right), \quad x > 0$$

$$\theta = \operatorname{sgn}(y) \cdot \frac{\pi}{2}, \quad x = 0$$

$$\theta = \begin{cases} \pi, & y = 0, x < 0 \\ \operatorname{atan}\left(\frac{y}{x}\right) + \operatorname{sgn}(y) \cdot \pi, & y \neq 0, x < 0 \end{cases}$$

x	y	$\rho$	$\theta$ (deg)
1	0	1	0
0.866025	0.5	1	30
0.707107	0.707107	1	45
0.5	0.866025	1	60
0	1	1	90
-0.5	0.866025	1	120
-0.70711	0.707107	1	135
-0.86603	0.5	1	150
-1	0	1	180
-0.86603	-0.5	1	-150
-0.70711	-0.70711	1	-135
-0.5	-0.86603	1	-120
0	-1	1	-90
0.5	-0.86603	1	-60
0.707107	-0.70711	1	-45
0.866025	-0.5	1	-30

## Rectangular Conversion

---

**xcplxrect(z, [angle], [Digit\_Max])**

**cplxrect(z, [angle])**

Converts a complex number from its polar form to the equivalent rectangular form. The optional parameter *angle* sets the angle unit (RAD, DEG) (default RAD).

$$(\rho, \theta) \Rightarrow (x, y)$$

Where

$$\begin{aligned}x &= \rho \cos(\theta) \\ y &= \rho \sin(\theta)\end{aligned}$$

## Complex absolute

---

**xcplxabs(z, [Digit\_Max])**

**cplxabs(z)**

Returns the absolute value of a complex number

$$|z| = \sqrt{z_1^2 + z_2^2}$$

## Complex power

---

**xcplxpow(z, [n], [Digit\_Max])**

**cplxpow(z, [n])**

Returns the  $n^{\text{th}}$  integer power of a complex number  $z^n$  (default  $n = 2$ )

$$z^n = (x + iy)^n = \rho^n \cdot e^{n\theta}$$

Where

$$\rho = \sqrt{x^2 + y^2} \quad , \quad \theta = \text{atan}\left(\frac{y}{x}\right)$$

## Complex Roots

---

**xcplxroot(z, [n], [Digit\_Max])**

**cplxroot(z, [n])**

Returns all the  $n^{\text{th}}$  roots of a complex extended number  $z^{(1/n)}$  (default  $n = 2$ )  
The function returns a matrix of  $(n \times 2)$  values. Remember to press the sequence CTRL+SHIFT+ENTER for insert properly this function.

The root of a complex number is computed by the de Moivre-Laplace formula.

$$\sqrt[n]{z} = \sqrt[n]{x + iy} = \sqrt[n]{\rho} \cdot \left[ \cos\left(\frac{\theta + 2k\pi}{n}\right) + i \cdot \sin\left(\frac{\theta + 2k\pi}{n}\right) \right] \quad , \quad k = 0, 1 \dots n-1$$

where

$$\rho = \sqrt{x^2 + y^2} \quad , \quad \theta = \text{atan}\left(\frac{y}{x}\right)$$



## Xnumbers Tutorial

Note: If you select only one row, the function return only the first complex root (given for  $k = 0$ ).

Example: compute all the 3 complex cubic roots of the number  $z = 8$

	B6					
1						
2	complex number					
3	z =	8	0			
4						
5	roots of complex number					
6	$z^{1/3} =$	2	0			
7		-1	1.732051			
8		-1	-1.732051			
9						

**{=cplxroot(B3:C3;3)}**  
all 3 complex roots

## Complex Log

**xcplxLn(z, [Digit\_Max])**

**cplxLn(z)**

Returns the natural logarithm of a complex number

$$\log(z) = \log(x + iy) = \log(\rho) + \theta$$

Where:

$$\rho = \sqrt{x^2 + y^2}, \quad \theta = \text{atan}\left(\frac{y}{x}\right)$$

## Complex Exp

**xcplxExp(z, [Digit\_Max])**

**cplxExp(z)**

Returns the exponential of a complex number

$$e^z = e^{x+iy} = e^x \cos(y) + ie^x \sin(y)$$

## Complex inverse

**xcplxinv(z, [Digit\_Max])**

**cplxinv(z)**

Returns the inverse of a complex number

$$\frac{1}{z} = \frac{1}{x + iy} = \frac{x}{x^2 + y^2} - i \frac{y}{x^2 + y^2}$$

## Complex negative

---

**xcplxneg(z)**

**cplxneg(z)**

Returns the complex negative

$$-z = -(x + iy) = -x - iy$$

## Complex conjugate

---

**xcplxconj(z)**

**cplxconj(z)**

Returns the conjugate of a complex number

$$\bar{z} = \overline{x + iy} = x - iy$$

## Complex Sin

---

**=cplxsin(z)**

Returns the sine of a complex number

## Complex Cos

---

**cplxcos(z)**

Returns the cosine of a complex number

## Complex Tangent

---

**cplxtan(z)**

Returns the tangent of a complex number

## Complex ArcCos

---

**cplxacos(z)**

Returns the arccosine of a complex number

## Complex ArcSin

---

### **cplxasin(z)**

Returns the arcsine of a complex number

## Complex ArcTan

---

### **cplxatan(z)**

Returns the arctangent of a complex number

## Complex Hyperbolic Sine

---

### **cplxsinh(z)**

Returns the hyperbolic sine of a complex number

Parameter “z” can be a real or complex number (two adjacent cells)

## Complex Hyperbolic Cosine

---

### **cplxcosh(z)**

Returns the hyperbolic cosine of a complex number

Parameter “z” can be a real or complex number (two adjacent cells)

## Complex Hyperbolic Tan

---

### **cplxtnh(z)**

Returns the hyperbolic tangent of a complex number

## Complex Inverse Hyperbolic Cos

---

### **cplxacosh(z)**

Returns the inverse of the hyperbolic cosine of a complex number

## Complex Inverse Hyperbolic Sin

---

### **cplxasinh(z)**

Returns the inverse of the hyperbolic sine of a complex number

## Complex Inverse Hyperbolic Tan

---

### **cplxatanh(z)**

Returns the inverse of the hyperbolic tangent of a complex number

## Complex digamma

---

### **cplxdigamma(z)**

Returns the logarithmic derivative of the gamma function for complex argument.

$$\Psi(x) = \frac{d}{dx} \ln(\Gamma(x)) = \frac{\Gamma'(x)}{\Gamma(x)}$$

## Complex Exponential Integral

---

### **cplxexpint(z)**

Returns the exponential integral of a complex number

$$Ei(x) = - \int_{-x}^{\infty} \frac{e^{-t}}{t} dt$$

## Complex Error Function

---

### **cplxerf(z)**

Returns the "error function" or "Integral of Gauss's function" of a complex number

$$erf(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$$

## Complex Complementary Error Function

---

### **cplxerfc(z)**

Returns the complementary error function for a complex number

$$erfc(z) = 1 - erf(z)$$

## Complex Gamma Function

---

### **cplxgamma(z)**

Returns the gamma function for a complex number

$$\Gamma(x) = \int_0^{\infty} t^{x-1} e^{-t} dt$$

## Complex Logarithm Gamma Function

---

### **cplxgammaLn(z)**

Returns the natural logarithm of the Gamma function for a complex number

## Complex Zeta Function

---

### **cplxzeta(z)**

Returns the Riemann zeta function  $\zeta(s)$  for a complex number. It is an important special function of mathematics and physics which is intimately related with very deep results surrounding the prime number, series, integrals, etc.

Definition: For  $|s| > 1$  the function is defined

$$\zeta(n) = \sum_{k=1}^{\infty} \frac{1}{k^n}.$$

## Complex Quadratic Equation

### cplxEquation2(a, b, c, [DgtMax])

Returns the multiprecision solution of the quadratic equation with complex coefficients

$$a \cdot z^2 + b \cdot z + c = 0$$

where a, b, c are complex

The solutions are found by the resolution formula

$$z = -\frac{b}{2a} \pm \frac{\sqrt{b^2 - 4ac}}{2a}$$

This function returns an (2 x 2) array

The optional parameter DgtMax, from 1 to 200, sets the number of the significant digits. If missing, the computation is in standard double precision.

Example: Find the solution of the following complex equation with 20 digits precision

$$z^2 + (9 - 2i) \cdot z + 4 + i = 0$$

F5		fx {=cplx_equation2(B4:C4;B5:C5;B6:C6;B3)}					
	A	B	C	D	E	F	G
1	$z^2 + (9 - 2i) \cdot z + 4 + i = 0$						
2							
3	Digits	20					
4	a =	1	0			real	imm
5	b =	9	-2		z1 =	-8.5918392090132821071947	2.2219443982515907319261
6	c =	4	1		z2 =	-0.4081607909867178928053	-0.2219443982515907319261
7							

## Number Theory

### Maximum Common Divisor

**xMCD(a1, [a2], [Digit\_Max])**

**MCD(a1, [a2], [a3]...)**

Returns the Maximum Common Divisor (also called Greatest Common Divisor, GCD) of two or more extended numbers

The arguments "a1" and "a2" may be single numbers or arrays (range). At least, two values must be input. If "a1" is a range, "a2" may be omitted

### Minimum Common Multiple

**xMCM(a1, [a2], [Digit\_Max])**

**MCM(a1, [a2], [a3]...)**

Returns the Minimum Common Multiple (also Least Common Multiple, LCM) of two or more extended numbers

The arguments "a1" and "a2" may be single numbers or arrays (range). At least, two values must be input. If "a1" is a range, "a2" may be omitted

Example

	A	B	C	D
1		<b>x</b>		
2		831402		
3		1339481		
4		291720		
5		1650649		
6		255255		
7		1205776	=MCD(B2:B8)	
8		2387242	=MCM(B2:B8)	
9				
10	GCD =	2431	=xprod(B2:	
11	LCM =	9967402918352880		
12	Π =	3.94008780758332018835283346146E+41		

Tip.. The LCM may easily overcome the standard precision limit even if the arguments are all standard precision.

## Rational Fraction approximation

**xfrac(x, [Digit\_Max])**

**fract(x, [ErrMax])**

Returns the rational fractional approximation of a decimal number x, the functions returns a vector of two numbers, numerator N and denominator D :

$$x \approx N / D$$

The optional parameter ErrMax sets the accuracy of the fraction conversion (default=1E-14). The function tries to calculate the fraction with the maximum accuracy possible.

The algorithm employed in this routine uses the continued fraction expansion<sup>9</sup>

$$N_0 = 0, N_1 = 1$$

$$D_0 = 1, D_1 = 0$$

$$N_{i+1} = a_i \cdot N_i + N_{i-1}$$

$$D_{i+1} = a_i \cdot D_i + D_{i-1}$$

Where  $a_i$  are found by the following algorithm:

$$a_{i+1} = \text{int}(x_i / y_i)$$

$$x_{i+1} = y_i$$

$$y_{i+1} = x_i - y_i \cdot a_{i+1}$$

In the example below we want to find the fraction form of decimal number 0.126.

The function returns the solution:

N = 63 , D = 500

	A	B	C
1	Decimal	N	D
2	0.126	63	500
3			
4	{=fract(A2)}		

Often the rational form is not so easy to find, and depends strongly on the precision we want to reach.

See, for example, the fractions that approximate  $\sqrt{2}$  with increasing precision

Digit	N	D	N/D	Error
2	3	2	1.5000000000000000	0.08579
3	7	5	1.4000000000000000	0.01421
4	41	29	1.413793103448280	0.00042
5	99	70	1.414285714285710	7.2E-05
6	239	169	1.414201183431950	1.2E-05
7	1393	985	1.414213197969540	3.6E-07
8	3363	2378	1.414213624894870	6.3E-08
9	8119	5741	1.414213551646050	1.1E-08
10	47321	33461	1.414213562057320	3.2E-10
11	114243	80782	1.414213562427270	5.4E-11
12	275807	195025	1.414213562363800	9.3E-12
13	1607521	1136689	1.414213562372820	2.8E-13
14	3880899	2744210	1.414213562373140	4.2E-14
15	9369319	6625109	1.414213562373090	1.3E-14

You can regulate the desiderate approximation with the parameter ErrMax

<sup>9</sup> form *The art of Computer Programming*, D.E.Knuth, Vol.2, Addison-Wesley, 1969



## Check Prime

---

### Prime(n)

### CheckPrime(n)

These functions<sup>10</sup> state whether a number is prime. They differ only for the values returned

Prime(n) =	"prime" the lowest factor "not found"	if n is prime if n is not prime if the function is not able to check n.
CheckPrime(n) =	TRUE FALSE "?"	if n is prime if n is not prime if the function is not able to check n.

## Next Prime

---

### NextPrime(n)

This function<sup>10</sup> returns the prime number greater than n or "not found"

nextprime(9,343,560,093) = 9,343,560,103

## Modular Power

---

### xPowMod(a, p, m, [digit\_max])

Returns the modular integer power of  $a^p$

That is defined as the remainder of the integer division of  $a^p$  by m

$$r = a^p - m \cdot \left\lfloor \frac{a^p}{m} \right\rfloor$$

Example: compute

$$3^{24} \pmod{9005}$$

$$\text{xPowMod}(3, 24, 9005) = 3306$$

It's easy to prove that

$$3^{24} \pmod{9005} = 282429536481 \pmod{9005} = 3306$$

When the number a or p become larger it is impossible to compute the integer power directly. But the function xPowMod can return the correct result.

---

<sup>10</sup> These functions appears by the courtesy of Richard Huxtable

## Xnumbers Tutorial

Examples: compute

$12^{3939040} \pmod{3001}$

It would be impossible to compute all the digits of this power. Using multiprecision we have

```
xPow(12, 3939040) = 1.24575154970238125896669174496E+4250938
```

This result shows that  $12^{3939040}$  has more than 4 million of digits!  
Nevertheless the remainder of this impossible division is

```
xPowMod(12, 3939040, 3001) = 947
```

## Perfect Square

---

### xisSquare(n)

Checks if a number n is a perfect square

```
xisSquare(1000018092081830116) = TRUE
```

Because:  $1000018092081830116 = 1000009046^2$

```
xisSquare(2000018092081830116) = FALSE
```

## Check odd/even

---

### xisOdd (n)

Checks if a number n is odd (TRUE) or even (FALSE)

## Factorize

---

### Factorize()

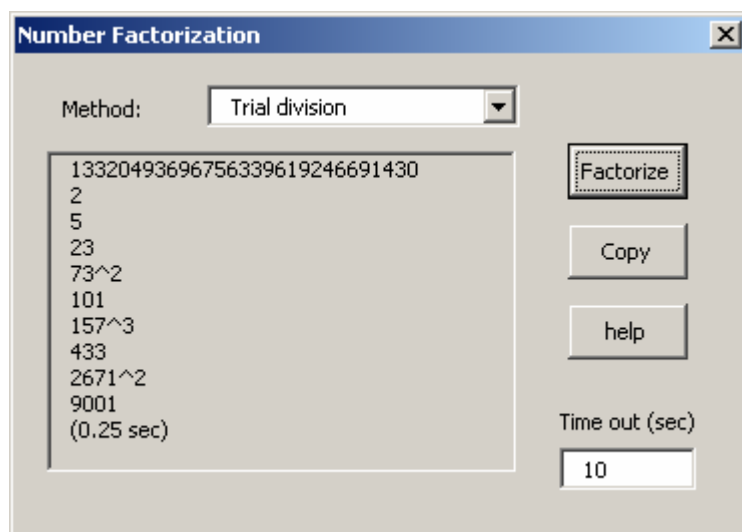
This macro factorizes an integer number returning the list of its prime factors with their exponents

$$n = p_1^{e1} \cdot p_2^{e2} \cdot p_3^{e3} \cdot \dots p_k^{ek} \quad \text{where } p_i \in \{\text{prime}\}$$

Example. Assume to have in the cell A2 the following extended number

13320493696756339619246691430

Select the cell contains the number you want to factorize and the run the macro Factorize (from the Xnumbers menu *Macros > Numbers* or from the Handbook). Choose a factorization method, for example the "Trial Division" and click "Factorize"



Click "copy" if you want to copy the list in the worksheet, starting from the cell just below the number cell A2.

The macro stops itself after the time out is reached, and prompts if you want to continue or interrupt the factorization task

This macro uses the trial division method with the prime table generated by the *Eratostene's sieve* algorithm. This method is adapt for numbers having factors no more that 7 digits max. For higher factor the elaboration time becomes extremely long. In this situation we can choose a second factorization method, the so called *Pollard rho* algorithm, for craking a number into two lower factors (not necessary prime). Each factors, if not prime, can be factorized separately with the trial division method.

Example. The number

$$18446744073709551617 = 274177 * 67280421310721$$

can be factorized with both methods: it requires about 33 sec with trial division, but less then 3 sec with Pollard method

The following number instead can be factorize only with Pollard method (about 40 sec).

$$10023859281455311421 = 7660450463 * 1308520867$$

Note that in this case both factors have 10 digits. The factors are prime so the factorization stops.

For prime testing see the probabilistic *Fermat's Prime Test*

## Factorize function

### Factor(n)

This function performs the decomposition in prime factor of a given number  
Returns an array of two columns: the first column contains the prime factors and the second column contains the exponents

Note. This function is adapt for low-moderate numbers.

	A	B	C
1			
2	2277785128000	fact	exp
3		2	6
4		5	3
5	{=Factor(A2)}	23	2
6		73	2
7		101	1
8		#N/D	#N/D
9			

In this example, the given number is decomposed in 5 factors

$$2277785128000 = 2^6 5^3 23^2 73^2 101$$

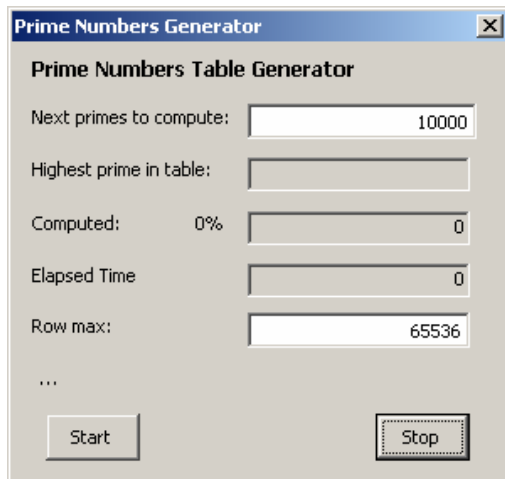
The #ND symbol indicates the end of factors list. To make sure to get all factors you have to extend the selection until you see this symbol

## Prime Numbers Generator

---

### PrimeGenerator()

This macro is useful to generate your own table of prime number. The table begins from the cell A1 of the active worksheet.



The macro computes 10.000 (default) prime numbers for each time.

The macro can be stop and restart as you like

It always restarts from the last prime number saved.

## Fermat's Prime Test

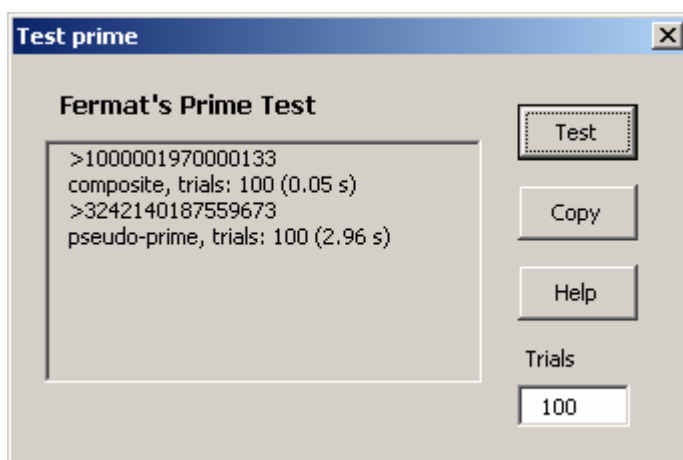
---

### Prime\_Test\_Fermat()

This macro perform the probabilistic prime test with the Fermat's method. This is adapt for long number. Using it is very simple.

Start the macro from the menu Macros > Numbers > Prime test

Select the number that you want to test and press "Test". After few seconds you get the results.



Note that this test is exact for detecting composite numbers, but it can detect a prime number with a finite probability (usually very high).

Numbers satisfying the Fermat' test are called **"pseudo-prime"**

The probability is correlated to the number of trials "T" with the following approximate formula

## Xnumbers Tutorial

$$p = 1 - 2^{-T} = 1 - 2^{-100}$$

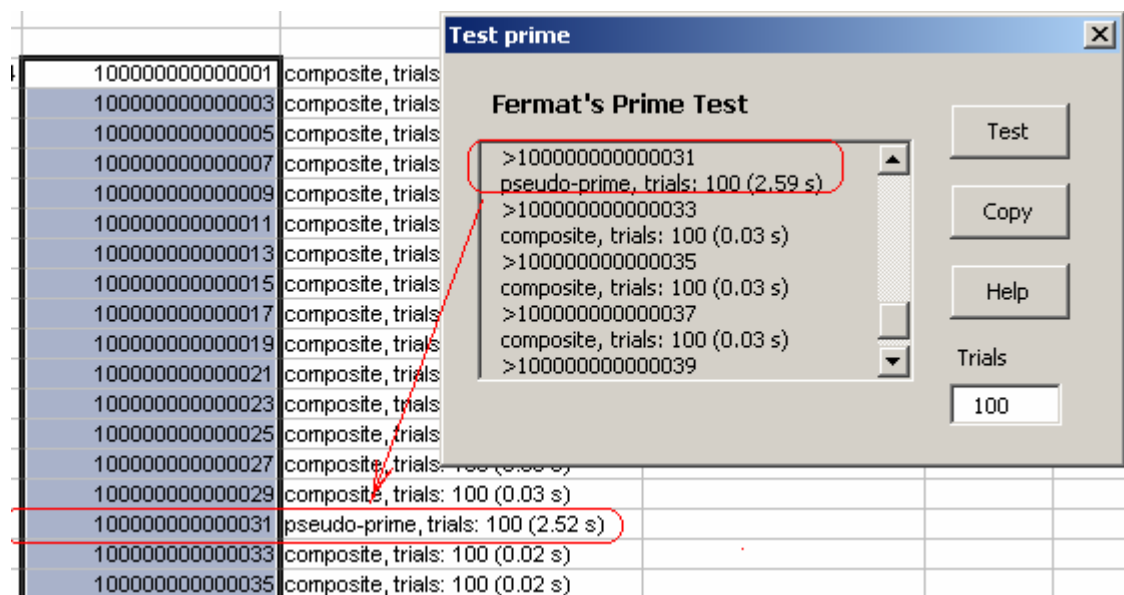
For  $T = 100$  the probability is about  $1 - 7.8 \cdot 10^{-31}$

You can also select a list of cells containing several numbers to test. This is useful for find new long prime numbers

Example: find the next prime number after 100000000000000 (1E14)

A prime number must be odd, so let's begin to prepare a sequence of 20 or more odd numbers starting from 100000000000001

The frequency of prime numbers, in this range, is about 5%, so we hope to find a prime number in our list. If this does not happen we try with a successive set of numbers, and so on, until a prime comes out.



In this case we have found a probable prime 1000000000000031  
We can prove that it is a true prime

## Diophantine Equation

### DiophEqu(a, b, c)

This function solves the Diophantine linear equation

$$a x + b y = c \quad x, y \in \mathbb{Z}$$

where a, b, c, x, y are all integer numbers

The integer solutions  
can be expressed as

$$\begin{cases} x_k = x_0 + k \cdot D_x \\ y_k = y_0 + k \cdot D_y \end{cases} \quad \text{for } k = 0, \pm 1, \pm 2 \dots$$

This function return an array (2, 2) of four integer values.

The first row contains a particular solution, while the second row contains the integer increments for generating all the solutions.

If you want only a particular solution [x0, y0] simply select an array of 2 adjacent cells. If the equation has no solution the function return "?"

$$\begin{bmatrix} x_0 & y_0 \\ D_x & D_y \end{bmatrix}$$

Example. Find all the integer solutions of the equation  $2x+3y = 6$

	A	B	C	D
1				
2		<b>a</b>	<b>b</b>	<b>c</b>
3		2	3	6
4		{=DiophEqu(B4;C4;D4)}		
5				
6		<b>x</b>	<b>y</b>	
7		-6	6	
8		3	-2	
9				

As we can see, the function returns one solution (-6, 6) and the increments (3, -2). So all the integer solutions of the above equation can be obtained from the following formulas for any integer value of k

$$\begin{cases} x_k = -6 + 3k \\ y_k = 6 - 2k \end{cases}$$

Often is not so easy to find the solution of a diophantine equation. Let's see

**Long numbers.** This function works also with extended numbers.

Example. Find a solution of the equation  $ax+by = c$  having the following coefficients

<b>a</b>	<b>b</b>	<b>c</b>
18760000596690052	13650847757772	64

Note that the first coefficients has 17 digits and the second one has 14 digits. Without multiprecision it would be difficult to solve this problem. But fortunately the function return the following results

<b>x</b>	<b>y</b>
-6662999124128	9156784235119760

You can enjoy yourself to prove that this result is correct

## Linear Algebra Functions

### Matrix Addition

---

#### **xMatAdd(mat1, mat2, [DgtMax])**

Performs the addition of two matrices in multiprecision  
mat1 and mat2 are (n x m) arrays

$$\begin{bmatrix} c_{11} & \dots c_{1n} \\ c_{n1} & \dots c_{nm} \end{bmatrix} = \begin{bmatrix} a_{11} & \dots a_{1n} \\ a_{n1} & \dots a_{nm} \end{bmatrix} + \begin{bmatrix} b_{11} & \dots b_{1n} \\ b_{n1} & \dots b_{nm} \end{bmatrix}$$

### Matrix Subtraction

---

#### **xMatSub(mat1, mat2, [DgtMax])**

Performs the subtraction of two matrices in multiprecision  
mat1 and mat2 are (n x m) arrays

$$\begin{bmatrix} c_{11} & \dots c_{1n} \\ c_{n1} & \dots c_{nm} \end{bmatrix} = \begin{bmatrix} a_{11} & \dots a_{1n} \\ a_{n1} & \dots a_{nm} \end{bmatrix} - \begin{bmatrix} b_{11} & \dots b_{1n} \\ b_{n1} & \dots b_{nm} \end{bmatrix}$$

### Matrix Multiplication

---

#### **xMatMult(mat1, mat2, [DgtMax])**

Performs the multiplication of two matrices in multiprecision  
mat1 (n x p) and mat2 (p x m) are arrays

$$\begin{bmatrix} c_{11} & \dots c_{1n} \\ c_{n1} & \dots c_{nm} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots a_{1p} \\ a_{n1} & a_{n2} & \dots a_{np} \end{bmatrix} \cdot \begin{bmatrix} a_{11} & \dots a_{1m} \\ a_{21} & \dots a_{2m} \\ a_{p1} & \dots a_{pm} \end{bmatrix}$$

### Matrix Inverse

---

#### **xMatInv(A, [DgtMax])**

Returns the inverse of square matrix (n x n) in multiprecision  
It returns "?" for singular matrix.

This function uses the Gauss-Jordan diagonalization algorithm with partial pivoting method.

## Matrix Determinant

---

### **xMatDet(A, [DgtMax])**

Returns the determinant of a square matrix in multiprecision  
It returns "?" for singular matrix.

## Matrix Modulus

---

### **xMatAbs(A, [DgtMax])**

Returns the absolute value of a matrix or vector in multiprecision.  
It is also known as "modulus" or "norm"  
Parameters A may be an (n x m) array or a vector

$$\|A\| = \sqrt{\sum_{i=1}^n \sum_{j=1}^m (a_{i,j})^2}$$

## Scalar Product

---

### **xProdScal( v1, v2, [DgtMax])**

Returns the scalar product of two vectors in multiprecision

$$c = V_1 \bullet V_2 = \sum_{i=1}^n V_{1i} \cdot V_{2i}$$

Note: The scalar product is zero if, and only if, the vectors are perpendicular

$$V_1 \bullet V_2 = 0 \quad \Leftrightarrow \quad V_1 \perp V_2$$

## Similarity Transformation

---

### **= xMat\_BAB(A, B, [DgtMax])**

Returns the matrix product:

$$C = B^{-1} A B$$

This operation is also called the "*similarity transformation*" of the matrix A by the matrix B. This operation plays a crucial role in the computation of eigenvalues, because it leaves the eigenvalues of the matrix A unchanged. For real, symmetrical matrices, B is orthogonal. The similarity transformation is also called the "*orthogonal transformation*". A and B must be square matrices.



## Matrix Power

```
= xMatPow(A, n, [DgtMax])
```

Returns the integer power of a square matrix.

$$B = A^n = \overbrace{A \cdot A \cdot A \dots A}^{n \text{ time}}$$

## Matrix LU decomposition

```
= xMat_LU(A, [Pivot], [DgtMax])
```

Returns the LU decomposition of a square matrix A  
It uses Crout's algorithm

$$A = L \cdot U = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

Where  $L$  is a lower triangular matrix, and  $U$  is an upper triangular matrix

The parameter Pivot (default=TRUE) activates the partial pivoting.

Note: if partial pivot is activated, the LU decomposition can refer to a permutation of A

If the square matrix has dimensions  $(n \times n)$ , this function returns an  $(n \times 3n)$  array where the first  $n$  columns are the matrix  $L$ , the next  $n$  columns are the matrix  $U$ , and the last  $n$  columns are the matrix  $P$ .

Globally, the output of the `Mat_LU` function will be:

- Columns (1, n) = Matrix **L**
- Columns (n+1, 2n) = Matrix **U**
- Columns (2n+1, 3n) = Matrix **P**

When pivoting is activated the right decomposition formula is  $\mathbf{A} = \mathbf{P} \mathbf{L} \mathbf{U}$  , where  $\mathbf{P}$  is a permutation matrix

Note: LU decomposition does not work if the first element of the diagonal of A is zero

Example: find the factorization of the following 3x3 matrix **A**

[illegible]

Note: if you want to get only the L and U matrices select a range (3 x 6) before entering this function

## Matrix $LL^T$ decomposition

**= xMat\_LL(A, [DgtMax])**

This function returns the  $LL^T$  decomposition of a square matrix A  
It uses Cholesky's algorithm

$$A = L \cdot L^T = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \cdot \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix}^T$$

Where L is a lower triangular matrix

The function returns an (n x n) array

Note: Cholesky decomposition works only for positive definite matrices

Example.

	A	B	C	D	E	F	G	H	I	J	K
1											
2		1	0	2	1			matrix L			
3		0	1	-1	-2			1	0	0	0
4		2	-1	21	8			0	1	0	0
5		1	-2	8	7			2	-1	4	0
6								1	-2	1	1
7											

`{=xMat_LL(B2:E5)}`

The diagonal elements of the L matrix are all positive. So the matrix A is definite positive and the decomposition is correct. This function simply stops when detects a negative diagonal element, returning the incomplete decomposition.

See this example

	A	B	C	D	E	F	G	H	I
1			A				L		
2		4	8	4		2	0	0	
3		8	4	3		4	-12	0	
4		4	3	4		2	0	0	
5									

A diagonal element of the L matrix is negative. So the matrix is not positive definite and the decomposition cannot be completed

## Vector Product

**= xProdVect(v1, v2, [DgtMax])**

Returns the vector product of two vectors

$$V_1 \times V_2 = \begin{bmatrix} v_{11} \\ v_{21} \\ v_{31} \end{bmatrix} \times \begin{bmatrix} v_{12} \\ v_{22} \\ v_{32} \end{bmatrix} = \begin{bmatrix} v_{21}v_{32} - v_{22}v_{31} \\ v_{12}v_{31} - v_{11}v_{32} \\ v_{11}v_{22} - v_{21}v_{12} \end{bmatrix}$$

Note that if V1 and V2 are parallels, the vector product is the null vector.

## Solve Linear Equation System

---

### xSYSLIN( A, B, [DgtMax])

Solves a system of linear algebraic equations in multiprecision.

The input parameter A is an (n x n) array, B may be a vector (n x 1) or an (n x m) array

Returns a vector (n x 1) or an (n x m) array depending by the argument B

A set of m linear systems in n unknowns looks like this:

$$[A] \cdot x_1 = b_1, [A] \cdot x_2 = b_2, \dots, [A] \cdot x_m = b_m$$

It can be rewritten as:

$$[A] \cdot [x] = [B] \Rightarrow \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{n1} & \dots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} x_{11} & \dots & x_{1m} \\ \dots & \dots & \dots \\ x_{n1} & \dots & x_{nm} \end{bmatrix} = \begin{bmatrix} b_{11} & \dots & b_{1m} \\ \dots & \dots & \dots \\ b_{n1} & \dots & b_{nm} \end{bmatrix}$$

This function uses the Gauss-Jordan diagonalization algorithm with partial pivoting method.

Example. Find the solution of the following 7x7 linear system

A							b
462	792	1287	2002	3003	4368	12376	24290
924	1716	3003	5005	8008	12376	31824	62856
1716	3432	6435	11440	19448	31824	75582	149877
3003	6435	12870	24310	43758	75582	167960	333918
5005	11440	24310	48620	92378	167960	352716	702429
8008	19448	43758	92378	184756	352716	705432	1406496
12376	31824	75582	167960	352716	705432	1352078	2697968

The solution is the vector [1, 1, 1, 1, 1, 1, 1]. Solving with standard arithmetic, we get an average accuracy of about 1E-8, while in multiprecision we have an accuracy better than 1E-28

[illegible]

## Solve Linear Equation System with Iterative method

**SYSLIN\_ITER\_G(A, b, x0, [Nmax])**

This function find the solution of a linear system by the iterative Gauss-Seidel algorithm.

$$[\mathbf{A}] \cdot \mathbf{x} = \mathbf{b}$$

The parameter  $A$  is the system matrix ( $n \times n$ )

The parameter  $b$  is the system vector  $n \times 1$ )

The parameter `x0` is the starting approximate solution vector ( $n \times 1$ )

The parameter `Nmax` is the maximum steps performed (default = 1)

The function returns the vector at Nmax step, if the matrix is convergent, this vector is closer to the exact solution.

In the example below it is returned the 20<sup>th</sup> GS iteration step.

As we can see, the values approximate the exact solution  $[4, -3, 5]$ . Precision increase with steps (of course, for convergent matrices)

	A	B	C	D	E	F	G	H
1	<b>Linear system resolution with iterative methods</b>							
2						<b>Step =&gt;</b>	0	20
3	6	-1	2	37		<b>X1 =&gt;</b>	0	3,999984082
4	2	-7	6	59		<b>X2 =&gt;</b>	0	-2,999979881
5	-1	3	5	12		<b>X3 =&gt;</b>	0	4,999984745
6								
7	{=SYSLIN_ITER_G(A3:C5;D3:D5;G3:G5;H2)}							
8								

For  $N_{\max}=1$ , we can study the iterative method step by step

Usually, the convergence speed is quite low, but it can be greatly accelerated by the Aitken's extrapolation formula, also called as "square delta extrapolation"

## Square Delta Extrapolation

### ExtDelta2(x)

### xExtDelta2(x, [DgtMax])

This function returns the Aitken's extrapolation, also known as "Square Delta Extrapolation". The parameter x is a vector of n value ( $n > 2$ ), in vertical consecutive cells. ( $n = 2$  for the multi-precision function xExtDelta2).

This formula can be applied to any generic sequence of values (vector with  $n > 2$ ) for accelerating the convergence.

$$(x_1, x_2, x_3, \dots, x_n) \xrightarrow{\Delta^2} (v_1, v_2, v_3, \dots, v_{n-2})$$

Note that this algorithm produces a vector with  $n-2$  values. If  $n = 3$ , the result is a single value.

Taking the difference:

$$\Delta_i = x_{i+1} - x_i$$

The Aitken's extrapolation formula is:

$$v_i = x_i - \frac{\Delta_{i-1}^2}{\Delta_{i-1} - \Delta_{i-2}} = x_i - \frac{(x_i - x_{i-1})^2}{(x_i - 2x_{i-1} + x_{i-2})}$$

This formula can be applied to the second sequence to obtain a new sequence with  $n-4$  values, and so on. The process stops when the last sequence has less than 3 values.

Example. we want to find the numeric solution of the equation  $x = \cos(x)$

We choose the central point method. Starting from  $x_0 = 0$  we build the iterations

$$x_{n+1} = \cos(x_n)$$

As we can see in the following table, the convergence is evident but very slow (after 12 iterations the precision is about  $3E-5$ ).

	A	B	C	D
1	n	x	cos(x)	x <sub>n</sub> -x <sub>n-1</sub>
2	0	0	1	1
3	1	1	0.540302306	0.4596977
4	2	0.540302306	0.857553216	0.3172509
5	3	0.857553216	0.65428979	0.2032634
6	4	0.65428979	0.793480359	0.1391906
7	5	0.793480359	0.701368774	0.0921116
8	6	0.701368774	0.763959683	0.0625909
9	7	0.763959683	0.722102425	0.0418573
10	8	0.722102425	0.750417762	0.0283153
11	9	0.750417762	0.731404042	0.0190137
12	10	0.731404042	0.744237355	0.0128333
13	11	0.744237355	0.73560474	0.0086326
14	12	0.73560474	0.741425087	0.0058203

The functions of this worksheet are:

The cell B2 contains the starting value  $x_0$

The cell B3 contains the formula  
the formula

=C2

The cell C2 contains the formula

=COS(B2)

The cell D3 contains the formula

=ABS(B2-C2)

	A	B	C	D	E
12	10	0.731404042	0.744237355	0.012833	
13	11	0.744237355	0.73560474	0.008633	
14	12	0.73560474	0.741425087	0.00582	
15					
16		0.739076383	=ExtDelta2(B12:B14)		
17					

As we can see the last 12<sup>th</sup> value has an error of about  $3.5E-3$ . Taking the delta extrapolation of the three last values we get a new value having an accuracy better than  $1E-5$ .

Now let's repeat the iterative process using systematically the square delta extrapolation

	A	B	C	D
1	n	x	cos(x)	$ x_n - x_{n-1} $
2	0	0	1	1
3	1	1	0.540302306	0.4596977
4	2	0.540302306	0.857553216	0.3172509
5	3	0.685073357	0.774372634	0.0892993
6	4	0.774372634	0.714859872	0.0595128
7	5	0.714859872	0.755185104	0.0403252
8	6	0.738660156	0.739371336	0.0007112
9	7	0.739371336	0.738892313	0.000479
10	8	0.738892313	0.739215005	0.0003227
11	9	0.739085106	0.739085151	4.495E-08
12	10	0.739085151	0.739085121	3.028E-08
13	11	0.739085121	0.739085141	2.04E-08
14	12	0.739085133	0.739085133	1.11E-16
15				
16				
17				

In this process, we have systematically repeated the  $\Delta^2$  extrapolation every 3 iterations

We have insertet in the cell B5

=ExtDelta2(B2:B4)

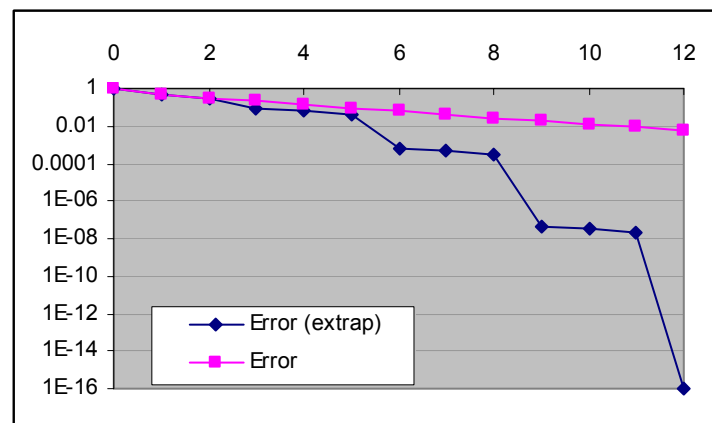
In the cell B8

=ExtDelta2(B3:B7)

In the cell B14

=ExtDelta2(B9:B11)

The acceleration is superb!. After only 12 steps, the precision is better than 1E-15. The graph below shows better than many words this acceleration effect



The Aitken's extrapolation formula work very well with the Gauss-Seidel iterative method, and for accelerating the convergence of many series.

## Multiprecision Matrix operations (macro)

This application collects a set of useful macros performing multiprecision matrix operations

<b>Determinant</b>	$\det(A)$	Gauss-Jordan algorithm
<b>Addition</b>	$A + B$	
<b>Subtraction</b>	$A - B$	
<b>Multiplication</b>	$A \cdot B$	
<b>Scalar multiplication</b>	$k \cdot A$	
<b>Inverse</b>	$A^{-1}$	Gauss-Jordan algorithm
<b>Similarity transform</b>	$B^{-1} A B$	
<b>Linear System</b>	$AX = B$	Gauss-Jordan algorithm
<b>Linear System overdetermined.</b>	$Ax = b$	rows > columns
<b>LU decomposition</b>	$A = LU$	Crout's algorithm
<b>Cholesky decomposition</b>	$A = LL^T$	Cholesky's algorithm
<b>Norm</b>	$\ A\ $	
<b>Scalar product</b>	$A^T \cdot B$	
<b>SVD</b>	$U \cdot \Sigma \cdot V^T$	Golub-Reinsch algorithm

The use of this macro is quite simple. If the operation requires only one matrix (determinant, inversion, etc.) select the matrix, start the macro and choose the appropriate operation

Other operations require two matrices (addition, multiplication, etc.). In that case you have also to select the second matrix.

The internal calculus is performed in multiprecision. The result is converted in standard precision (15 significant digits max) for more readability, but you may also leave it in full multiprecision format.

Example: If you want to solve the following linear system  $Ax = b$ .

	A	B	C	D	E	F	G	H	I	J
1		<b>A</b>					<b>b</b>		<b>x</b>	
2		4	5	3	3		250			
3		3	4	0	2		140			
4		9	4	6	1		295			
5		-1	2	-4	0		-60			
6										

Select the matrix A and start the macro

## Xnumbers Tutorial

Choose the operation “Linear System” and then move in the right field to select the vector b.

Matrix / vector A (4 x 4)  
\$B\$2:\$E\$5

Matrix / Vector B (4 x 1)  
\$G\$2:\$G\$5

Output  
starting from cell: Foglio4!\$I\$2  
convert into double ☒

Indicate, if necessary, the upper-left cell of the range where you want to write the result.

Then, press OK. The result will be filled starting from the output cell I2.

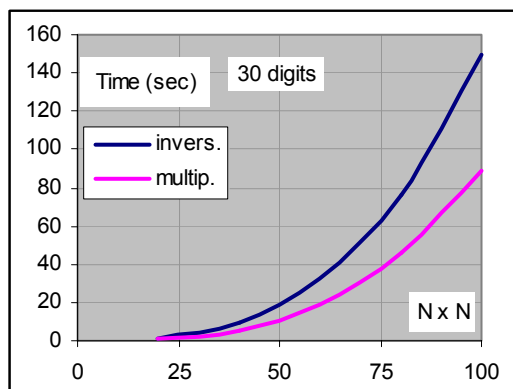
### Smart Selector



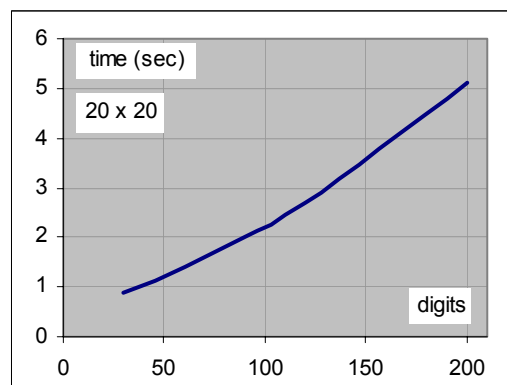
The special button near the input field is useful for selecting large matrices. Select the first cell, or an internal cell of the matrix and then press this button. The entire matrix will be selected

## Elaboration time

Multiprecision computation slows down the computation considerably. It takes much more time than the standard double precision. The time depends on the matrix dimension and on the precision digits. The following graphs show the average time for the inversion and the multiplication of dense matrices.



Multiprecision



standard precision

As we can see, the inversion of a (100 x 100) matrix, with 30 precision digits, takes about 150 seconds. Clearly, for this kind of tasks, macros are more suitable than functions.



## Integrals & Series

### Discrete Fourier Transform

---

**=DFT(samples)**

**=FFT(samples)**

Returns the complex matrix of the DFT transformation of N samples.

This function returns an (N x 2) array. The first column contains the real part; the second column the complex part

If N is an integer power of 2, thus  $N=2^p$ , use the fastest FFT

FFT uses the Cooley and Tukey decimation-in-time algorithm.

#### Formulas

Given N samples (  $f(0), f(1), f(2), \dots, f(N-1)$  ) of a periodic function  $f(t)$  with a normalized sampling rate ( $T=1$ ), the DFT is defined by:

$$F(k) = F_r(k) + i \cdot F_i(k) = \sum_{n=0}^{N-1} f(n) \cdot [\cos(2\pi nk / N) - i \cdot \sin(2\pi nk / N)]$$

The components ( $F_r, F_i$ ) are called the harmonic spectrum of  $f(t)$

From the Fourier series, we can approximate a periodic function  $f(t)$  by:

$$f(t) \cong a_0 + \sum_{k=1}^{K-1} a_k \cos(k\omega \cdot t) + b_k \sin(k\omega \cdot t)$$

where the coefficients ( $a_k, b_k$ ) are the components  $2F_r$  and  $2F_i$  of DFT

Example: Find the 16-FFT of the following periodic function ( $T = 1$  sec)

$$f(t) = 3 + \cos(\omega t) + 0.5 \cos(3\omega t) \quad \text{where} \quad \omega = \frac{2\pi}{T}$$

First of all we have to sample the given function. Setting  $N = 16$ , we have a sampling period of

$$\Delta t = \frac{T}{N} = \frac{1}{16} \Rightarrow t_i = i \cdot \Delta t, \quad i = 0, 1, \dots, N-1$$

$$f_i = 3 + \cos(\omega t_i) + 0.5 \cos(3\omega t_i)$$

Applying the FFT function at the samples set (  $f_0, f_1, f_2, \dots, f_{15}$  ), we get the complex discrete Fourier's transform

	A	B	C	D	E
1	T (sec)	n	$\Delta T$	$\omega$	
2	1	16	0.0625	6.283185	
3					
4	n°	t	f(t)	FFT re	FFT im
5	1	0	4.5	3	0
6	2	0.0625	4.115221	0.5	4.58E-16
7	3	0.125	3.353553	-8.97E-32	5.55E-17
8	4	0.1875	2.920744	0.25	3.12E-17
9	5	0.25	3	1.79E-31	-1.11E-16
10	6	0.3125	3.079256	-2.5E-16	-5.2E-16
11	7	0.375	2.646447	8.97E-32	-5.55E-17
12	8	0.4375	1.884779	-1.11E-16	-9.47E-16
13	9	0.5	1.5	0	0
14	10	0.5625	1.884779	5.55E-17	-3.19E-16
15	11	0.625	2.646447	-8.97E-32	5.55E-17
16	12	0.6875	3.079256	4.58E-16	-5.79E-16
17	13	0.75	3	-1.79E-31	1.11E-16
18	14	0.8125	2.920744	0.25	4.93E-16
19	15	0.875	3.353553	8.97E-32	-5.55E-17
20	16	0.9375	4.115221	0.5	1.38E-15
21					
22					
23					

Note that the FFT returns a (16 x 2) matrix. The first column contains the real part of FFT while the second column the imaginary one.

The magnitude and phase can be easily obtained with the following formulas

$$A_i = \sqrt{(FFT_{re})^2 + (FFT_{im})^2}$$

$$\theta_i = \arctan\left(\frac{FFT_{im}}{FFT_{re}}\right)$$

Note that the first row of the FFT contains the average of f(t).  
Note also that the rows from 10 to 16 are the mirror copy of the previous rows.

## Discrete Fourier Inverse Transform

**=DFT\_INV(samples)**

**=FFT\_INV(samples)**

Returns the inverse of the DFT transform of N complex samples.

This function returns an (N x 2) array containing the samples of the function f(t)

If N is an integer power of 2, thus  $N=2^p$ , use the fastest FFT\_INV function

FFT\_INV uses the Cooley and Tukey decimation-in-time algorithm.

### Formulas

$$f(n) = \sum_{k=0}^{N-1} [F_r(k) + i \cdot F_i(k)] \cdot [\cos(2\pi nk / N) + i \cdot \sin(2\pi nk / N)]$$

Where the components (Fr , Fi ) are the harmonic spectrum of f(t)

Example: Find the inverse transform of the FFT computed in the previous example

	A	B	C	D	E	F	G
1	T (sec)	n	$\Delta T$	$\omega$			
2	1	16	0.0625	6.283185	{=FFT_INV(D5:E20)}		
3							
4	n°	t	f(t)	FFT re	FFT im	IFFT re	IFFT im
5	1	0	4.5	3	0	4.5	0
6	2	0.0625	4.115221	0.5	4.58E-16	4.11522125	-1.1102E-16
7	3	0.125	3.353553	-8.97E-32	5.55E-17	3.35355339	-1.1102E-16
8	4	0.1875	2.920744	0.25	3.12E-17	2.92074367	3.4694E-18
9	5	0.25	3	1.79E-31	-1.11E-16	3	3.5872E-31
10	6	0.3125	3.079256	-2.5E-16	-5.2E-16	3.07925633	3.1225E-17
11	7	0.375	2.646447	8.97E-32	-5.55E-17	2.64644661	-5.5511E-17
12	8	0.4375	1.884779	-1.11E-16	-9.47E-16	1.88477875	5.5511E-17
13	9	0.5	1.5	0	0	1.5	0
14	10	0.5625	1.884779	5.55E-17	-3.19E-16	1.88477875	1.1102E-16
15	11	0.625	2.646447	-8.97E-32	5.55E-17	2.64644661	1.1102E-16
16	12	0.6875	3.079256	4.58E-16	-5.79E-16	3.07925633	-3.4694E-18
17	13	0.75	3	-1.79E-31	1.11E-16	3	-3.5872E-31
18	14	0.8125	2.920744	0.25	4.93E-16	2.92074367	-3.1225E-17
19	15	0.875	3.353553	8.97E-32	-5.55E-17	3.35355339	5.5511E-17
20	16	0.9375	4.115221	0.5	1.38E-15	4.11522125	-5.5511E-17

As we can see, the first column of FFT\_INV returns the samples of f(t) that have originated the FFT

## Discrete Fourier Spectrum

**=DFSP(samples, [dB], [Angle])**

This function returns the harmonic spectrum of a samples set

The parameter samples are a vector of N equidistance samples

The optional parameter dB (default FALSE) sets the output in decibel

The optional parameter Angle (default "RAD") sets the angle unit (RAD, GRAD, DEG)

The function returns an (N x 2) array, containing the amplitude and phase.

The spectrum is computed for real positive frequencies.

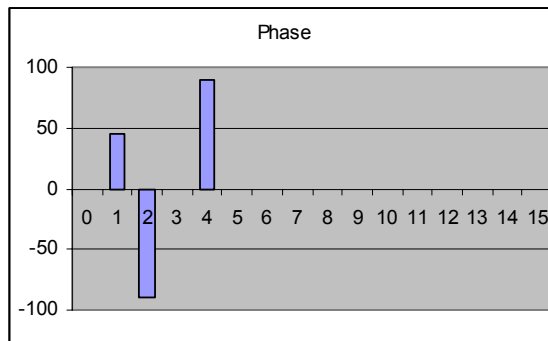
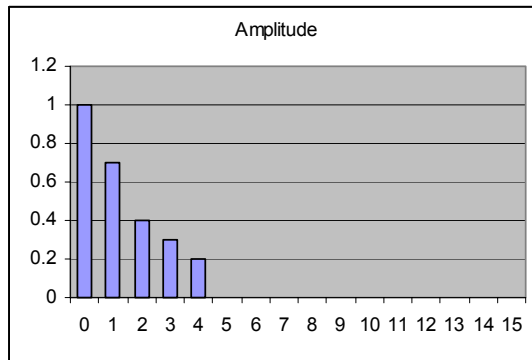
$$(A_n, \theta_n)$$

Where

$$f(t) \cong f(0) + \sum A_n \sin(n\omega t + \theta_n)$$

Example: Find the harmonic spectrum of the following 32 samples

	A	B	C	D	E
7	Sample	Time	f(t)	Amp	Phase
8	0	0.0000	1.29497	1	0
9	1	0.0313	1.52057	0.7	45
10	2	0.0625	1.64104	0.4	-90
11	3	0.0938	1.68629	0.3	0
12	4	0.1250	1.71213	0.2	90
13	5	0.1563	1.75673	0	0
14	6	0.1875	1.81475	0	0
15	7	0.2188	1.84356	0	0
16	8	0.2500	1.79497	0	0
17	9	0.2813	1.65043	0	0
18	10	0.3125	1.43592	0	0
19	11	0.3438	1.20674	0	0
20	12	0.3750	1.01213	0	0
21	13	0.4063	0.86318	0	0
22	14	0.4375	0.72644	0	0
23	15	0.4688	0.54964	0	0
24	16	0.5000	0.30503		
25	17	0.5313	0.02317		
26	18	0.5625	(=DFSP(C8:C39;,"DEG"))		
27	19	0.5938			
28	20	0.6250	-0.11213		
29	21	0.6563	0.26658		
30	22	0.6875	0.75093		
31	23	0.7188	1.17839		
32	24	0.7500	1.40503		
33	25	0.7813	1.37151		
34	26	0.8125	1.12977		
35	27	0.8438	0.81656		
36	28	0.8750	0.58787		
37	29	0.9063	0.54783		
38	30	0.9375	0.70787		
39	31	0.9688	0.9941		



## Inverse Discrete Fourier Spectrum

**=DFSP\_INV(spectrum, [dB], [Angle])**

This function rebuilds the temporal sequence from its real spectrum (amplitude, phase)

$$(A_n, \theta_n) \Rightarrow f(t_i)$$

The parameter spectrum is an (M x 2) array. Each row contains a harmonic. The first column contains the amplitude and the second column the phase

The optional parameter dB (default FALSE) sets the output in decibel

The optional parameter Angle (default "RAD") sets the angle unit (RAD, GRAD, DEG)

The function returns the vector (N x 1) where N = 2M

## 2D Discrete Fourier Transform

**=FFT2D (samples)**

This function performs the 2D-FFT of a bidimensional data samples (x, y).

The parameter Samples is an (N x M) array where N and M are integer powers of 2 (4, 8, 16, 32, 64...)

The function returns an (2N x M) array. The first N rows contain the real part, the last N rows contain the imaginary part.

Note: This function requires a large amount of space and effort. Usually it can works with matrices up to (64 x 64).

Example: Analyze the harmonic component of the following 8x8 data matrix

	A	B	C	D	E	F	G	H	I
9		0	0.125	0.25	0.375	0.5	0.625	0.75	0.875
10	0	2.007	0.741	0.393	0.459	0.193	0.459	1.807	2.741
11	0.125	0.771	0.505	0.571	0.222	0.371	1.636	2.571	1.919
12	0.25	0.493	0.641	0.293	0.359	1.507	2.359	1.707	0.641
13	0.375	0.629	0.364	0.429	1.495	2.229	1.495	0.429	0.364
14	0.5	0.393	0.541	1.607	2.259	1.407	0.259	0.193	0.541
15	0.625	0.629	1.778	2.429	1.495	0.229	0.081	0.429	0.364
16	0.75	1.907	2.641	1.707	0.359	0.093	0.359	0.293	0.641
17	0.875	2.771	1.919	0.571	0.222	0.371	0.222	0.571	1.919

	K	L	M	N	O	P	Q	R	S
6									
7									
8									
9									
10	0	1	0.1	0	0	0	0	0	0.1
11	1	0.05	0.354	0	0	0	0	0	0
12	2	0	0	0	0	0	0	0	0
13	3	0	0	0	0	0	0	0	0
14	4	0	0	0	0	0	0	0	0
15	5	0	0	0	0	0	0	0	0
16	6	0	0	0	0	0	0	0	0
17	7	0.05	0	0	0	0	0	0	0.354
18	0	0	0	0	0	0	0	0	0
19	1	0	0.354	0	0	0	0	0	0
20	2	0	0	0.25	0	0	0	0	0
21	3	0	0	0	0	0	0	0	0
22	4	0	0	0	0	0	0	0	0
23	5	0	0	0	0	0	0	0	0
24	6	0	0	0	0	0	0	-0.25	0
25	7	0	0	0	0	0	0	0	-0.35
26									
27									
28									

The 2D-FFT can be computed in a very straight way. Simply select a 16 x 8 array and insert the FFT2D where the input parameter is the given matrix (range B10:I17).

We can easily extract the harmonic components:

$$H(0,0) = 1$$

$$H(1,0) = 0.05$$

$$H(0,1) = 0.1$$

$$H(1,1) = 0.354 + 0.354j$$

$$H(2,2) = 0.25j$$

If we compute the inverse transform `DFT2D_INV("L10:S25")` we will obtain again the given starting matrix.

## 2D Inverse Discrete Fourier Transform

### =FFT2D\_INV (samples)

This function `FFT2D_INV` performs the inverse task of the `FFT2D`. It accepts as input an (2N x M) array having the real part in the first N rows and the imaginary part in the last N rows. It returns an (N x N) array

## Macro DFT (Discrete Fourier Transform)

This macro performs:

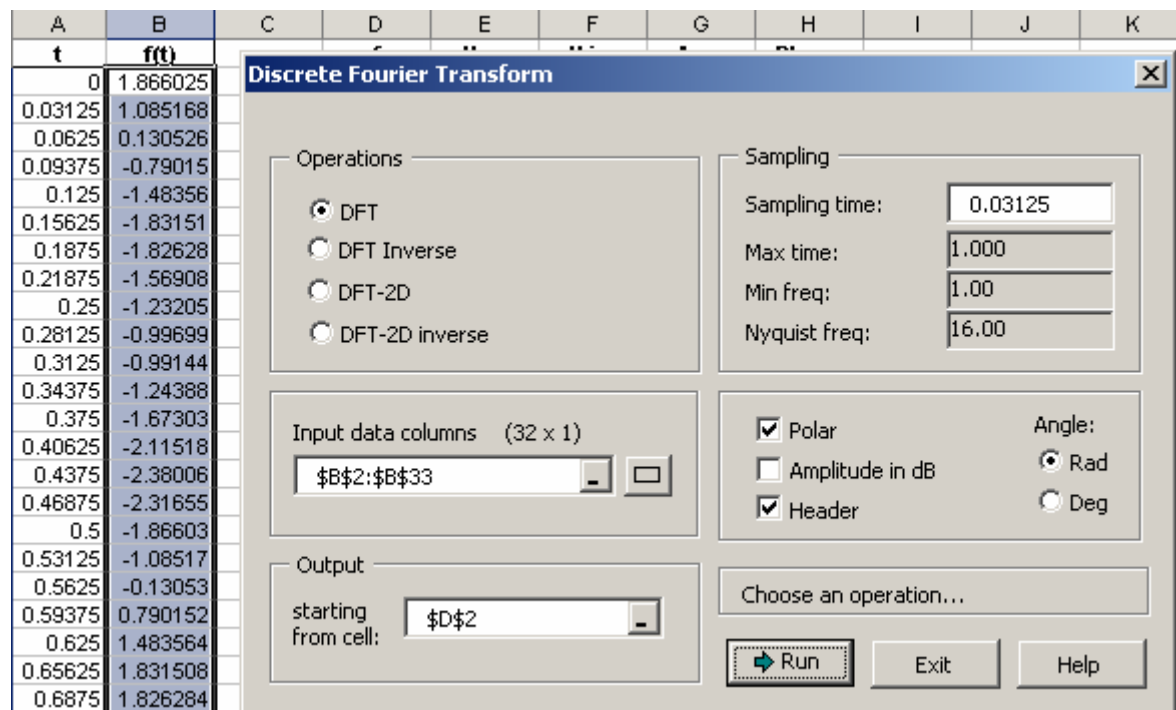
- the DFT of a data set of N samples
- the DFT-Inverse of a data set of N complex samples
- the 2D-DFT of a matrix of N x M samples
- the 2D-DFT-Inverse of a two matrices of N x M samples

### DFT

It works for any number N.

If N is a powers of 2 (8, 16, 32, 64, etc.) the macro uses the faster FFT algorithm and the elaboration is more efficient.

The use is quite simple. Select the vector of samples f(k) and then start this macro



The column "t" is not strictly necessary. If present, the macros use it to calculate the sampling parameters (see the top-right box).

Note that if you have a large input vector, you can select only the first cell f(1) and the macro automatically select the entire column.

The macro writes the result in the following way

D	E	F	G	H
f	H re	H im	Amp	Phase
0	0	0	0	0
1	0.5	0.866025	1	1.047198
2	0	0	0	0
3	0.433013	0.25	0.5	0.523599
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0

**f** = frequency sample

**Hre** = Real part of DFT transform

**Him** = Imaginary part of DFT transform

**Amp** = Amplitude (if "polar" is checked)

**Phase** = Phase (if "polar" is checked)

The amplitude can be converted in dB

$Amp_{dB} = 20 \cdot \log(Amp)$

## Operation DFT-inverse

In this case you have to select two columns: the real and imaginary part of the DFT ( $H_{re}$ ,  $H_{im}$ ). Then start the macro as usually.

If the DFT is in polar form (Amplitude, Phase), you have to checked the “polar” option and choose consequently the appropriate units: dB and angle

## Operation 2D-DFT

It works only for N and M integer power of 2

In this case you have to select a matrix of N x M values (do not select the axes-scales)

Then start the macro as usually.

If you want the DFT in polar form (Amplitude, Phase), you have to checked the “polar” option and choose consequently the appropriate units: dB and angle

	A	B	C	D	E	F	G	H	I
8									
9		0	0.13	0.25	0.38	0.5	0.63	0.75	0.88
10	0	2.01	0.74	0.39	0.46	0.19	0.46	1.81	2.74
11	0.13	0.77	0.51	0.57	0.22	0.37	1.64	2.57	1.92
12	0.25	0.49	0.64	0.29	0.36	1.51	2.36	1.71	0.64
13	0.38	0.63	0.36	0.43	1.49	2.23	1.49	0.43	0.36
14	0.5	0.39	0.54	1.61	2.26	1.41	0.26	0.19	0.54
15	0.63	0.63	1.78	2.43	1.49	0.23	0.08	0.43	0.36
16	0.75	1.91	2.64	1.71	0.36	0.09	0.36	0.29	0.64
17	0.88	2.77	1.92	0.57	0.22	0.37	0.22	0.57	1.92

The macro generates two matrices containing the real and imaginary parts of the 2D-DFT

	0	1	2	3	4	5	6	7
0	1	0.1	0	0	0	0	0	0.1
1	0.05	0.35	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0.05	0	0	0	0	0	0	0.35
0	0	0	0	0	0	0	0	0
1	0	0.35	0	0	0	0	0	0
2	0	0	0.25	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	-0.3	0
7	0	0	0	0	0	0	0	-0.4

## Operation 2D-DFT inverse

In this case you have to select a matrix of 2N x M values (do not select the axes values) containing both real and imaginary part.

Then start the macro as usually.

If the DFT is in polar form (Amplitude, Phase), you have to checked the “polar” option and choose consequently the appropriate units: dB and angle

## Macro Sampler

This is a simple but very useful macro for function sampling  
It can generate samples of functions such as:

$$f(x), f(x_1, x_2) \text{ or even more variables } f(x_1, \dots, x_m)$$

The samples can be arranged in a list and, for two variables only, also in a table  
Examples of lists and tables generated by this macro are shown in the following sheet

	A	B	C	D	E	F	G	H	I	J	
1		<b>x</b>	<b>f(x)</b>			<b>x1</b>	<b>x2</b>	<b>Function</b>			
2	<b>Start</b>	0	0		<b>Start</b>	0	1	1	Function Seed		
3	<b>Samples</b>	10			<b>Samples</b>	10	5				
4	<b>Period</b>	1.8			<b>Period</b>	1.8	1.6				
5	<b>Step</b>	0.2			<b>Step</b>	0.2	0.4				
6	<b>Cyclic</b>				<b>Cyclic</b>						
7											
8		<b>x</b>	<b>f(x)</b>			<b>x2</b> →	1	1.4	1.8	2.2	2.6
9		0	0		<b>x1</b> ↓	0	1	1.96	3.24	4.84	6.76
10		0.2	0.04			0.2	1.2	2.16	3.44	5.04	6.96
11		0.4	0.16			0.4	1.4	2.36	3.64	5.24	7.16
12		0.6	0.36			0.6	1.6	2.56	3.84	5.44	7.36
13		0.8	0.64			0.8	1.8	2.76	4.04	5.64	7.56
14		1	1			1	2	2.96	4.24	5.84	7.76
15		1.2	1.44			1.2	2.2	3.16	4.44	6.04	7.96
16		1.4	1.96			1.4	2.4	3.36	4.64	6.24	8.16
17		1.6	2.56			1.6	2.6	3.56	4.84	6.44	8.36
18		1.8	3.24			1.8	2.8	3.76	5.04	6.64	8.56
19											

The tables at the top are the skeletons to generate the samples-list or the samples-table just below. The skeleton contains the parameter for the sampler

<b>Start</b>	starting point of the variable $X_0$
<b>Samples</b>	number of samples to generate: N
<b>Period</b>	length of the sampling: P
<b>Step</b>	length between two consecutive point $H = X_1 - X_0$
<b>Cyclic</b>	True or False (default), specifies if the function is periodic with period P.

The difference between a cyclic or no-cyclic function is in the formula for the step calculation

$$S = P / N \quad \text{for cyclic function}$$

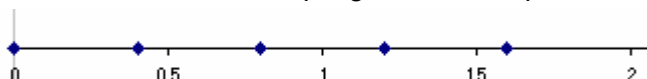
$$S = P / (N-1) \quad \text{for no-cyclic function}$$

For example, the sampling of  $N = 5$ , from  $X_0 = 0$  and  $P = 2$ , needs a step  $H = 0.5$



The first and the points, in this case, are always taken

But, for a periodic function, the same sampling needs a step of  $H = 0.4$



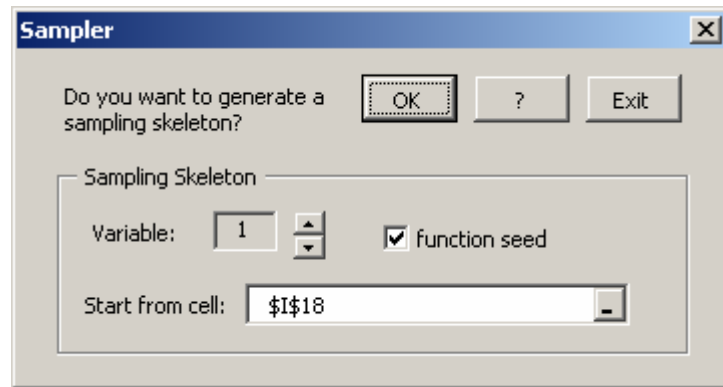
Practically, the last point  $X = 2$ , in this case, is discharged, because of being periodic, is  $f(0) = f(2)$ . Usually periodic functions require to set Cyclic = "True" for the FT analysis



## Xnumbers Tutorial

The skeleton can be drawn by hand or automatically. In this case you have only to give the number of variables that you need.

The check-box “Function seed” tells the macro to created also the cell in which you can insert the function to sample



A simple skeleton for one variable is:

	A	B	C	D	E
1		x	f(x)		
2	Start	0	0	Function Seed	
3	Samples	10			
4	Period	1.8			
5	Step	0.2			
6	Cyclic				

In the cell C2 you must insert the function  $f(x)$  to sample. The reference for the independent variable  $x$  is the cell B2. For example, if the function is  $y = x + 2x^2$  You have to insert the formula  
 $= B2 + 2 * B2^2$  in the cell C2

Parameters N (Samples), P (Period), H (Step) are not all independent. Only two parameters can be freely chosen.

The macro chooses the first two parameters found from top to bottom

The remain parameter is obtained by the step-formula

Synthetically you can have one of the following three cases

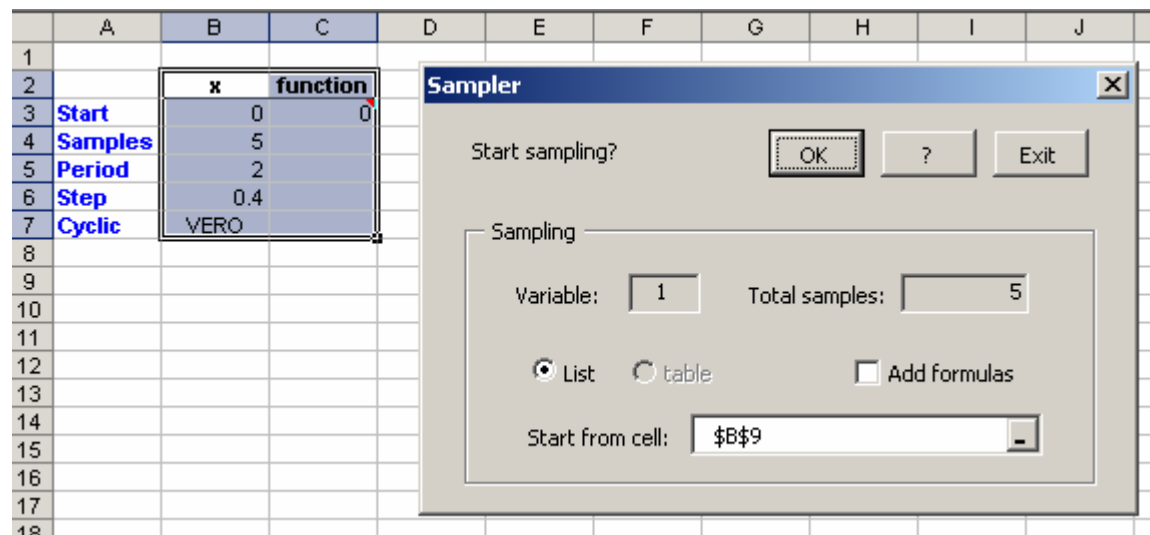
Given parameters	Obtained parameter
Samples, Period (N, P)	Step (H)
Samples, Step (N, H)	Period (P)
Period, Step (P, H)	Samples (N)

Look at the following three examples below for better explanation. The given parameters are in blue while the obtained parameter is in red.

H	I	J	K	L	M	N	O	P
	x	f(x)		x	f(x)		x	f(x)
Start	0	0		0	0		0	0
Samples	6			6			6	
Period	2			3			0.5	
Step	0.4			0.6			0.1	
Cyclic								
	x	f(x)		x	f(x)		x	f(x)
	0	0		0	0		0	0
	0.4	0.16		0.6	0.36		0.1	0.01
	0.8	0.64		1.2	1.44		0.2	0.04
	1.2	1.44		1.8	3.24		0.3	0.09
	1.6	2.56		2.4	5.76		0.4	0.16
	2	4		3	9		0.5	0.25

After you have set and filled the skeleton, select it and start the sampler macro again (remember that range must always have 6 rows, including the header)

The macro show the following window



The check-box “Add formula” tells to the macro to leave the formula in the sample set. Otherwise the sample set will contain only the values. Formulas can be add only for a monovariable list or for a table

## Data Integration (Romberg method)

**=IntegrDataR(x,y)**

**=IntRombergMat(x,y)**

The first function computes the integral of a discrete set of equidistant points  $(x_i, y_i)$  using the Romberg method

The set of point may be obtained by sampling with step h.

$$x_i = x_0 + i \cdot h, \quad y_i = f(x_i), \quad \text{for } i = 0, 1, 2, \dots, (2^p+1), \quad \text{where } p = 0, 1, 2, 3, \dots$$

Usually p is called the rank of Romberg integration

The second function returns the  $(p+1) \times (p+1)$  Romberg integration matrix R.

R(0,0)				
R(1,0)	R(1,1)			
R(2,0)	R(2,1)	R(2,2)		
R(3,0)	R(3,1)	R(3,2)	R(3,3)	
R(4,0)	R(4,1)	R(4,2)	R(4,3)	R(4,4)
.....	.....	.....	.....	.....

The first column  $R(p,0)$  of the above table contains the first integral approximation obtained by the trapezoidal rule with  $2^p+1$  points. The other columns are generated using the Richardson's extrapolation formula:

$$R_{p+1,j+1} = R_{p+1,j} + \frac{R_{p+1,j} - R_{p,j}}{4^j - 1}$$

The right-bottom  $R(p, p)$  element converges to the integral.

Relation between N (points), p (rank), and dim(R)

<b>p (rank)=&gt;</b>	0	1	2	3	4	5	6	7	8
<b>N (points)=&gt;</b>	2	3	5	9	17	33	65	129	257
<b>Dim(R) =&gt;</b>	1	2	3	4	5	6	7	8	9

In the following example we performs the numerical integration of the given data set  $(x_i, f_i)$ . We have also computed the Romberg matrix. From the last row it is evident the fast convergence of this method.

	A	B	C	D	E	F	G	H
1	<b>n</b>	<b>x</b>	<b>f(x)</b>		<b>Romberg matrix</b>			
2	0	0	1		1.299786802	0	0	0
3	1	0.2	0.993346654		1.367249492	1.389737055	0	0
4	2	0.4	0.973545856		1.383722783	1.389213881	1.389179002	0
5	3	0.6	0.941070789		1.38781761	1.389182552	1.389180464	1.389180487
6	4	0.8	0.896695114					
7	5	1	0.841470985				<b>=IntRombergMat(B2:B10;C2:C10)</b>	
8	6	1.2	0.776699238		<b>Integral =</b>	1.389180487		
9	7	1.4	0.703892664					
10	8	1.6	0.624733502		<b>=IntegrDataR(B2:B10;C2:C10)</b>			

By the way, the given data set was obtained by sampling of the function  $\sin(x)/x$ . So we have computed an approximation of the Sine-Integral for  $x = 1.6$

$$Si(x) = \int_0^x \frac{\sin(t)}{t} dt \quad Si(1.6) \cong 1.38918048587044...$$

Note that with only 9 points we have approximated the Sine-Integral with a precision better of 1e-9.

## Function Integration (Romberg method)

**=Integr\_ro(Funct, a, b, [Parm], [rank], [ErrMax])**

This function computes the numeric integral of a function f(x) by the Romberg method.

$$I = \int_a^b f(x)$$

The parameter Funct is a math expression string in the variable x, such as:

"x\*cos(x)", "1+x+x^2", "exp(-x^2)", ecc..

Remember the quote " " for passing a string to an Excel function.

Funct may be also a cell containing a string formula

Param contains values for parameters substitution (if there are)

Rank, from 1 to 16 (default), sets the maximum integration rank.

ErrMax (default 1E-15) , sets the maximum relative error.

For further details about writing a math string see [Math formula string](#)

The algorithm starts with rank =1 and continues incrementing the rank until it detects a stop condition.

$|R(p, p) - R(p, p-1)| < 10^{-15}$       *absolute error detect*  
 or  
 $(|R(p, p) - R(p, p-1)|) / |R(p, p)| < 10^{-15}$     if  $|R(p, p)| >> 1$     *relative error detect*  
 or  
 rank = 16

### Example

Compute the integral of x\*cos(x) for  $0 \leq x \leq 0.4$

`Integr_ro("x*cos(x)";0;0.4) = 0.0768283309263453`

This result is reached with rank =4 , s =16 sub-intervals, and an estimate error of about E= 3.75E-16

This function can also displays the number of sub-intervals and the estimate error, To see these values simply select three adjacent cells and give the CTRL+SHIFT+ENTER key sequence.

	D2	fx {=Integr(A2;B2;C2)}				
	A	B	C	D	E	F
1	f(x)	a	b	integr	Interv.	error
2	x*cos(x)	0	0.4	0.076828331	16	3.747E-16

## Function Integration (Double Exponential method)

---

**= Integr\_de(funcnt, a, b, [Param])**

This function<sup>11</sup> computes the numeric integral of a function  $f(x)$  by the Double Exponential method. This is particularly adapted for improper integrals and infinite, not oscillating integrals.

$$I = \int_a^b f(x)dx \quad I = \int_a^{+\infty} f(x)dx \quad I = \int_{-\infty}^{+\infty} f(x)dx$$

The parameter `funcnt` is a math expression string in the variable `x`, such as:

"`x*cos(x)`", "`1+x+x^2`", "`exp(-x^2)`", ecc.. .

Remember the quote " " for passing a string to an Excel function.

`Funcnt` may be also a cell containing a string formula

The limits "`a`" and "`b`" can also be infinite. In this case insert the string "`inf`"

`Param` contains labels and values for parameters substitution (if there are)

For further details about writing a math string see [Math formula string](#)

The Double Exponential method is a fairly good numerical integration technique of high efficiency adapt for integrating improper integrals, infinite integrals and "stiff" integrals having discontinue derivative functions.

This ingenious scheme, was introduced first by Takahasi and Mori [1974]

For finite integral, the formula, also called "***tanh-sinh transformation***" is the following

$$\int_a^b f(x)dx = \int_{-\infty}^{+\infty} f(x(t)) \cdot h(t)dt$$

where:

$$x(t) = \frac{b+a}{2} + \frac{b-a}{2} \tanh(\sinh(t)) \quad h(t) = \frac{b-a}{2} \frac{\cosh(t)}{\cosh^2(\sinh(t))}$$

Example

$$\int_0^1 x^{0.5} (1-x)^{0.3} dx = 0.474421154996...$$

The above integral is very difficult to compute because the derivative is discontinue at 0 and 1

The Romberg method would require more than 32.000 points to reach an accuracy of 1E-7. On the contrary, this function requires less then 100 points reaching the high accuracy of 1E-14

---

<sup>11</sup> This function uses the double exponential quadrature derived from the original FORTRAN subroutine INTDE of the DE-Quadrature (Numerical Automatic Integrator) Package , by Takuya OOURA, Copyright(C) 1996

## Xnumbers Tutorial

	A	B	C	D
1	<b>function</b>	<b>a</b>	<b>b</b>	<b>Integral</b>
2	$x^{0.5}*(1-x)^{0.3}$	0	1	0.474421155
3				
4		<code>=Integr_de(A2;B2;C2)</code>		

This function can also evaluate infinite and/or semi-infinite integral  
Example

$$\int_0^{\infty} x^{-n} dx$$

As known, the integral exist if  $n > 1$  and its value is  $I = 1/(n-1)$ . The parameter "n" is called "order of convergence".

For  $n = 1.1$  we get  $I = 10$

	A	B	C	D	E
1	<b>function</b>	<b>a</b>	<b>b</b>	<b>n</b>	<b>Integral</b>
2	$1/x^n$	1	inf	1.1	10
3					
4		<code>=Integr_de(A2;B2;C2;D1:D2)</code>			
5					

Note that we need to pass the parameter with its label "n". (Param = D1:D2)

This function cannot give reliable results if n is too close to 1. The minimum value is about  $n = 1.03$

For lower values the function returns "?".

The DE integration works very well for finite improper integral  
Example

$$\int_0^1 \ln(x^2) dx = \lim_{a \rightarrow 0^+} \int_a^1 \ln(x^2) dx = -2$$

	A	B	C	D
1	<b>function</b>	<b>a</b>	<b>b</b>	<b>Integral</b>
2	$\ln(x^2)$	0	1	-2
3				
4		<code>=Integr_de(A2;B2;C2)</code>		
5				

Note that the function  $f(x)$  is not defined for  $x = 0$

## Function Integration (mixed method)

**= Integr(Funct, a, b, [Param])**

This function computes the numeric integral of a function  $f(x)$  over a finite or infinite interval

$$\int_a^b f(x)dx \quad \int_a^{+\infty} f(x)dx \quad \int_{-\infty}^b f(x)dx \quad \int_{-\infty}^{+\infty} f(x)dx$$

This function can also works with improper integrals and piece-wise functions  
The parameter funct is a math expression string in the variable x, such as:

"x\*cos(x)", "1+x+x^2", "exp(-x^2)", ecc..

Remember the quote " " for passing a string to an Excel function.

Funct may be also a cell containing a string formula

The limits "a" and "b" can also be infinite. In this case, insert the string "inf"

Param contains labels and values for parameters substitution (if there are)

This function uses two quadrature algorithms

- 1) The double exponential method<sup>12</sup> (see function [integr\\_de](#) )
- 2) The adaptive Newton-Cotes schema (Bode's formula) (see macro [Integral\\_Inf](#) )

If the first method fails, the function switches on the second method

Oscillating functions, need specific algorithms. See [Integration of oscillating functions \(Filon formulas\)](#) and [Fourier's sine-cosine transform](#)

Example. Compute the integral of  $x \cdot \cos(x)$  for  $0 \leq x \leq 0.4$

	D2	fx {=Integr(A2;B2;C2)}				
	A	B	C	D	E	F
1	f(x)	a	b	integr	Interv.	error
2	x*cos(x)	0	0.4	0.076828331	16	3.747E-16

In the given interval the function is continuous, so its definite integral exists. This result is reached with rank = 4, s = 16 sub-intervals, and an estimate error of about 3.7E-16. This function returns the integral and can also displays the number of sub-intervals and the estimate error. To see these values simply select three adjacent cells and give the CTRL+SHIFT+ENTER keys sequence.

Note that the function Integr is surrounded by { } . This means that it returns an array

The function Integr can accept also parameters in the math expression string.  
See the example below.

<sup>12</sup> This function uses the double exponential quadrature derived from the original FORTRAN subroutines INTDE and INTDEI of the DE-Quadrature (Numerical Automatic Integrator) Package , by Takuya OOURA, Copyright(C) 1996

	A	B	C	D	E	F	G
1	<b>f(x)</b>	<b>a</b>	<b>b</b>	<b>k</b>	<b>integr</b>		
2	$x \cdot \cos(k \cdot x)$	0	0.4	2	0.067647896	<b>=Integr(A2;B2;C2;D1:D2)</b>	
3							
4	<b>f(x)</b>	<b>a</b>	<b>b</b>	<b>w</b>	<b>q</b>	<b>integr</b>	
5	$(1+w \cdot x) / (1+q \cdot x^2)$	0	0.4	0.8	0.25	0.457544261	
6							
7				<b>=Integr(A5;B5;C5;D4:E5)</b>			

Note that we must include the parameter labels in order to distinguish the parameters "k", "w", and "q". The integration variable is always "x"

### Beware of the poles

Before attempting to evaluate a definite integral, we must always check if the integral exists. The function integr does not perform this check and the result may be wrong. In other words, we have to make a short investigation about the function that we want to integrate. Let's see the following example

Assume to have to compute the following integrals

$$\int_0^{1/2} \frac{2}{2x^2-1} dx \quad , \quad \int_0^1 \frac{2}{2x^2-1} dx$$

We show that the first integral exists while, on the contrary, the second does not exist

For  $x_p = \sqrt{2}/2 \cong 0.707...$  the function has a pole; that is:

$$\lim_{x \rightarrow x_p^-} \left( \frac{2}{2x^2-1} \right) = -\infty \quad , \quad \lim_{x \rightarrow x_p^+} \left( \frac{2}{2x^2-1} \right) = +\infty$$

The first integral exists because its interval  $[0, 0.5]$  does not contain the pole and the function is continuous in this interval. We can compute its exact value:

$$\int \frac{2}{2x^2-1} dx = \frac{\sqrt{2}}{2} \log \left( \frac{|\sqrt{2}x-1|}{|\sqrt{2}x+1|} \right) \Rightarrow \int_0^{1/2} \frac{2}{2x^2-1} dx = \sqrt{2} \log(\sqrt{2}-1)$$

	A	B	C	D
1	<b>a =</b>	0		
2	<b>b =</b>	0.5		
3	<b>f(x) =</b>	$2/(2 \cdot x^2 - 1)$		
4	<b>integral =</b>	-1.24645048028	<b>=Integr(B3;B1;B2)</b>	
5	<b>refer. =</b>	-1.24645048028		
6	<b>error =</b>	8.43769E-15		
7				

In this situation the function **Integr** returns the correct numeric result with an excellent accuracy, better than 1E-14.

For this result the integration algorithm needs 128 sub-intervals

The interval of the second integral contains the pole, so we have to perform some more investigation. Let's begin to examine how the integral function approaches the pole  $x_p$  taking separately the limit from the right and from the left

$$\lim_{x \rightarrow x_p^-} \log \left( \frac{|\sqrt{2}x-1|}{|\sqrt{2}x+1|} \right) = -\infty \quad , \quad \lim_{x \rightarrow x_p^+} \log \left( \frac{|\sqrt{2}x-1|}{|\sqrt{2}x+1|} \right) = -\infty$$



## Xnumbers Tutorial

As we can see the both limits are infinite, so the second integral does not exist. Note that if we apply directly the fundamental integral theorem we would have a wrong result:

$$\int_0^1 \frac{2}{2x^2 - 1} dx \stackrel{\text{wrong!}}{=} \left[ \frac{\sqrt{2}}{2} \log \left( \frac{|\sqrt{2}x - 1|}{|\sqrt{2}x + 1|} \right) \right]_0^1 = \sqrt{2} \ln(\sqrt{2} - 1)$$

Let's see how the function **integr** works in this case.

	A	B	C	D
1	a =	0		
2	b =	1		
3	f(x) =	2/(2*x^2-1)		
4	integral =	19.13524077932	65536	2.366E-10
5	refer. =		{=Integr(B3;B1;B2)}	
6	error =	19.13524078		
7				

The numeric result is, of course, completely wrong because the given integral goes to the infinity. But, even in this situation, this function gives us an alert: the sub-intervals have reached the maximum limit of 65536 ( $2^{16}$ ). So the result accuracy must be regarded with a reasonable doubt.

## Complex Function Integration (Romberg method)

### =cplxintegr(Funct, a, b)

This function returns the numeric integral of a complex function  $f(z)$  by the Romberg method.

$$F(b) - F(a) = \int_a^b f(z) dz$$

The integration function **Funct** must be a string in the variable  $z$  and can be defined mixing all arithmetic operators, common elementary functions and complex numbers like:

"z\*cos(z)", "1+(1+i)\*z+z^2", "exp(-z^2)", ecc...

Remember the quote "" for passing a string to an Excel function.

Parameters "a" and "b" can be real or complex. Complex values are inserted as arrays of two cells.

Example: Evaluate the following integral

$$\int_{1-i}^{1+i} \frac{1+i}{z^2} dz$$

Because the integration function is analytic, then the given integral is independent from the integration path. Therefore it can be calculated by the function **cplxintegr**

## Xnumbers Tutorial

	A	B	C
1	f(z) =	(1+i)/z^2	
2		re	im
3	a =	1	-1
4	b =	1	1
5	integral =	-1	1
6	refer. =	-1	1
7	error =	3.220E-15	5.551E-16
8		{=cplxintegr(B1,B3:C3,B4:C4)}	
9			

The exact result is the complex number  $(-1+i)$

Note that, thanks to the excellent accuracy, the result is shown exactly even if it is intrinsically approximated

## Data Integration (Newton-Cotes)

### =IntegrDataC(x,y, [Degree])

This function returns the integral of a discrete set of points ( $x_i, y_i$ ) using the Newton-Cotes formulas. The points may be equidistance or random. The parameter degree, from 1(default) to 10, set the order of the Newton-Cotes formula written as:

$$\int_{x_0}^{x_0 + n h} f(x) dx = \frac{h}{K} \cdot \sum_{j=0}^n f_j \cdot b_j$$

where  $f_i = y_i$ ,  $h$  is the integration step,  $n$  is the degree; the coefficients ( $b_j, K$ ) can be extracted from the following table:

Degree	1	2	3	4	5	6	7	8	9	10
K	2	3	8	45	288	140	17280	14175	89600	299376
b0	1	1	3	14	95	41	5257	3956	25713	80335
b1	1	4	9	64	375	216	25039	23552	141669	531500
b2		1	9	24	250	27	9261	-3712	9720	-242625
b3			3	64	250	272	20923	41984	174096	1362000
b4				14	375	27	20923	-18160	52002	-1302750
b5					95	216	9261	41984	52002	2136840
b6						41	25039	-3712	174096	-1302750
b7							5257	23552	9720	1362000
b8								3956	141669	-242625
b9									25713	531500
b10										80335

As we can see, for degree=1, the Newton-Cotes formula coincides with the trapezoidal rule and, for degree = 2, with the popular Cavalieri-Simpson formula.

#### Trapezoid rule

$$h = x_1 - x_0$$

$$\int_{x_0}^{x_1} f(x) \cong \frac{h}{2} (f_0 + f_1)$$

#### Cavalieri-Simpson rule

$$h = \frac{x_2 - x_0}{2}$$

$$\int_{x_0}^{x_2} f(x) \cong \frac{h}{3} (f_0 + 4f_1 + f_2)$$

For degree = 4, the table gives the Bode's rule

$$h = \frac{x_4 - x_0}{4}$$

$$\int_{x_0}^{x_4} f(x) dx \cong \frac{h}{45} (14f_0 + 64f_1 + 24f_2 + 64f_3 + 14f_4)$$

Using the IntegrDataC is very easy.

Example. Given the data table (x y) of pag 142, calculate the integral with the Newton-Cotes formulas of degree = 1, 2, 4, 6

We already know that the table is the sampling of the function  $\sin(x)/x$  with step 0.2 and that the result approximates the function  $\text{Si}(1.6) \cong 1.38918048587044$ . Using the Romberg's method we have computed the integral with an accuracy better than 1E-9

## Xnumbers Tutorial

Let's see now how the Newton-Cotes formulas work.

	A	B	C	D	E	F	G
1	n	x	f(x)		degree	Integral	error
2	0	0	1		1	1.387817610	1.36E-03
3	1	0.2	0.993346654		2	1.389182552	2.07E-06
4	2	0.4	0.973545856		4	1.389180464	2.21E-08
5	3	0.6	0.941070789		6	1.389180487	9.87E-10
6	4	0.8	0.896695114				
7	5	1	0.841470985				
8	6	1.2	0.776699238				
9	7	1.4	0.703892664				
10	8	1.6	0.624733502				
11							

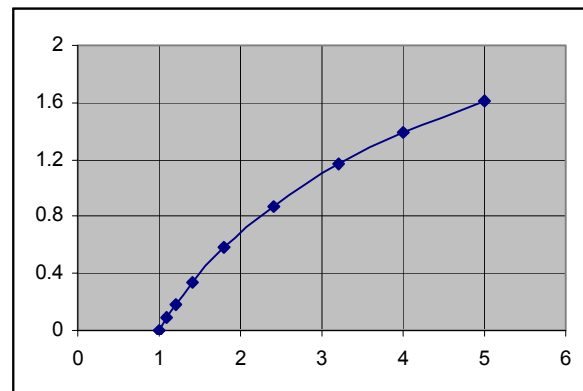
=IntegrDataC(B2:B10;C2:C10;E5)

As we can see, the convergence to the exact result is evident. The most accurate result is reached with the 6<sup>th</sup> degree Newton-Cotes formula. We observe that the accuracy is comparable with those of the Romberg method. From experience we observe that often the Romberg method gives a global accuracy comparable with the Newton-Cotes formulas between 4<sup>th</sup> and 6<sup>th</sup> order.

Differently from IntegrDataR (Romberg method), the IntegrDataC is suitable to work with random samples

Example. Given the data table (x y) , approximate the integral with the Cavalieri-Simpson formula

x	y
1	0
1.1	0.09531018
1.2	0.182321557
1.4	0.336472237
1.8	0.587786665
2.4	0.875468737
3.2	1.16315081
4	1.386294361
5	1.609437912



Note that the data points are not equidistant

	A	B	C	D	E
1	x	y			
2	1	0		integral	
3	1.1	0.09531018		4.04688	
4	1.2	0.182321557			
5	1.4	0.336472237			
6	1.8	0.587786665			
7	2.4	0.875468737			
8	3.2	1.16315081			
9	4	1.386294361			
10	5	1.609437912			
11					

=IntegrDataC(A2:A10;B2:B10;3)

The points have been extracted from the function

$y = \ln(x)$  .

Thus the exact integral is

$$5 \cdot \ln(5) - 4 \approx 4.0471896$$

## Data integration for random point.

For a distribution of set of points  $(x_i, y_i)$  not equidistant, we cannot use directly the Newton-Cotes formulas for fixed step.

In that case, **IntegrDataC** reorganizes the random data samples in equidistant data samples and after that, computes the integral using the standard formulas for fixed step

Random Samples	Converted to	Equispaced Samples
$\{ (x_i, y_i) ; i = 0, 1, \dots, n \}$	$\Rightarrow$	$\{ (x_i = x_0 + i h, y_i(x_i)) ; i = 0, 1, \dots, m \}$

For the computation of the function  $f(x_0 + i h)$  at the equispaced fixed points, **IntegrDataC** uses the Aitken's Interpolation algorithm.

### Aitken's interpolation algorithm.

Given a set of points:

$$f(x) \equiv \{ (x_i, y_i) \quad i = 0, 1, \dots, n \}$$

This method is used to find the interpolation  $y_p = f(x_p)$  at the wanted value  $x_p$ . It is efficient as the Newton formula, and it is also easy to code.

```

For j = 1 To n - 1
  For i = j + 1 To n
    y(i) = y(j) * (x(i) - xp) - y(i) * (x(j) - xp) / (x(i) - x(j))
  Next i
Next j
yp = yi(n)

```

## Function Integration (Newton-Cotes formulas)

### =Integr\_nc(funcnt, a, b, Intervals, [Degree])

This function returns the numeric integral of a function  $f(x)$  using the Newton-Cotes formulas.

$$F(b) - F(a) = \int_a^b f(x) dx$$

The parameter **Funcnt** is a math expression string in the variable  $x$ , such as:

"x\*cos(x)", "1+x+x^2", "exp(-x^2)", ecc...

Remember the quote "" for passing a string to an Excel function.

Funcnt may be also a cell containing a string formula

The parameters "a" and "b" are the limits of integration interval

The parameter "Intervals" sets the number of sub-intervals dividing the integration interval.

The parameter degree, from 1(default) to 10, set the order of the Newton-Cotes formula. The degree = 1 coincides with the Trapezoidal rule

The degree = 2 coincides with the Cavalieri-Simpson formula; the degree = 4 with the Bode's rule

Remember that the total knots of the function computation is:

$$\text{knots} = \text{Intervals} \times \text{Degree} + 1$$

Example: Approximate the following integral using 10 sub-intervals and three different methods: trapezoidal, Cavalieri-Simpson, and the Bode's rule.

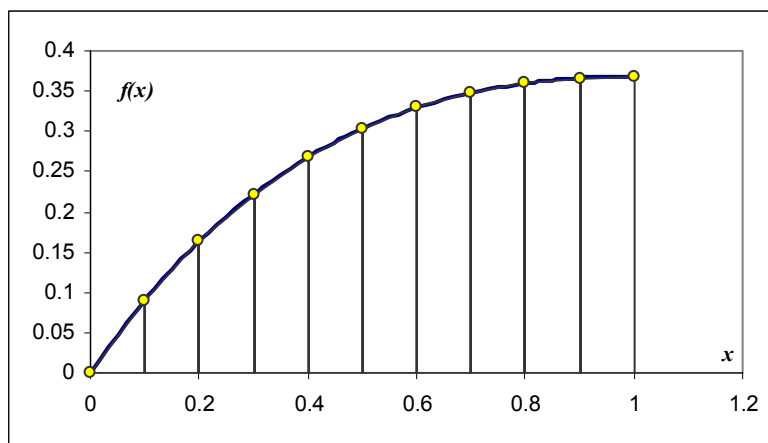
$$\int_0^1 x \cdot e^{-x} dx$$

The indefinite integral is known in a closed form:

$$\int x \cdot e^{-x} dx = -(x+1)e^{-x}$$

So we can compare the exact result, that is  $1 - 2e^{-1} \cong 0.264241117657115356$

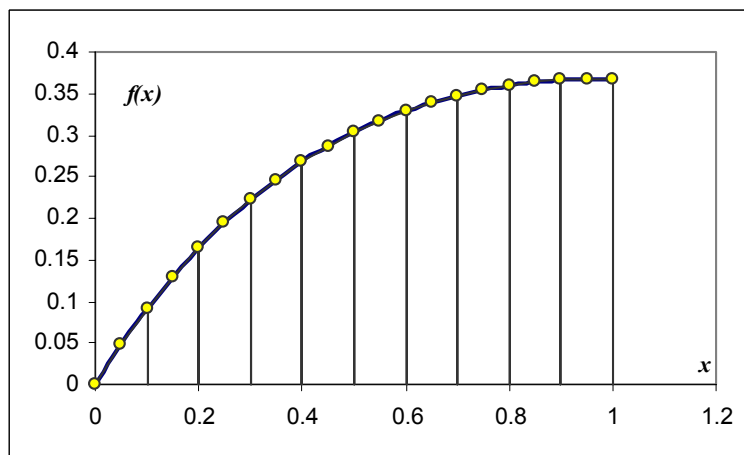
`Integr_nc("x*exp(-x)",0,1,10,1) = 0.263408098685072 (8.3E-04)`



The trapezoidal rule, with 10 sub-intervals, requires 2 knots for each sub-interval for a total of 11 function evaluations (11 knots)

The accuracy is better than 1E-3

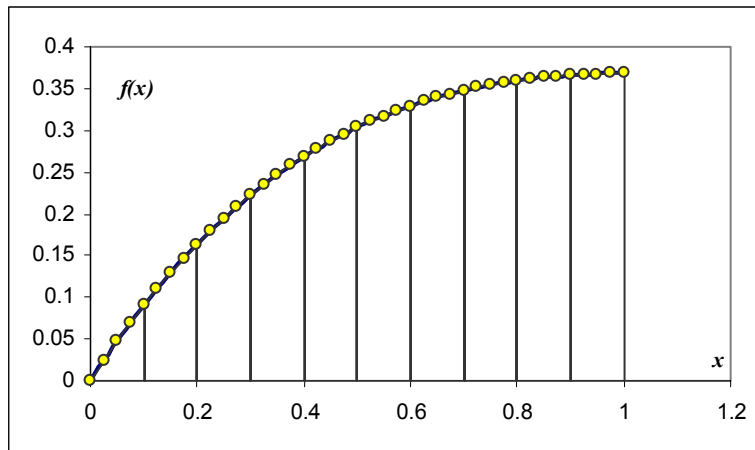
`Integr_nc("x*exp(-x)",0,1,10,2) = 0.264241039074082 (7.8E-08)`



The Cavalieri-Simpson rule, with 10 sub-intervals, requires 3 knots for each sub-interval for a total 21 function evaluation (21 knots)

The accuracy is better than 1E-7

```
Integr_nc("x*exp(-x)",0,1,10,4) = 0.264241117655293 (1.8E-12)
```



The Bode's rule, with 10 sub-intervals, requires 5 knots for each sub-interval for a total of 41 function evaluation (41 knots).

The accuracy is better than 2E-12

## Integration: symbolic and numeric approaches

The usual approach to the calculation of the definite integral involves two steps: the first is the construction of the symbolic anti-derivative  $F(x)$  of  $f(x)$

$$F(x) = \int f(x)dx$$

and the second step is the evaluation of the definite integral applying the fundamental integration theorem.

$$\int_a^b f(x)dx = F(b) - F(a)$$

This approach can only be adopted for the set of the functions of which we know the anti-derivative in a closed form. For the most  $f(x)$ , the integral must be approximated either by numerical quadrature or by some kind of series expansion.

It is usually accepted that symbolic approaches, when possible, gives more accurate result than the numeric one. This is not always true. Even if the symbolic anti-derivative is known in a closed form, it may often be unsuitable for further numerical evaluation. In particular, we have cases in which such "exact" answers when numerically evaluated give less accurate results than numerical quadrature methods<sup>13</sup> Let's see. Assume to have the following integral functions

$$F(x) = \int \frac{3x^2}{x^6 + 1} dx = \arctan(x^3) + c$$

We want to calculate the definite integral between  $a = 2000$  and  $b = 2004$ . The analytic approach gives

$$F(b) - F(a) = \arctan(b^3) - \arctan(a^3)$$

In the following worksheet we have compared the evaluations with the exact anti-derivative and the numerical quadrature with the Bode's rule

<sup>13</sup> "Improving Exact Integral from Symbolic Algebra System", R.J. Fateman and W. Kaham, University of California, Berkeley, July 18, 2000

## Xnumbers Tutorial

In the cell C2 we have inserted the anti-derivative function

=ARCTAN(B2^3)-ARCTAN(A2^3)

In the cell C2 we have inserted the Bode formula with 20 intervals

=Integr\_nc(D1;A2;B2;20;4)

	A	B	C	D
1	a	b	Integral F(b)-F(a)	$3*t^2/(t^6+1)$
2	2000	2004	7.4718009557E-13	7.47009970083E-13
3		error =	2.277E-04	9.011E-13
4				
5		ref. =	7.4700997008377E-13	

In the cell C5 we have also inserted the reference integral value

As we can see, the more accurate result is those obtained with the numerical quadrature; surprisingly, it is more than 200 millions times more accurate than the one of the exact method!

It is evident from this example that only the symbolic integration could not resolve efficiently the problem. For numerical integration the quadrature methods are often more efficient and accurate.



## Integration of oscillating functions (Filon formulas)

**=Integr\_fsин(Funct, a, b, k, Intervals)**

**=Integr\_fcos(Funct, a, b, k, Intervals)**

Oscillating functions can reserve several problems for the common polynomial integration formulas. The Filon's formulas is adapt to compute efficiently the following integrals.

$$\int_a^b f(t) \cdot \cos(k t) dt \quad , \quad \int_a^b f(t) \cdot \sin(k t) dt$$

for  $k = 1, 2, 3 \dots N$

The parameter Funct is a math expression string in the variable x, such as:

"x\*cos(x)", "1+x+x^2", "exp(-x^2)", ecc. . .

Remember the quote " " for passing a string to an Excel function.

Funct may be also a cell containing a string formula

The parameters "a" and "b" are the limits of integration interval

The parameter "k" is a positive integer

The parameter "Intervals" sets the number of sub-intervals dividing the integration interval.

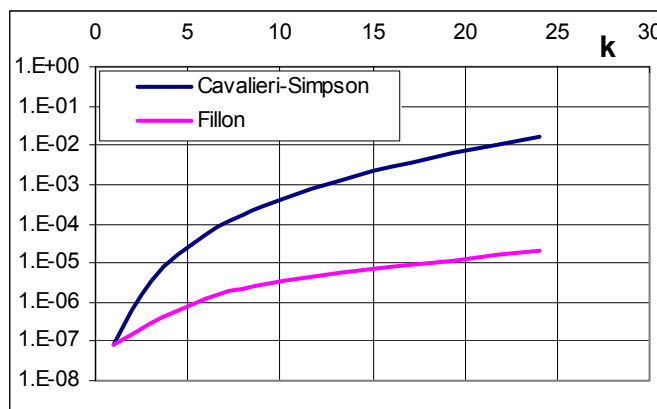
Remember that the total nodes of function computation is:

$$Nodes = Intervals \times 2 + 1$$

To understand the effort in this kind of numerical integration let's see this simple test. Assume we have to numerically evaluate the following integral for several integer values of k, with  $0 < k < 25$

$$\int_0^{\pi} x^4 \cdot \cos(k t) dt$$

If we perform the computation with the Cavalieri-Simpson formula (80 nodes) and with the Filon formula (80 nodes), we get the following result



### Relative error versus k

As we can see, the relative error increase with the number k much more for the Cavalieri-Simpson rule than the Filon formula.

For  $k = 24$  the first formula should have at least 400 nodes to reach the same accuracy of the Filon formula.

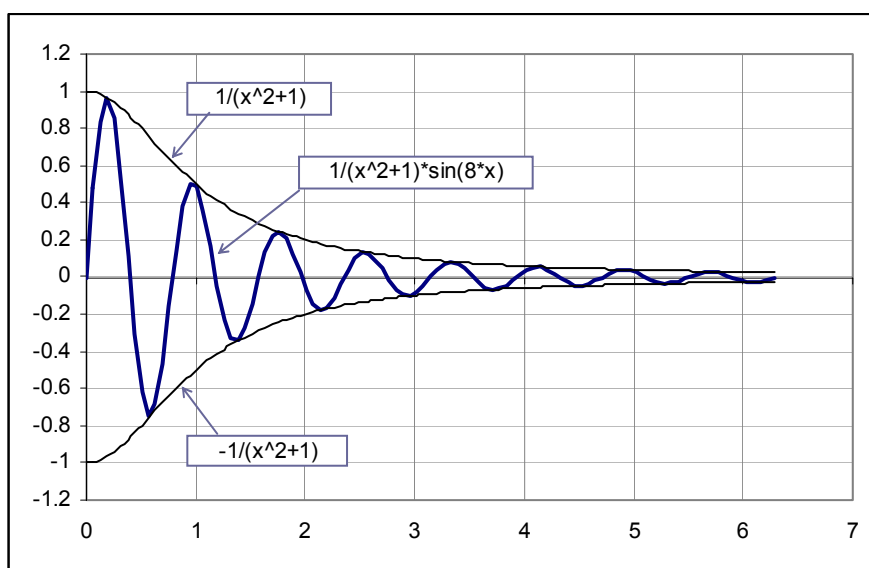
Example: evaluate the integral of the following oscillating function

$$\int_0^{2\pi} \frac{1}{x^2 + 1} \cdot \sin(8x)$$

that can be rearranged as

$$\int_0^{2\pi} g(x) \cdot \sin(8x) \quad \text{where} \quad g(x) = \frac{1}{x^2 + 1}$$

The plot of the integration function and the envelope function  $g(x)$  are shown in the following graph



Below, a simple arrangement to compute the given integral

	A	B	C	D	E	F
1						
2	<b>Fillon's integration of oscillating function</b>					
3						
4	<b>g(x)</b>	<b>a</b>	<b>b</b>	<b>k</b>	<b>Integral</b>	
5	1/(x^2+1)	0	6.28318531	8	0.12692412	
6						
7		=Integr_fsин(A5;B5;C5;D5)				
8						

The approximate error is less then 1E-8, with 300 intervals (default)

## Integration of oscillating functions (Fourier transform)

= **Fourier\_sin**(funct, k, [a], [param])

= **Fourier\_cos**(funct, k, [a], [param])

These functions<sup>14</sup> perform the numerical integration of oscillating functions over infinite intervals

$$\int_a^{+\infty} f(x) \cdot \sin(k \cdot x) dx \qquad \int_a^{+\infty} f(x) \cdot \cos(k \cdot x) dx$$

If a = 0 (default) , these integrals are called "*Fourier's sine-cosine transforms*"

The parameter funct is a math expression defining the function f(x), not oscillating and converging to 0 for x approaching to infinity:

"1/x", " 1/(8\*x^2)", " exp(-b\*x)", ecc.. .

Remember the quote " " for passing a string to an Excel function. Funct may be also a cell containing a string formula

The parameter "k" is a positive number

The "Param" contains labels and values for parameters substitution (if there are)

These functions return "?" if the integral is not converging or if they cannot compute the integral with sufficient accuracy

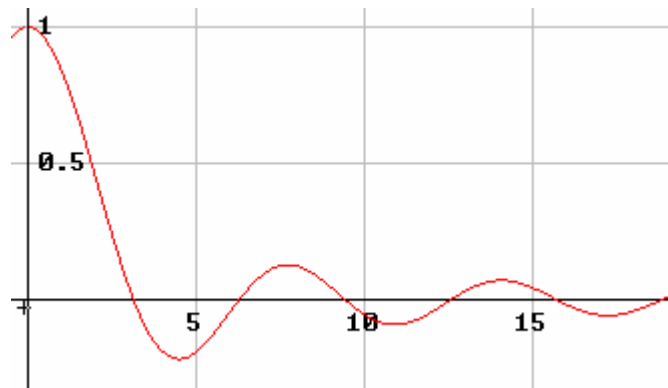
For finite integration see also [Integration of oscillating functions \(Filon formulas\)](#)

Example. Prove that is

$$\int_0^{+\infty} \frac{\sin x}{x} dx = \frac{\pi}{2}$$

The graph of the integration functions is at the right.

Numerically speaking, this integral is very difficult to calculate for many algorithms.



For example, the Bode adaptive quadrature needs more than 10.000 points for getting accuracy of about 1E-4. The **Fourier\_sin** function on the contrary is very efficient for this kind of integral

The integral can be arranged in the following form

<sup>14</sup> These functions use the double exponential quadrature derived from the original FORTRAN subroutine INTDEO of the DE-Quadrature (Numerical Automatic Integrator) Package , by Takuya OOURA, Copyright(C) 1996

$$\int_0^{+\infty} \frac{\sin x}{x} dx = \int_0^{+\infty} \frac{1}{x} \sin x dx$$

That is the Fourier's sine transform of  $1/x$

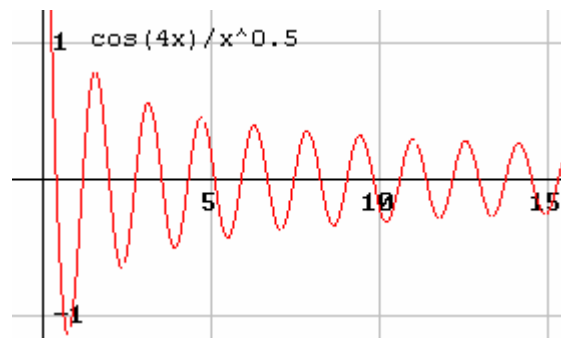
	A	B	C
1	<b>function</b>	<b>k</b>	<b>Integral</b>
2	1/x	1	1.570796327
3			
4	=Fourier_sin(A2;B2)		

We see that the accuracy is better than 1E-15. Note that the function automatically multiply the integration function  $f(x)$  for the factor  $\sin(k*x)$ . So we have only to write the  $f(x)$

Example. Verify that is

$$\int_0^{+\infty} \frac{\cos 4x}{\sqrt{x}} dx = \sqrt{\frac{\pi}{8}}$$

The graph of the integration functions is  
Observe that the integration function goes to infinity for  $x$  approaching to 0.



Numerically specking this function is "terrible".  
The integral can be arranged in the following form

$$\int_0^{+\infty} \frac{\cos 4x}{\sqrt{x}} dx = \int_0^{+\infty} \frac{1}{\sqrt{x}} \cos 4x dx$$

That is the Fourier's cosine transform of  $1/x^{0.5}$

	A	B	C
1	<b>function</b>	<b>k</b>	<b>Integral</b>
2	x^ (-0.5)	4	0.626657069
3			
4	=Fourier_cos(A2;B2)		

The accuracy is better than 1E-15

## Infinite Integration of oscillating functions

Generally, the infinite integration of real functions having a certain type of infinite oscillating tails may give some problem even to the most efficient quadrature algorithms. These problems can be avoided adopting specific integration tricks  
Let's see some of them.

Example. Assume to calculate the following integral

$$\int_0^{+\infty} \frac{\cos(x) - \cos(2x)}{x} dx$$

The integration function covers to zero but it contains two oscillating terms. So we cannot use directly the `integr` or `integr_de` function because they returns "?"  
For solving we can use the Fourier's cosine transform, separating each oscillating term.

The given integral can be re-arranged in the following way

$$\int_0^{+\infty} \frac{\cos(x) - \cos(2x)}{x} dx = \int_0^1 \frac{\cos(x) - \cos(2x)}{x} dx + \int_1^{+\infty} \frac{\cos(x)}{x} dx + \int_1^{+\infty} -\frac{\cos(2x)}{x} dx$$

Note that the last two integrals cannot have the lower limit 0 because they do not converge for x approaching to 0.

The first integral can be evaluated with the **integr** function and the two last integrals are evaluated with the **Fourier\_cos** function with a = 1. Let's see the following spreadsheet arrangement

	A	B	C	D	E
42	<b>Evaluation of oscillating infinite integrals</b>				
43					
44	integration function	a	b	l1	
45	(cos(x)-cos(2*x)) / x	0	1	0.607570275	=Integr(A45;B45;C45)
46					
47	integration function	a	k	l2	
48	1/x	1	1	-0.337403923	=Fourier_cos(A48;C48;B48)
49					
50	integration function	a	k	l3	
51	-1/x	1	2	0.422980829	=Fourier_cos(A51;C51;B51)
52					
53	I = l1 + l2 + l3 =			0.693147181	

Compare the accuracy with the exact result  $I = \ln(2)$

Example. Calculate the following integral

$$\int_0^{+\infty} \frac{\sin^4(x)}{x^2} dx$$

Remembering that is

$$\sin^4(x) = \frac{3}{8} - \frac{\cos(2x)}{2} + \frac{\cos(4x)}{8}$$

The given integral can be arranged as

$$\int_0^{+\infty} \frac{\sin^4 x}{x^2} dx = \int_0^1 \frac{\sin^4 x}{x^2} dx + \int_1^{+\infty} \frac{3}{8x^2} dx + \int_1^{+\infty} -\frac{\cos(2x)}{2x^2} dx + \int_1^{+\infty} \frac{\cos(4x)}{8x^2} dx$$

## Xnumbers Tutorial

The first and second integral can be evaluated with the **integr** function and the two last integrals are evaluated with the **Fourier\_cos** function with a = 1. Let's see the following spreadsheet arrangement

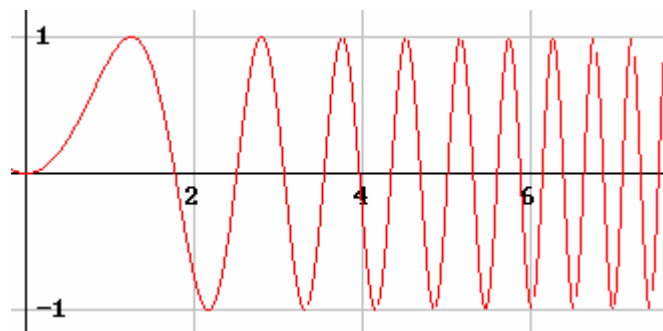
	A	B	C	D	E
68	integration function	a	b	I	
69	$\sin(x)^4/x^2$	0	1	0.224943442	=Integr(A69;B69;C69)
70	$3/(8*x^2)$	1	inf	0.375	=Integr(A70;B70;C70)
71					
72	integration function	a	k	I	
73	$-1/(2*x^2)$	1	2	0.173456768	=Fourier_cos(A73;C73;B73)
74	$1/(8*x^2)$	1	4	0.011997953	=Fourier_cos(A74;C74;B74)
75					
76	$I = I_1 + I_2 + I_3 =$			0.785398163	

Compare the accuracy with the exact result  $I = \pi/2$

Example. Calculate the following integral

$$\int_0^{+\infty} \sin(x^2) dx$$

This function oscillates very badly. Note that the function does not converge to zero, oscillating continuously from 1 and -1, but we can show that its integral is finite.



Let's perform the substitution

$$x^2 = t \Rightarrow x = \sqrt{t} \Rightarrow dx = \frac{1}{2\sqrt{t}} dt$$

So, the given integral becomes

$$\int_0^{+\infty} \sin(x^2) dx = \int_0^{+\infty} \frac{\sin(t)}{2\sqrt{t}} dt$$

That can be easily computed by the Fourier's cosine transform

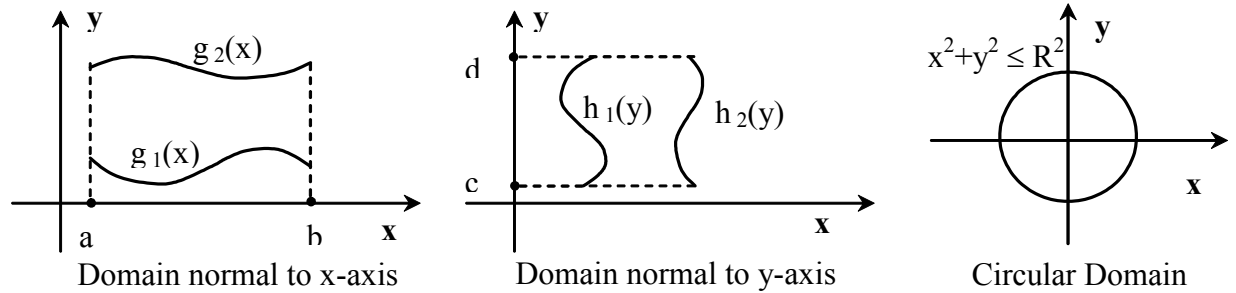
	A	B	C	D	E
82	integration function	a	k	I	
83	$1/(2*x^{0.5})$	0	1	0.626657069	=Fourier_sin(A83;C83;B83)

Compare the accuracy with the exact result  $I = (\pi/8)^{1/2}$

## Double Integral

### 2D Integration for Normal Domains

Xnumbers contains routines for integrating bivariate functions  $f(x, y)$  over a normal domain (normal to the x-axis and/or to the y-axis) or a circular domain.



For those kinds of 2D-domains the integration formulas can be re-written as the following

$$\iint_{D_x} f(x, y) ds = \int_a^b \int_{g_1(x)}^{g_2(x)} f(x, y) dy dx$$

$$\iint_{D_y} f(x, y) ds = \int_c^d \int_{h_1(y)}^{h_2(y)} f(x, y) dx dy$$

$$\iint_C f(x, y) ds = \int_0^{2\pi} \int_0^R f(\rho \cos(\theta), \rho \sin(\theta)) \rho d\rho d\theta$$

Note that a normal domain implies that - at least - one axis must have constant limits. Rectangular domains are a sub-case of normal domains in which both axes have constant limits.

The routines are the macro **Integr2D** - adapted for integrating smooth functions  $f(x, y)$  – and its function version **Integr\_2D** that uses the same bidimensional Romberg algorithm, but limited to about 65.000 points.

## Double Integration macro

### Integr2D()

This macro performs the numerical integration of a smooth, regular function  $f(x, y)$  over a plane normal domain  $D(x, y)$ .

$$\int_a^b \int_c^d f(x, y) dx dy$$

The integration functions  $f(x, y)$  and – eventually – also the bounding limits – a, b, c, d – can be written in symbolic expression

## Xnumbers Tutorial

The integration function can be:

- bi-variate functions like  $x^2+y^2-x*y$ ,  $\log(1+x+y)$ ,  $1/(1+x^2+2*y^2)$ , etc.
- constant numbers like 0, 2, 1.5, 1E-6, etc.
- constant expressions like  $1/2$ ,  $\sqrt{2}+1$ ,  $\sin(0.1)$ , etc.

Boundary limits can be:

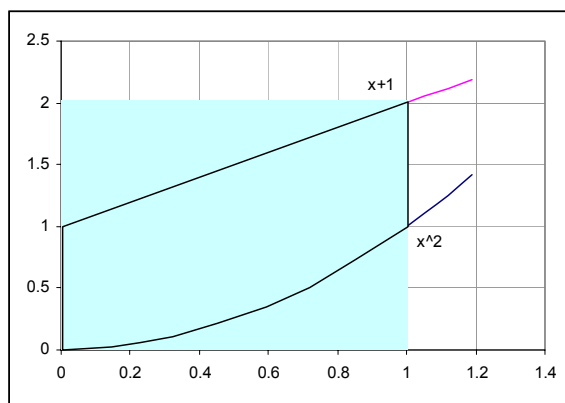
- constant numbers like 0, 2, 10, 3.141, etc.
- constant expressions like  $1/2$ ,  $\sqrt{2}+1$ ,  $\pi$ ,  $\sin(1/2*\pi)$ ,  $\exp(1)$ , etc.
- mono-variable functions like  $x/2$ ,  $3y-10$ ,  $x^2+x-1$ , etc.

A normal domain has, at least, two constant boundary limits.

Function and limits can be passed to the macro directly or by reference. That is: you can write directly the symbolic expressions or constants into the input-fields or you can pass the cells containing the expressions. This second mode is more easy and straight. There is also a function version of this routine.

Let's see how it works

Approximate the following double integral of the function  $\ln(1+x+y)$  in the closed region delimited by the given constraints



Integration function

$$\ln(1+x+y)$$

Integration domain D

$$0 \leq x \leq 1$$

$$x^2 \leq y \leq x+1$$

The domain D is shown in the above plot. As we can see, it is a domain normal to the x-axis

Verify that the given integral approximates the symbolic expression at the right

$$\iint_{D(x,y)} f(x,y) ds = \int_0^1 \int_{x^2}^{x+1} \ln(1+x+y) dy dx \quad \frac{-9\ln(3)}{4} + 7\ln(2) - \frac{\pi\sqrt{3}}{12} - \frac{17}{8}$$

The macro assumes as default the following simple arrangement (but, of course, it is not obligatory at all)

	A2		f(x,y)			
	A	B	C	D	E	
1	f(x,y)	a	b	c	d	
2	ln(1+x+y)	0	1	x^2	x+1	
3						
4						
5						

Select the A2 cell and start the **Integr2D** macro.

select the integration function



As we can see, the entire input fields are filled with the right cell references.

The output result will start from the A4 cell

The macro outputs 5 results:

- 1) Integral
- 2) Relative error estimation
- 3) Total points evaluated
- 4) Elaboration time
- 5) Error message

Optional we can adjust the Error limit or the Rank. But usually the only thing to do is clicking on the "run" button

Note: the computation effort increases exponentially with the rank, because is:

Total points =  $4^K$ .

The results will appear as the following

	A	B	C	D	E
1	$f(x,y)$	a	b	c	d
2	$\ln(1+x+y)$	0	1	$x^2$	$x+1$
3					
4	Integral	Err. rel.	Points	Time	
5	0.98225833	7E-13	4225	0.0547	

## Double integration function

**=Integr\_2D (Fxy, a, b, c, d, [Polar],[ErrMax])**

This function returns the numeric integral of a smooth regular function  $f(x, y)$  over a plane normal domain  $D(x, y)$ .

$$\int_a^b \int_c^d f(x, y) dx dy$$

The integration functions  $f(x, y)$  and – eventually – also the bounding limits – a, b, c, d – can be written in symbolic expression

The integration function can be:

- bivariate functions like  $x^2+y^2-x*y$ ,  $\log(1+x+y)$ ,  $1/(1+x^2+2*y^2)$ , etc.
- constant numbers like 0, 2, 1.5, 1E-6, etc.
- constant expressions like  $1/2$ ,  $\sqrt{2+1}$ ,  $\sin(0.1)$ , etc.

The boundary limits can be:

- constant numbers like 0, 2, 10, 3.141, etc.
- constant expressions like  $1/2$ ,  $\sqrt{2+1}$ ,  $\pi$ ,  $\sin(1/2*\pi)$ ,  $\exp(1)$ , etc.
- monovariate functions like  $x/2$ ,  $3y-10$ ,  $x^2+x-1$ , etc.

A normal domain has, at least, two constant boundary limits.

## Xnumbers Tutorial

Example. Approximate the following double integral

$$\int_0^1 \int_{x^2}^{x+1} \ln(x+y+1) dy dx \quad \frac{-9\ln(3)}{4} + 7\ln(2) - \frac{\pi\sqrt{3}}{12} - \frac{17}{8}$$

The integration domain is shown in the previous example.  
The computing can be arranged as the following

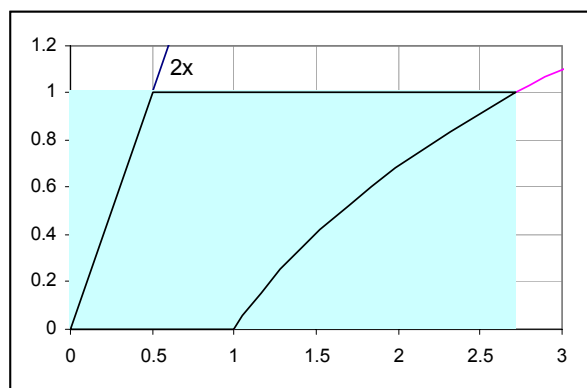
F2		fx =Integr_2D(A2;B2;C2;D2;E2)				
	A	B	C	D	E	F
1	f(x,y)	a	b	c	d	integral
2	ln(1+x+y)	0	1	x^2	x+1	0.982258329
3						
4		=Integr_2D(A2;B2;C2;D2;E2)				

In order to avoid long elaboration time, the function limits the total evaluation points to about 65.000 (rank = 8). For heavy computations use the macro **Integr2D**  
Tip: this function can also return the relative error, the total of evaluation points and the error message (if any). To see these values simply select a range of two, three or four, adjacent cells (vertical or horizontal) and give the CTRL+SHIFT+ENTER key sequence.

Example: Approximate the following integral

$$\int_0^1 \int_{y/2}^{e^y} \frac{1}{x^2 + y^2 + 1} dx dy$$

The integration domain is represented in the following plot



Integration domain D

$$0 \leq y \leq 1$$

$$\frac{y}{2} \leq x \leq e^y$$

As we can see the domain is normal to the y-axes

The computation of this double integral can be arranged as the following

	A	B	C	D	E	F	G
1	F(x,y)	a	b	c	d		
2	1/(1+x^2+y^2)	y/2	exp(y)	0	1		
3							
4	Integral	Rel.err	Points	{=Integr_2D(A2;B2;C2;D2;E2)}			
5	0.671420437	1.4E-14	16641				
6							

## Infinite integral

---

### Integral\_Inf()

This macro performs the numeric integration of a smooth, regular, not oscillating function  $f(x)$  over an unlimited (or very long) interval

$$\int_a^{+\infty} f(x)dx \quad , \quad \int_{-\infty}^a f(x)dx \quad , \quad \int_{-\infty}^{+\infty} f(x)dx$$

This macro can use two different methods:

- The Bode formula with adaptive step
- The double exponential algorithm

The Bode formula with 8 steps to calculate the integral and the truncation error.

$$I_{h1} = \frac{2h}{45}(7f_0 + 32f_1 + 12f_2 + 32f_3 + 7f_4)$$

$$I_{h2} = \frac{2h}{45}(7f_4 + 32f_5 + 12f_6 + 32f_7 + 7f_8)$$

$$I_h = I_{h1} + I_{h2}$$

$$E_T \approx \frac{I_h - I_{h2}}{63}$$

After each step the routine detects the truncation error and recalculates the step in order to keep a constant error (variable step integration method).

The double exponential algorithm, also called "*tanh-sinh quadrature*", first introduced by Takahasi and Mori, is based on the hyperbolic variable transformations.

$$x = \tanh(\sinh(t)) \qquad dx = \frac{\cosh(t)}{2\cosh^2(\sinh(t))} dt$$

It is more complicated than the polynomial Newton-Cotes schema but, on the other hand, it is much more efficient.

Using this macro is very easy.

Example: Approximate the given integral

$$\int_0^{+\infty} 100 \cdot x^2 \cdot e^{-x} dx$$

The integration function is regular over the entire x-axes; the exponential assures the convergence. Thus the infinite integral exists.

Put the symbolic expression "100\*x^2\*exp(-x)" in any cell that you like (A3 for example), and arrange the worksheet in the following way (but it is not obligatory at all)

## Xnumbers Tutorial

	A	B	C	D
1				
2	<b>Integr. function f(x)</b>	<b>a</b>	<b>b</b>	
3	100*x*exp(-x)	0	inf	
4				
5				
6				

The word "inf" means – of course – infinity.

It is not necessary to specify the sign, because the macro always assumes

"b" as +inf

"a" as -inf

Now select the cell A3 and run the macro **Integral\_Inf** . The input fields will be automatically filled

Choose "run" to start the integration routine. The result will be similar to the worksheet below (without formatting) where we have compare the result of both methods

	A	B	C	D	E	F
1						
2	<b>Integration function f(x)</b>	<b>a</b>	<b>b</b>			
3	100*x*exp(-x)	0	inf			
4						
5	<b>Integral</b>	<b>Err. rel.</b>	<b>Points</b>	<b>Time</b>		
6	100	1.6E-13	199	0.015625	(double exponential)	
7	100	6.57E-18	2048	0.039063	(variable step)	

As we can see the integral is 200 with excellent approximation for both methods but the double exponential is more efficient. It required only 199 function evaluations.

Sometime we have to calculate the integral over the entire x- axes. Let' see

$$\int_{-\infty}^{+\infty} \frac{x^2 + 2x + 3}{x^4 + x + 1} dx$$

## Xnumbers Tutorial

	A	B	C	D	E	F
1						
2	Integration function f(x)	a	b			
3	$(x^2+2x+3)/(x^4+x+1)$	-inf	inf			
4						
5	Integral	Err. rel.	Points	Time		
6	9.105282814	7.24E-15	344	0.007813	(double exponential)	
7	9.105282814	9.09E-17	10468	0.320313	(variable step)	
8						

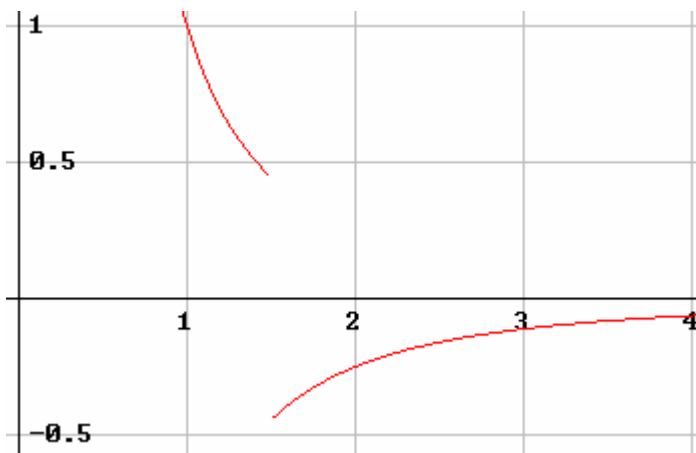
Note that in this case we have needed more than 10.000 evaluation point for the variable step method but only 344 for the DE method. The superiority is ever so evident? Not ever. There are cases in which the adaptive quadrature schema works better. For example when the integration function has a finite discontinuity (jump) inside the integration interval; this usually happens for the piecewise functions.

Example, Assume to have to compute the following didactical integral

$$\int_1^{+\infty} \operatorname{sgn}(1.5-x) \frac{1}{x^2} dx$$

The integration function is sketched in the following graph

In this case is easy to calculate the integral simply separating the given interval  $[1, +\infty]$  in two sub-intervals:  $[1, 1.5] \cup [1.5, +\infty]$ .  
Calculating each integral and summing we get the exact result  $I = -1/3$ .



But we want here to investigate how the two methods works in this situation

	A	B	C	D	E	F
1						
2	Integration function f(x)	a	b			
3	$\operatorname{sgn}(1.5-x)/x^2$	1	inf			
4						
5						
6	Convergence error					
7	Integral	Err. rel.	Points	Time		
8	-0.333333333	-5.3E-15	7208	0.148438	(variable step)	
9	Convergence error				(double exponent)	
10						

As we can see the variable step method has find the result with high precision using about 7200 steps. The double exponential algorithm even fails the convergence

We have to put in evidence that using this macro in a “blind” way, can lead to wrong result. We should always study the integration function to discover singularities, discontinuities, convergence rate, etc. If the integration function is “sufficiently” smooth, then the numeric integration can give good approximate results.

This routine can also be used over a closed long interval, when other algorithms would take too long computational time.

## Series Evaluation

**=xSerie(Funct, Id, Id\_min, Id\_max, [Param], [DgtMax])**

Returns the numeric series of a function  $f(n)$ .

$$s = \sum_{n=\min}^{\max} f(n)$$

The parameter **Funct** is a math expression string such as:

"2^n/n\*(-1)^(n+1)", "x^n/n!", "(-1)^(n)\*(3+a)\*x/(n-1)", ...

Remember the quote "" for passing a string to an Excel function.

For further details about the math string see [Math formula string](#)

**Id** indicates the integer index of the sum (usually "n", "k", "i", etc.)

**Id\_min** and **Id\_max** indicate the range of the index.

The function may have also other parameters ("x", "y", "a", etc.) that can assume real values.

**Param** contains labels and values for parameters substitution (if there are). If we pass the variable range without "labels", the function will assign the values to the variables in the same order that they appear in the formula string, from left to right.

The parameter **DgtMax** sets the multiprecision arithmetic. if omitted or zero the function uses the fastest standard arithmetic

Example 1.

`xSerie("x^n/n*(-1)^(n+1)", "n", 1, 10, 2)`

The function substitutes  $x = 2$  and then, computes the series  $f(n)$  for  $n = 1, 2, 3, \dots, 10$

$$\sum_{n=1}^{10} \frac{2^n}{n \cdot (-1)^{n+1}} = 2^1 - \frac{2^2}{2} + \frac{2^3}{3} - \frac{2^4}{4} + \frac{2^5}{5} - \dots - \frac{2^{10}}{10}$$

Example 2. Compute

$$s = \sum_{n=0}^{10} \frac{x^n}{n!}$$

for  $x = -1.5$ , with standard precision (15 digits) and with 25 digits. As known, this series approximates the exponential  $e^{(-1.5)}$

	A	B	C	D	E
1	<b>F(x, n)</b>	<b>where</b>	<b>from</b>	<b>to</b>	<b>for x =</b>
2	$x^n/n!$	n	0	10	-1.5
3	$\Sigma$				
4	0.223132084437779	<code>=xSerie(A2,B2,C2,D2,E2)</code>			
5					
6	0.2231320844377790178571427	<code>=xSerie(A2,B2,C2,D2,E2,25)</code>			
7					

## Xnumbers Tutorial

The function **xSerie** accepts one or more parameters.

Example 3. Compute the following series

$$s = \sum_{n=1}^{10} \frac{b \cdot n + a}{n}$$

for  $a = 0.7$  and  $b = 1.5$ , in standard precision

	A	B	C	D	E	F
1	<b>F(n,a,b)</b>	<b>index</b>	<b>from</b>	<b>to</b>	<b>Σ</b>	
2	$(b \cdot n + a) / n^2$	n	1	10	5.478289793	
3		<b>Parameters</b>				
4		<b>a</b>	<b>b</b>			
5		0.7	1.5			

**=xSerie(A2,B2,C2,D2,B4:C5)**

Note that we have enclosed the labels "a" and "b" in the range B4:C5 passed to the function as the argument "Param". The labels indicate to the function the correct assignment between the variables and their values

Labels are optional. If we pass only the range B5:C5, without labels, the function assign the values to the variables in the order from left to right.

Note how compact and straight is the calculation using the **xSerie** function.

## Series acceleration with $\Delta^2$

Many series are very slow to converge requiring therefore methods to accelerate their convergence. The Aitken's extrapolation formula ( $\Delta^2$  extrapolation) can be used for this scope. Practically we build a new series  $S^{(1)}$ , whose partial sums  $S_n^{(1)}$  are given by the Aitken's formula. It is possible to repeat the process starting from the series  $S^{(1)}$  to obtain  $S^{(2)}$ , and so on.

Example. We want to approximate the following series:

$$S = \sum_{k=0}^{\infty} \frac{(-1)^k}{1+k}$$

	A	B	C
1	<b>f(k)=</b>	$(-1)^k / (k+1)$	
2			
3	<b>k</b>	<b>Sk</b>	<b>Error</b>
4	0	1	0.306853
5	1	0.5	-0.193147
6	2	0.83333333	0.140186
7	3	0.58333333	-0.109814
8	4	0.78333333	0.090186
9	5	0.61666667	-0.076481
10	6	0.75952381	0.066377
11	7	0.63452381	-0.058623
12	8	0.74563492	0.052488
13	9	0.64563492	-0.047512
14	10	0.73654401	0.043397
15	11	0.65321068	-0.039937
16	12	0.73013376	0.036987
17	extrap =	0.69314719	7.31E-09

We know the exact result that is

$$\Sigma = \text{Log}(2) = 0.693147180559945...$$

In the cell B4 we have insert the formula

**=Series(\$B\$1;"k";0;A4)**

In the cell C4 we have inserted

**=(B4-LN(2))**

we fill the rows from 5 to 16 simply selecting the range B4:C4 and dragging it down.

In the last cell B17 we have inserted the function

**=ExtDelta2(B10:B16)**

performing the  $\Delta^2$  extrapolation using the last 7 values of the sum

## Xnumbers Tutorial

As we can see, the cell B16 shows the sum with 12 terms; its approximation is very poor having an error of more than 0.01. But if we apply the  $\Delta^2$  extrapolation at the last seven partial sums  $S^{(12)}, S^{(11)}, S^{(10)} \dots S^{(6)}$  we have a good approximation with an error less than  $1E-8$

Note that for reaching this accuracy the given series would need more than 100 million terms!

## Complex Series Evaluation

### =cplxserie(Formula, min, max, [z0])

This function returns the numeric series of a complex function  $f(z, n)$ .

$$S = \sum_{n=n_0}^{n_1} f(z, n)$$

Formula is a math expression string defined by arithmetic operators and common elementary functions such as:

" $2^n/n * (-1)^{(n+1)}$ ", " $x^n/n!$ ", " $(-1)^{(n)} * (3+j) * x / (n-1)$ ", ...

Remember the quote "" to pass a string to an Excel function.

The integer variable must be "n".

The parameters "min" and "max" set the minimum and the maximum limits of the integer variable "n".

The function may have also a complex variable "z". In that case specify its value in the parameter z0.

Example: evaluate the given series for  $z = z_0 = 2-i$

$$S = \sum_{n=1}^{20} \frac{z}{n} = z + \frac{z}{2} + \frac{z}{3} + \dots + \frac{z}{20}$$

E4		{=cplxserie(B2;B3;B4;E2:F2)}				
	A	B	C	D	E	F
1	<b>Complex Series</b>				re	im
2	f(z) =	z/n		z0 =	2	-1
3	n min =	1				
4	n max =	20		Σ =	7.195479	-3.59774



## Double Series

**= xSerie2D(Funct, Id1, Id1\_min, Id1\_max, Id2, Id2\_min, Id2\_max, [Param], [DgtMax])**

Returns the numeric double series of a function  $f(n, m)$ .

$$s = \sum_n \sum_m f(n, m)$$

The parameter **Funct** is a math expression string such as:

"  $x^{(n+2*m)} / (n! * m!)$  ", "  $(n+1) / (m+1) !$  ", "  $\text{comb}(n, k) * a^k * b^{(n-k)}$  " ...

Remember the quote " " for passing a string to an Excel function.

For further details about the math string see the par. [Math formula string](#)

**Id1** , **Id2** indicate the integer indexes of the sum (usually "n", "m", "k", "i", etc.)

**Id1\_min** and **Id1\_max** , **Id2\_min** and **Id2\_max** indicate the range of the correspondent index.

The function may have also other parameters ("x", "y", "a", etc.) that can assume real values.

**Param** contains labels and values for parameters substitution (if there are). If we pass the variable range without "labels", the function will assign the values to the variables in the same order that they appear in the formula string, from left to right.

The parameter **DgtMax** sets the multiprecision arithmetic. if omitted or zero the function uses the fastest standard arithmetic

Example. Compute the following double series, in standard (15 digits) and multiprecision (25 digits)

$$s = \sum_{m=0}^4 \sum_{n=1}^{10} \frac{x^{(n+2m)}}{n!m!}$$

for  $x = 0.8$

	A	B	C	D	E
1	F(x, n, m)	for x =	where	from	to
2	$x^{(n+2*m)} / (n! * m!)$	0.8	n	1	10
3			m	0	4
4	$\Sigma$				
5	2.32298975273825	=xSerie2D(A2,C2,D2,E2,C3,D3,E3,B2)			
6					
7	2.322989752738248560531618	=xSerie2D(A2,C2,D2,E2,C3,D3,E3,B2,25)			

Take care to the index limits because, for large interval, this function can slow down your Excel application

## Trigonometric series

**= Serie\_trig(t, period, spectrum, [offset], [Angle])**

It returns the trigonometric series defined by

$$f(t) = f(0) + \sum_{n=1}^N a_n \sin(n\omega t + \theta_n)$$

$$\omega = \frac{2\pi}{T}$$

The set

$$(a_n, \theta_n), n=1 \dots N$$

is called "*spectrum*" of the function  $f(t)$   
Each couple is called *harmonic*.

The parameter "t" can be a single value or a vector values

The parameter "period" is the period T.

The parameter "spectrum" is an array of (n x 2) elements: the first column contains the amplitude, the second column the phase (in "rad", "deg", or "grad" degree).

The optional parameter "offset" adds the average level (default 0)

The optional parameter "Angle" sets the angle unit: (RAD (default), DEG, GRAD)

Example:

	A	B	C	D	E	F
1	<b>Harmonic Analysis</b>				t	y(t)
2	to =	0		B2=	0	2.4242641
3	n =	16		E2+\$B\$5=	0.0625	3.0769529
4	T =	1			0.125	3.4
5	ΔT =	0.0625	=B4/(B3-1)		0.1875	3.0769529
6	y avg =	2			0.25	2.4242641
7	Angle =	DEG			0.3125	2.0131316
8					0.375	2
9	Harm.	Amp.	Phase		0.4375	1.9868684
10	1	1	45		0.5	1.5757359
11	2	0	0		0.5625	0.9230471
12	3	0.4	-45		0.625	0.6
13	4	0	0		0.6875	0.9230471
14	5	0	0		0.75	1.5757359
15	6	0	0		0.8125	1.9868684
16	7	0	0		0.875	2
17	8	0	0		0.9375	2.0131316
18						
19	{=Serie_trig(E2:E17;B4;A10:C17;B6;B7)}					

Here is a worksheet arrangement to tabulate a trigonometric serie having a spectrum of max 8 harmonics (the formulas inserted are in blue)

The independent parameters are N (samples) and T (periodo)

From those, we get the sampling interval

$$\Delta T = T/(N-1)$$

The table at the left contains the parameters for each harmonic: the integer multiple of the harmonic, its amplitude and its phase

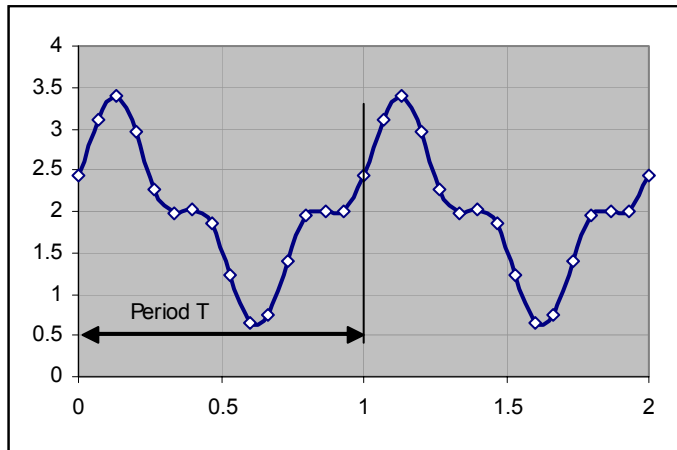
The following plot is obtained for

$$f(t) = 2 + \sin(\omega t + \pi/4) + 0.4 \sin(3\omega t - \pi/4)$$

where:

$$\omega = \frac{2\pi}{T}$$

and T = 1



$T = 1$

n° Arm.	Amp	Phase
1	1	45
2	0	0
3	0.4	-45

Note that you can always transform cosine terms into sine terms with the following formula

$$\cos(\alpha) = \sin(\alpha + \pi / 2)$$

## Trigonometric double serie

= **Serie2D\_trig(x, y, Lx, Ly, Spectrum, [offset], [Angle])**

It returns the trigonometric double serie  $f(x,y)$  defined by:

$$f(x, y) = f_0 + \sum_{n=1}^N \sum_{m=1}^M a_{n,m} \cos(n\omega_x x + m\omega_y y + \theta_{n,m})$$

where

$$\omega_x = \frac{2\pi}{L_x} \quad , \quad \omega_y = \frac{2\pi}{L_y}$$

The set

$$[a_{n,m}, \theta_{n,m}] \quad n = 0 \dots N \quad , \quad m = 0 \dots M$$

is called "spectrum" of the function  $f(t)$ . Each couple is called "harmonic".

The parameters "x" and "y" are vectors

The parameters "Lx" and "Ly" are the base lengths of the x-axis and y-axis.

The parameter "Spectrum" is an array of (n x 4) elements: containing the following information: index n, index m, amplitude and phase.

That is, for example:

n	m	Amplitude	Phase
0	1	1	45
2	1	0.5	-45
3	1	0.25	15.5
1	4	0.125	30

The optional parameter "offset" adds the average level (default 0)

The optional parameter "Angle" sets the angle unit (RAD (default), DEG, GRAD)

The function  $f(x, y)$  is returned as an (N x M) array.

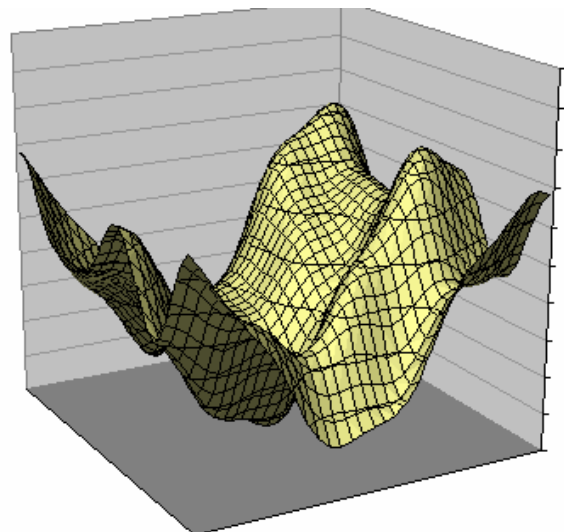
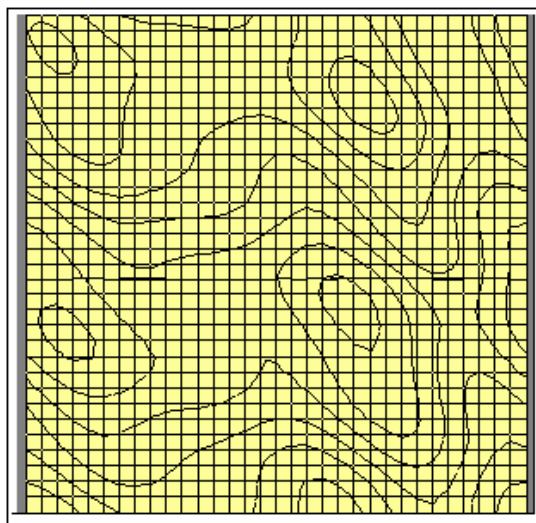
Use the CTRL+SHIFT+ENTER key to insert this function.

## Xnumbers Tutorial

Example: Here it is a worksheet arrangement to tabulate a trigonometric serie  $f(x, y)$  having a spectrum of max 4 harmonics

	A	B	C	D	E	F	G	H	I	J
1		N	L	dL		n	m	A	P	
2	x =>	16	1	0.0625		0	1	1	45	
3	y =>	16	1	0.0625		2	1	0.5	-45	
4						3	1	0.25	15.5	
5	offset	Angle				1	4	0.125	30	
6	1	DEG	=B10+\$D\$2							
7										
8										
9										
10		0	0.0625	0.125	0.1875	0.25	0.3125	0.375	0.4375	0.5
11	0	2.4098	2.3137	1.8754	1.4938	1.3579	1.3561	1.3559	1.4293	1.7115
12	0.0625	1.9791	1.6786	1.2051	0.9443	0.9664	1.0856	1.1895	1.3728	1.7101
13	0.125	1.5149	1.1235	0.7267	0.6323	0.7801	0.9426	1.0539	1.2293	1.4851
14	0.1875	1.1722	0.6902	0.3497	0.3461	0.5118	0.6252	0.6871	0.822	0.9863
15	0.25	0.6879	0.1209	-0.151	-0.07	0.1177	0.2242	0.3081	0.472	0.605
16	0.3125	0.051	-0.455	-0.537	-0.292	-0.026	0.1246	0.2576	0.4313	0.4839
17	0.375	-0.326	-0.627	-0.466	-0.089	0.1856	0.2988	0.3798	0.447	0.3258

Here is the contour plot and the 3D plot of the function  $f(x, y)$



## Discrete Convolution

### Convol(f, g, h)

This function approximates the convolution of two sampled functions  $f(t)$ ,  $g(t)$

$$f * g = \int_{-\infty}^{+\infty} f(v)g(t-v)dv$$

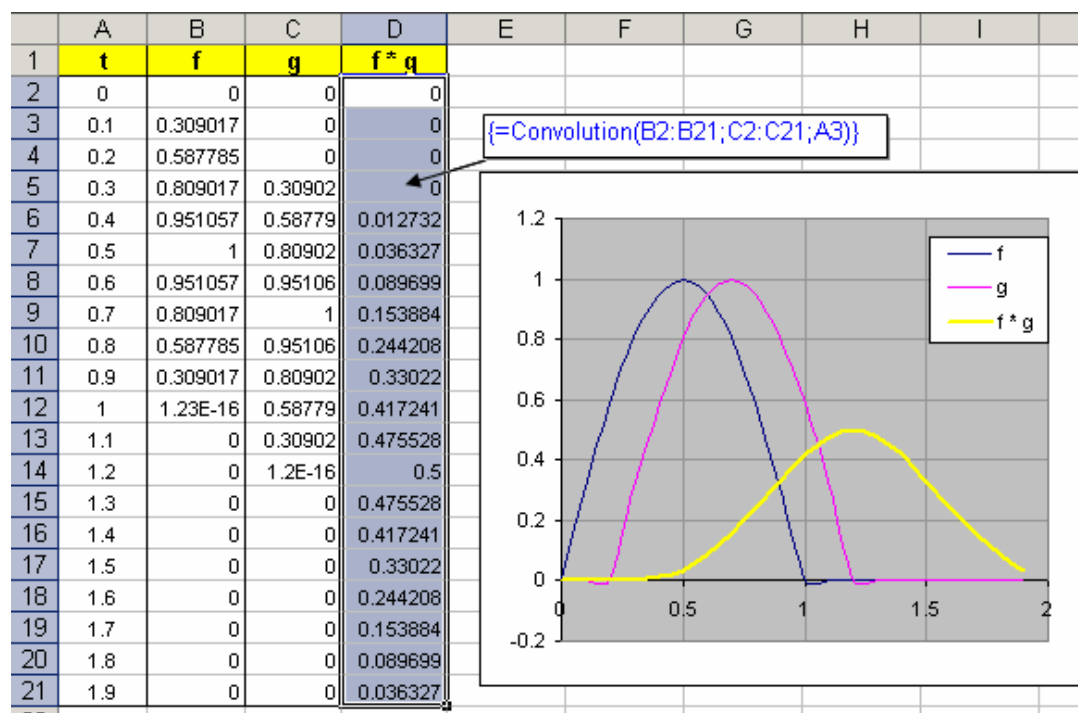
The parameters "f" and "g" are column-vectors

The parameter "h" is the sampling step (also called  $\Delta t$ )

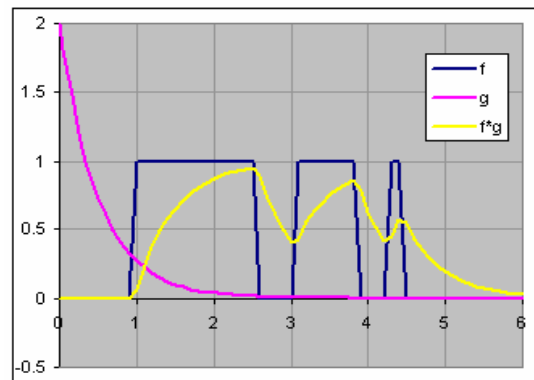
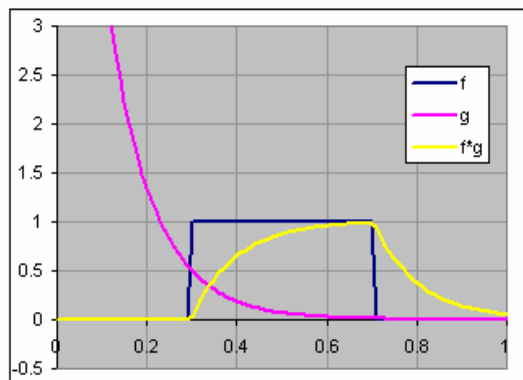
Returns a vector with the same dimension of the two vectors f and g.

The convolutions is also called "Faltung"

Example

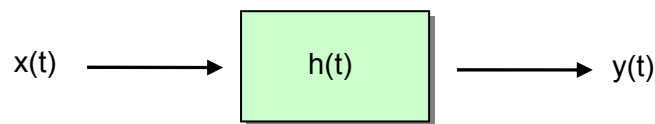


Here are other examples of convolution plots



In the signals analysis the function  $f$  is called "input signal"  $x(t)$  and  $g$  is called "system impulse response"  $h(t)$ . The convolution  $f*g$  is called "system response"  $y(t)$

The system behavior is reassumed in the following schema



In a linear system, the outputs signal  $y(t)$  depends by the input signal  $x(t)$  and by the impulse response of the system  $h(t)$ . That is:

$$y(t) = \int x(v) h(t-v) dv$$

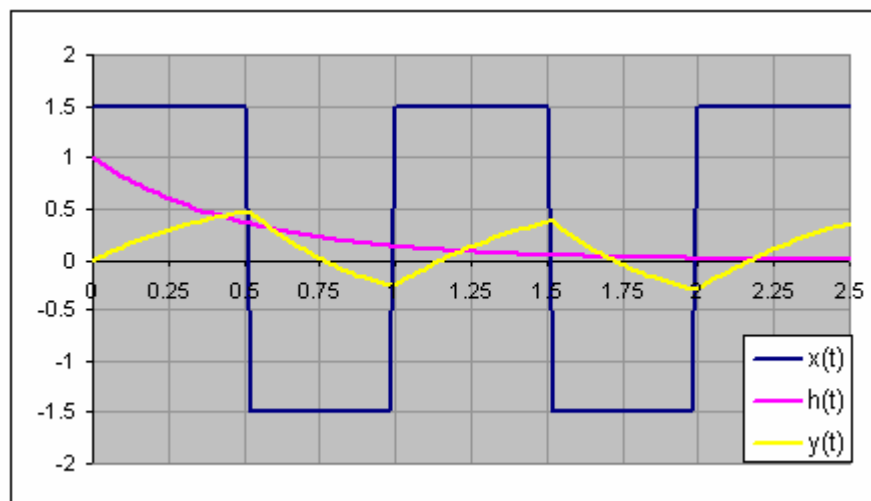
If the system is described by the following differential equation

$$y'(t) + k \cdot y(t) = x(t)$$

which has an impulse response given by

$$h(t) = e^{-k \cdot t}$$

We will use convolution to find the zero input response of this system to the square signal of period  $T = 1$  and amplitude  $x_{\max} = 1.5$



For obtaining this graph we have used a sampling step of  $\Delta t = 0.02$ , but this value is not critical at all. You can choose the size that you like in order to obtain the needed accuracy.

## Interpolation

### Interpolation with continue fraction

**Fract\_Interp\_Coef**(xi, yi)

**Fract\_Interp**(x, xi, coeff)

**xFract\_Interp\_Coef**(xi, yi, [Digit\_Max])

**xFract\_Interp**(x, xi, coeff, [Digit\_Max])

These functions perform the interpolation with the continue fraction method.  
Given, for example, a set of 5 points

$$xi = [x_0, x_1, x_2, x_3, x_4], yi = [y_0, y_1, y_2, y_3, y_4]$$

the function **Fract\_Interp\_Coef** returns the coefficients vector  $[a_0, a_1, a_2, a_3, a_4]$  of the continue fraction expansion given by the following formula:

$$y \cong a_0 + \frac{x - x_0}{a_1 + \frac{x - x_1}{a_2 + \frac{x - x_2}{a_3 + \frac{x - x_3}{a_4}}}}$$

The function **Fract\_Interp** returns the interpolate value  $y$  at the point  $x$   
For multiprecision computing use the function **xFract\_Interp** and **xFract\_Interp\_Coef**

Example: find the continue fraction interpolation coefficients for the following 10 samples

n	x	y samples
0	0.5	-3.461538462
1	0.6	-4.37037037
2	0.7	-6.073170732
3	0.8	-10.15384615
4	0.9	-31.22222222
5	1	30
6	1.1	10.35483871
7	1.2	6.37037037
8	1.3	4.670886076
9	1.4	3.735849057

	A	B	C	D
1				
2	n	x	y samples	coefficients
3	0	0.5	-3.461538462	-3.461538462
4	1	0.6	-4.37037037	-0.110031348
5	2	0.7	-6.073170732	2.989457243
6	3	0.8	-10.15384615	1.284514107
7	4	0.9	-31.22222222	1.472081218
8	5	1	30	-6.70179E+11
9	6	1.1	10.35483871	-4.0173E-13
10	7	1.2	6.37037037	-3.33594E+12
11	8	1.3	4.670886076	1.02373E+13
12	9	1.4	3.735849057	0
13				
14				
15				

=Fract\_Interp\_Coef(B3:B12;C3:C12)

These points are extracted from the following function

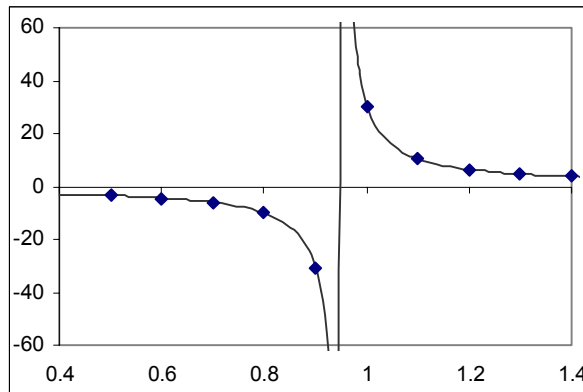
$$y = \frac{x^2 + 2}{x^2 - 0.9}$$

## Xnumbers Tutorial

You can verify that the interpolation with these coefficients are better than  $1E-14$  for all  $x$ -values in the range  $[0.4 - 1.6]$

Note also that this great precision is reached in spite of the pole at  $x \cong 0.95$

The continue fraction interpolation is adapt just to interpolate rational functions



Interpolation with continued fraction

The blue dot are the given knots.

The light black line is the interpolation obtained

There is a pole at  $x \cong 0.95$

Example: Find an interpolation formula for the function  $\tan(x)$  in the range  $0 \leq x \leq 1.5$  with no more than 7 points.

The function  $\tan(x)$  has a pole at  $x = 1.57\dots$ , closed to the upper bound 1.5; so its presence suggest to adopt a fraction interpolation. Assume to take samples of the function  $\tan(x)$  at the values (0, 0.2, 0.6, 1, 1.25, 1.45, 1.5).

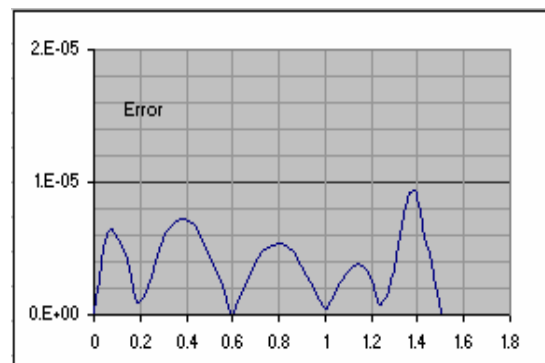
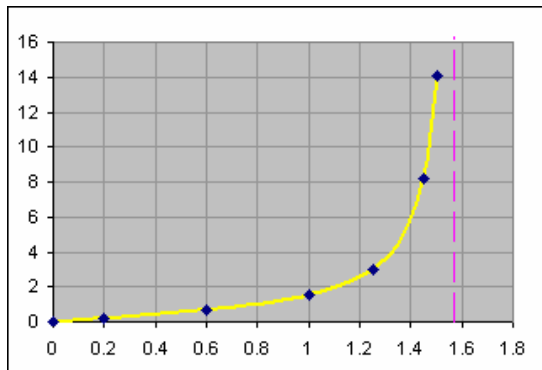
	A	B	C
1	x	tan(x)	interp coeff
2	0	0	0
3	0.2	0.202710	0.986631
4	0.6	0.684137	-3.649189
5	1	1.557408	0.301377
6	1.25	3.009570	4.348352
7	1.45	8.238093	3.175407
8	1.5	14.101420	-1.566518
9	{=Fract_Interp_Coef(A2:A8;B2:B8)}		
10			

The column A contains the knots of the interpolations

In column B we have inserted the correspondent values of  $\tan(x)$

And in column C we have computed the coefficients of the fraction interpolation.

Now using the function **Fract\_Interp** we can interpolate any value between 0 and 1.5 obtaining the graph to the left. The second graph shows the absolute error in the given range. You can verify that the interpolation is better than  $1E-5$  for any value  $x$ .





## Interpolation with Cubic Spline

**cspline\_interp(Xin , Yin , Xtarget )**

**cspline\_eval(Xin, Yin, Ypp, Xtarget)**

These functions<sup>15</sup> perform the natural cubic spline interpolation

Xin is the vector containing the x-values.

Yin is the vector containing the y-values..

Xtarget is the x value which we want to compute the interpolation

Xpp is the vector containing the 2<sup>nd</sup> derivative

The cubic spline interpolation is based on fitting cubic polynomial curves through all the given set of points, called knots

The cubic spline follows these rules:

- the curves pass through all the knots
- at each knot, the first and second derivatives of the two curves that meet there are equal
- at the first and last knot, the second derivatives of each curve is equal to 0 (natural cubic spline constrain).

The natural cubic spline has a continuous second derivative (acceleration). This characteristic is very important in many applied sciences (Numeric Control, Automation, etc...) when we need to reduce vibration and noise in electromechanical motions, although cubic spline is much slower than other interpolation methods.

The function cspline\_eval is faster than cspline\_interp, because the first uses the information of the 2nd derivatives and does not have to calculate them all over again like the cspline\_interp does.

The 2nd derivatives can be computed by the function cspline\_pre (see next page)

Example:

	A	B	C	D	E
1	Original Data		Interpolated Data		
2	<b>Xi</b>	<b>Yi</b>	<b>X</b>	<b>Y Interp</b>	
3	0	0	0	0,0000	
4	1	2	0,1	0,1801	
5	2,5	4	0,2	0,3613	
6	3	3	0,3	0,5450	
7	4	4	.....	.....	
8	5	1	.....	.....	
9					
10	=cspline_interp(\$A\$3:\$A\$8,\$B\$3:\$B\$8,A3)				
11					

<sup>15</sup> These functions appear thanks to the courtesy of Olgierd Zieba

## Cubic Spline 2nd derivatives

### cspline\_pre(Xin , Yin)

This function<sup>16</sup> Returns the cubic spline 2<sup>nd</sup> derivatives at a given set of points (knots). Xin is the vector containing the x-values.

Yin is the vector containing the y-values..

For n knots, it returns an array of n 2<sup>nd</sup> derivative values. The first and the last values are zero (natural spline constrain).

The 2<sup>nd</sup> derivatives depend only by the given set of knots. So this function can be evaluate only once for the whole range of the interpolation. By cspline\_eval function we can compute fast interpolation

Example. Perform the sub-tabulation with  $\Delta x = 0.1$  of the following table

	A	B	C	D	E	F
1					<b>sub-tabulation</b>	
2	<b>x</b>	<b>knots</b>	<b>y''</b>		<b>x</b>	<b>y interp</b>
3	-2	0.02999	0		-2	0.02999
4	-1.5	0.00003	0.11805		-1.9	0.02305
5	-1	0.08522	2.29168		-1.8	0.01635
6	-0.6	0.46400	1.18838		-1.7	0.01012
7	-0.3	0.83296	-2.94331		-1.6	0.00460
8	0.2	0.92262	-3.90098		-1.5	0.00003
9	0.4	0.71970	-1.15153		-1.4	-0.00269
10	0.8	0.23561	2.47036		-1.3	0.00013
11	1.2	0.01724	1.23449		-1.2	0.01282
12	1.6	0.00000	0.13416		-1.1	0.03974
13	2	0.02999	0		-1	0.08522
14					-0.9	0.15244
15	{=cspline_pre(A3:A13;B3:B13)}				-0.8	0.23981
16					-0.7	0.34459

The given table is in the range A3:B13

In the adjacent column C we have computed the 2<sup>nd</sup> derivatives by the function cspline\_pre.

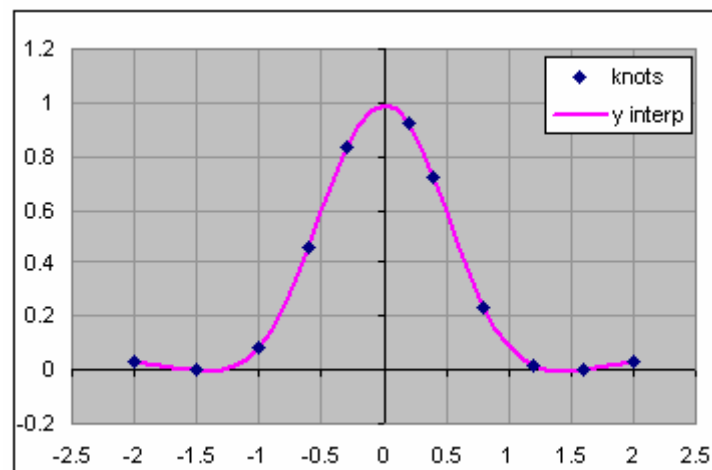
Note that this function returns a vector of 11 values. It must be inserted with the ctrl+shift+enter keys sequence

At the right we have set the new table with step 0.1; the value of F3 has been interpolate by the formula

= cspline\_eval(\$A\$3:\$A\$13; \$B\$3:\$B\$13; \$C\$3:\$C\$13; E3)

The other values are computed simply by dragging down the cell F3.

The following figure shows the knots and the cubic spline fit



The points of the original table was extracted from the function  $y = [\cos(x)]^4$ . You can verify that the interpolation accuracy is better than 1% over the entire range.

<sup>16</sup> These functions appear thanks to the courtesy of Olgierd Zieba

## Cubic Spline Coefficients

### cspline\_coeff(Xin , Yin)

This function<sup>17</sup> returns the coefficients of the cubic spline polynomials

Xin is the vector containing the x-values.

Yin is the vector containing the y-values..

It returns an (n-1 x 4 ) array where n is the number of knots. Each row contains the coefficients of the cubic polynomial of each segment s  $[a_{s,3} \ a_{s,2} \ a_{s,1} \ a_{s,0}]$

$$y_s = a_{s,3}(x - x_s)^3 + a_{s,2}(x - x_s)^2 + a_{s,1}(x - x_s) + a_{s,0}$$

where s = 1, 2, (n-1)

Example. Find the cubic spline polynomials that fit the given knots

	A	B	C	D	E	F	G
1	<b>Knots</b>			<b>Cubic Spline coefficients</b>			
2	<b>X</b>	<b>Y</b>		<b>a3</b>	<b>a2</b>	<b>a1</b>	<b>a0</b>
3	0	0	1st spline =	0.20149	0	1.79851	0
4	1	2	2nd spline =	-0.8784	0.60447	2.40298	2
5	2.5	4	3rd spline =	5.54709	-3.3482	-1.7127	4
6	3	3	4th spline =	-3.0718	4.9724	-0.9006	3
7	4	4	5th spline =	1.41437	-4.2431	-0.1713	4
8	5	1					
9				$p(x) = a3*x^3 + a2*x^2 + a1*x + a0$			
10							
11				{=cspline_coeff(A3:A8;B3:B8)}			
12							

<sup>17</sup> These functions appear thanks to the courtesy of Olgierd Zieba

## Multi-variables Interpolation

**=InterpL(Point, Knots, Funct)**

**=InterpL\_Coef(Point, Knots, Funct)**

These functions perform the linear multivariate interpolation of a function

$$y = f(x_1, x_2 \dots x_n)$$

Point = an (n) vector containing the point that you want to interpolate

Knots = an (m x n) array containing m knots of the interpolation

Funct = a (m) vector containing the m function values at the given knots

Given a vector  $(x_1, x_2, \dots x_n)$  the linear interpolation formula is

$$\hat{y} = a_0 + a_1x_1 + a_2x_2 + \dots + a_nx_n$$

The first function returns the  $\hat{y}$  value while the second returns the coefficients vector

Example.

Interpolate the function  $f(x,y)$  at the point (6.5 , 3.2). Note that the knots in the given table are neither equidistant, nor sorted (random sampling)

	A	B	C	D	E	F
1	x	y	f(x,y)		a	b
2	3.75	9	232.25		6.5	3.2
3	1.5	8	164.5			
4	4.5	7	175.5		f(a,b) =	76.3
5	5.25	10	288.75			
6	3	6	119		{=InterpL(E2:F2,A2:B13,C2:C13)}	
7	6	8	232			
8	1.5	6	96.5		coefficients	
9	2.25	5	73.75		a <sub>0</sub> =>	-130
10	6	11	334		a <sub>1</sub> =>	15
11	4.5	5	107.5		a <sub>2</sub> =>	34
12	6	6	164			
13	7.5	10	322.5			
14					{=InterpL_Coef(E2:F2,A2:B13,C2:C13)}	
15						

The interpolate value is  $f(6.5, 3.2) = 76.3$ , given by the linear formula

$$f(x, y) = 15x + 34y - 130$$

Both **InterpL** and **InterpL\_Coef** can also work in 3D and more dimensions.

## 2D Interpolation

### =Interp\_Mesh(TableXY)

This function performs the linear interpolation of a bivariate function given in a pivot table XY.

The x-values and y-values of the table must be sorted but not necessarily equidistant. This function returns an array. Let's see how it works.

A8		= {=Interp_Mesh(A1:G6)}					
	A	B	C	D	E	F	G
1		0	2	2.9	3.5	4.2	5
2	0	100	50	27.5	12.5	-5	-25
3	1.5	115	65	42.5	27.5	10	-10
4	2	120	70	47.5	32.5	15	-5
5	3.1	131	81	58.5	43.5	26	6
6	4	140	90	67.5	52.5	35	15
7							
8		0	1	2	3	4	5
9	0	100	75	50	25	0	-25
10	1	110	85	60	35	10	-15
11	2	120	95	70	45	20	-5
12	3	130	105	80	55	30	5
13	4	140	115	90	65	40	15
14							
15							
16							

### Regularization

As we can see, the use of this function is straight. Simply select the area you want to insert the new table and pass the old table as parameter.

Note that both axes are not regular

The function Interp\_mesh returns the equidistant-linear-interpolated array. Or, in other words, it returns the regularized table

	A	B	C	D	E	F	G	H	I
1		0	0.2	0.4	0.6				
2	0	10	10.2	10.4	10.6				
3	0.5	11	11.2	11.4	11.6				
4	1	12	12.2	12.4	12.6				
5	1.5	13	13.2	13.4	13.6				
6	2	14	14.2	14.4	14.6				
7									
8		0	0.1	0.2	0.3	0.4	0.5	0.6	
9	0	10	10.1	10.2	10.3	10.4	10.5	10.6	
10	0.25	10.5	10.6	10.7	10.8	10.9	11	11.1	
11	0.5	11	11.1	11.2	11.3	11.4	11.5	11.6	
12	0.75	11.5	11.6	11.7	11.8	11.9	12	12.1	
13	1	12	12.1	12.2	12.3	12.4	12.5	12.6	
14	1.25	12.5	12.6	12.7	12.8	12.9	13	13.1	
15	1.5	13	13.1	13.2	13.3	13.4	13.5	13.6	
16	1.75	13.5	13.6	13.7	13.8	13.9	14	14.1	
17	2	14	14.1	14.2	14.3	14.4	14.5	14.6	

### Rescaling

We can obtain a sub-tabulated function in a very fast way

Simply select a larger area  
The function Interp\_mesh counts the cells that you have selected and fill all the cells with the linear interpolated values

In this case the given table has 5 x 4 = 20 values.  
The new table has 9 x 7 = 63 values; therefore, there are 43 new interpolated values

### Interpolation of Tabulated data function

Given a tabulated data  $(x_i, y_i)$ ,  $i = 1 \dots N$ , generally not equidistant, the task is estimating  $y$  for an arbitrary  $x$  value, where  $x_1 \leq x \leq x_N$

The points  $(x_i, y_i)$  are called **knots** of the interpolation

### Cubic Spline interpolation

The goal of cubic spline interpolation is to get a polynomial interpolation formula that is smooth in the 1<sup>st</sup> derivative, and continuous in the 2<sup>nd</sup> derivative, within the interval and at each boundaries.

This method ensures that the functions  $y(x)$ ,  $y'(x)$ , and  $y''(x)$  are equal at the interior node points for adjacent segments. The cubic polynomials  $P_i(x)$  satisfy these constraints.

$$\begin{aligned} P_i(x_{i-1}) &= y_{i-1} & \text{for } i &= 2 \dots N \\ P_i(x_i) &= y_i & \text{for } i &= 2 \dots N \\ P'_i(x_i) &= P'_{i+1}(x_i) & \text{for } i &= 2 \dots N-1 \\ P''_i(x_i) &= P''_{i+1}(x_i) & \text{for } i &= 2 \dots N-1 \end{aligned}$$

### Formulas

One form to write the interpolation polynomials is:

$$P(x) = A P_i + B P_{i+1} + C P''_i + D P''_{i+1}, \quad \text{for } i = 1 \dots (N-1)$$

Where:

$$\begin{aligned} A &= (x_{i+1} - x) / (x_{i+1} - x_i) \\ B &= 1 - A \\ C &= 1/6 (A^3 - A) (x_{i+1} - x_i)^2 \\ D &= 1/6 (B^3 - B) (x_{i+1} - x_i)^2 \end{aligned}$$

The 2<sup>nd</sup> derivatives can be evaluated by the following linear equations

$$(x_i - x_{i-1}) P''_{i-1} + 2(x_{i+1} - x_{i-1}) P''_i + (x_{i+1} - x_i) P''_{i+1} = H_i \quad \text{for } i = 2 \dots (N-1)$$

where:

$$\begin{aligned} H_i &= 6[(P_{i+1} - P_i)/(x_{i+1} - x_i) - (P_i - P_{i-1})/(x_i - x_{i-1})] \\ P''_1 &= 0 \\ P''_N &= 0 \end{aligned}$$

That gives the following tridiagonal matrix system

$2(x_3 - x_1)$	$(x_3 - x_2)$	0	0	...	0
$(x_3 - x_2)$	$2(x_4 - x_2)$	$(x_4 - x_3)$	0	...	0
0	$(x_4 - x_3)$	$2(x_5 - x_3)$	$(x_5 - x_4)$	...	0
0	0	$(x_5 - x_4)$	$2(x_6 - x_4)$	...	...
...	...	...	...	...	$(x_N - x_{N-1})$
0	0	0	...	$(x_N - x_{N-1})$	$2(x_N - x_{N-2})$

$=$ 

$P''_2$
$P''_3$
$P''_4$
$P''_5$
...
$P''_{N-1}$

$H_2$
$H_3$
$H_4$
$H_5$
$H_{N-1}$

Another common way to write the interpolation polynomial is:

## Xnumbers Tutorial

$$P(x) = a_{3i} (x - x_i)^3 + a_{2i} (x - x_i)^2 + a_{1i} (x - x_i) + a_{0i} \quad , \quad x_i \leq x < x_{i+1}$$

for  $i = 1 \dots (N-1)$

Where the coefficients are:

$$a_{3i} = (P''_{i+1} - P''_i) / (x_{i+1} - x_i) / 6$$

$$a_{2i} = P''_i / 2$$

$$a_{1i} = (P_{i+1} - P_i) / (x_{i+1} - x_i) - (x_{i+1} - x_i) (2 P''_i + P''_{i+1}) / 6$$

$$a_{0i} = P_i$$

The matrix of the system is tridiagonal, therefore can be solved in  $O(N)$  operations. We note also that its solution  $(P''_1, P''_2, \dots, P''_N)$  depends only by the given knots, therefore the 2<sup>nd</sup> derivatives can be evaluated only once for any interpolate. This example shows very well how the interpolation spline works.

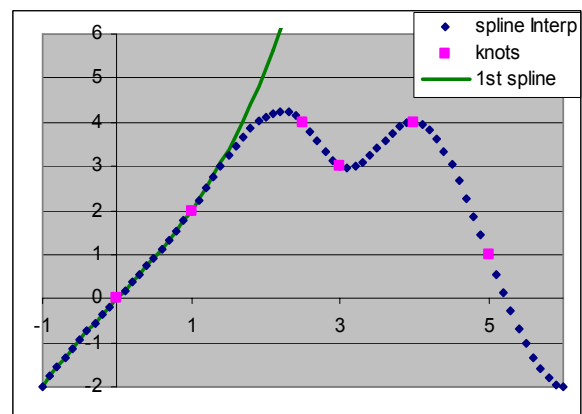
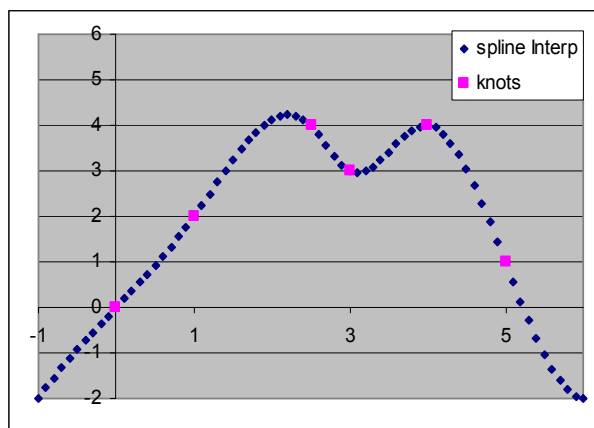
X	Y
0	0
1	2
2.5	4
3	3
4	4
5	1

Assuming to have to sub-tabulate with a step  $\Delta x = 0.1$  a given function known only in the following 6 points  
Note that these points are unequal spaced

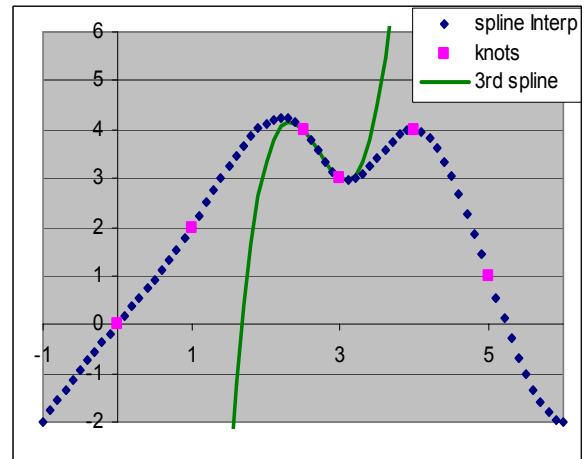
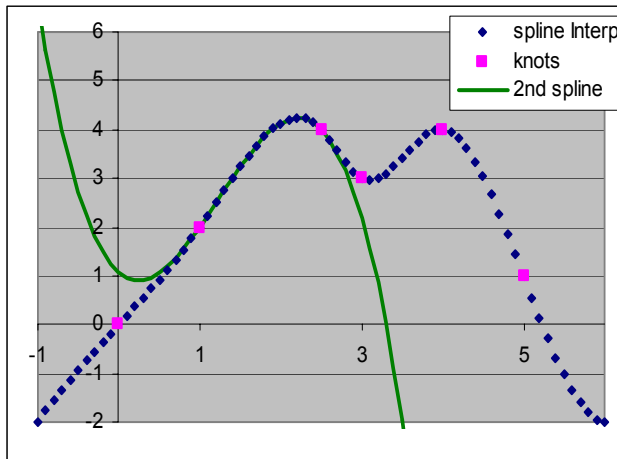
For these 6 knots we obtain 5 cubic polynomials having the following coefficients

Polynomials	a3	a2	a1	a0	Range
1st spline	0.20148927	0	1.79851073	0	$0 \leq x < 1$
2nd spline	-0.8783764	0.60446781	2.40297854	2	$1 \leq x < 2.5$
3rd spline	5.54708717	-3.348226	-1.7126588	4	$2.5 \leq x < 3$
4th spline	-3.0718353	4.97240473	-0.9005694	3	$3 \leq x < 4$
5th spline	1.41436706	-4.2431012	-0.1712659	4	$4 \leq x < 5$

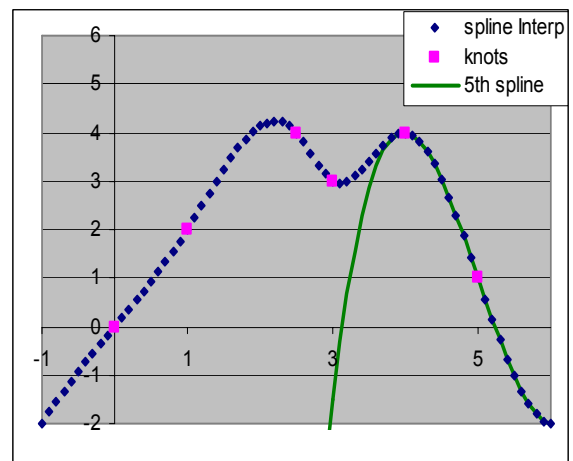
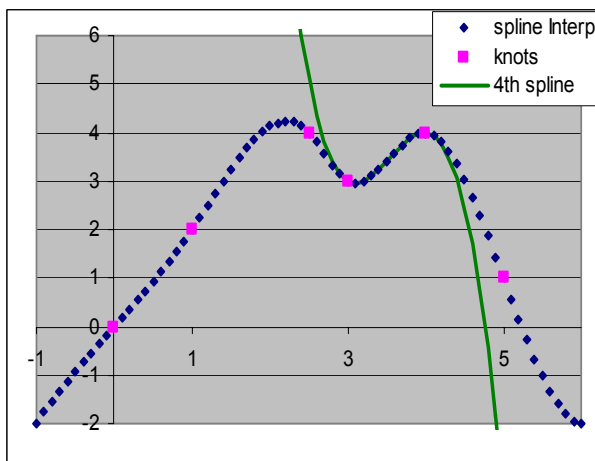
In the graphs below we can see the interpolated points (dotted line) fitting the data points and the cubic polynomials (green line) passing through the nodes of each segment. Each polynomial interpolates inside the proper segment. That is: the 1st spline works for  $0 \leq x < 1$ , the 2nd spline for  $1 \leq x < 2.5$ , and so on. In the graphs below are shown the entire interpolation line (left) and the 1<sup>st</sup> spline (right).



In the graphs below are shown the 2<sup>nd</sup> spline (left) and the 3<sup>rd</sup> spline (right)



In the graphs below are shown the 4<sup>th</sup> spline (left) and the 5<sup>th</sup> spline (right)

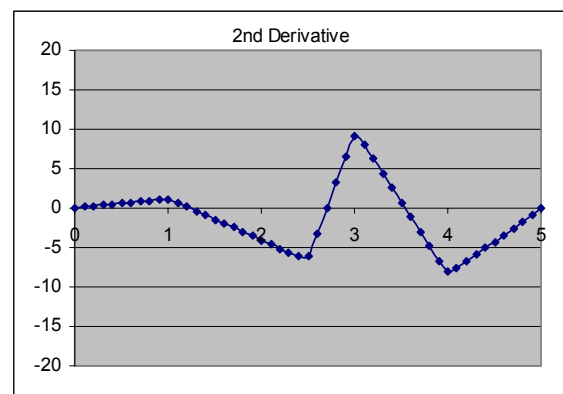
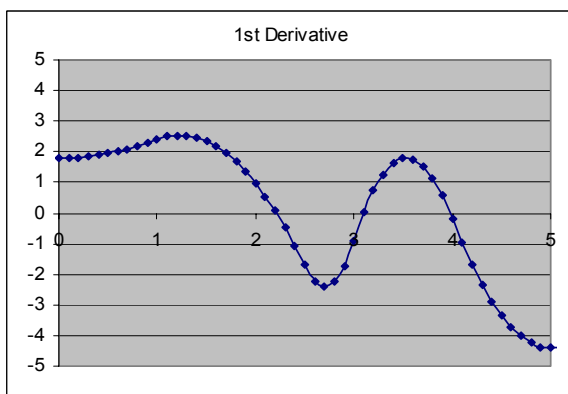


Let's examine the 1<sup>st</sup> and 2<sup>nd</sup> derivatives. We can compute them either analytically or numerically using – for example -the following derivative formulas:

$$y'(x_i) \cong (y_{i+1} - y_{i-1}) / 2\Delta x$$

$$y''(x_i) \cong (y_{i+1} - 2y_i + y_{i-1}) / \Delta x^2$$

In both ways, we get the following graphs



As we can see the 1<sup>st</sup> derivatives is smooth and the 2<sup>nd</sup> is continuous. This last feature is particularly appreciated in many fields of engineering. Although this algorithm is much slower than other polynomial interpolation methods, it has the advantage of



following the interpolated curve without the spurious oscillations that other schemes can create

## Cubic poly interpolation

In many documents we found sentences like this: “...a typical curve fit involves forming one polynomial equation through all  $n$  points of the given interval...”

We are induced to believe that for 6 knots we should choose a 6<sup>th</sup> degree polynomial and for 100 knots a 100<sup>th</sup> one!

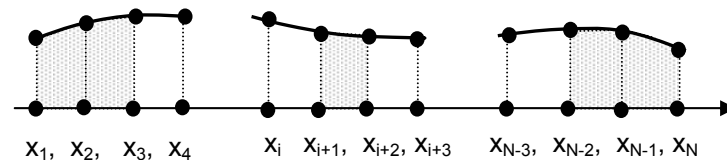
This is not all exact. We can apply for this kind of interpolation the same method of the spline. That is, we can freely choose the polynomial degree and then calculate it for a sub-set of consecutive knots.

The only difference is that we can use only the knots information and nothing else.

So, if we choose a 3<sup>th</sup> degree polynomial we can fit the first 4 points ( $y_1, y_2, y_3, y_4$ ).

Of course we can interpolate  $y(x)$  in any value between  $x_1$  and  $x_4$ , but we are induced to think that central values  $x_2 \leq x < x_3$  are better approximated. Moving to the next set of 4 nodes ( $y_2, y_3, y_4, y_5$ ) we obtain a new polynomial adapted to interpolate values for  $x_3 \leq x < x_4$ , and so on..

This method can be repeated for any internal segment, except for the first and the last one. In these cases we have to interpolate with the first and the last polynomial, tolerating a (probable) less accuracy.



## Formulas

Many algorithms can be used for computing the interpolation polynomial: formulas of Lagrange, Newton, Aitken, Everett, Taylor, Stirling, Bessel, Hermite, etc...

For simplicity, we choose the Newton cubic formula.

$$y(x) = y_1 + D(x_1, x_2) (x - x_1) + D(x_1, x_2, x_3) (x - x_1) (x - x_2) + \\ + D(x_1, x_2, x_3, x_4) (x - x_1) (x - x_2) (x - x_3)$$

where  $D$  are:

$$D(x_1, x_2) = (y_1 - y_2) / (x_1 - x_2)$$

$$D(x_2, x_3) = (y_2 - y_3) / (x_2 - x_3)$$

$$D(x_3, x_4) = (y_3 - y_4) / (x_3 - x_4)$$

$$D(x_1, x_2, x_3) = (D(x_1, x_2) - D(x_2, x_3)) / (x_1 - x_3)$$

$$D(x_2, x_3, x_4) = (D(x_2, x_3) - D(x_3, x_4)) / (x_2 - x_4)$$

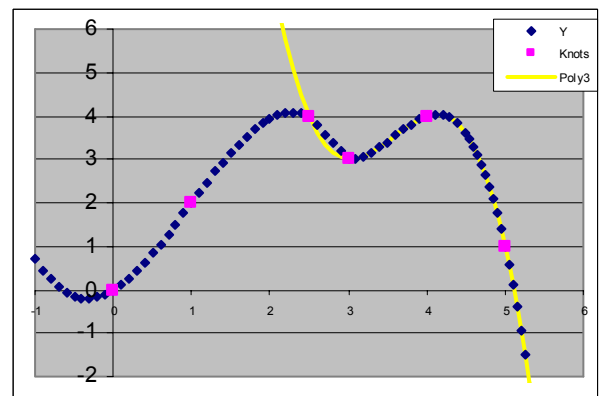
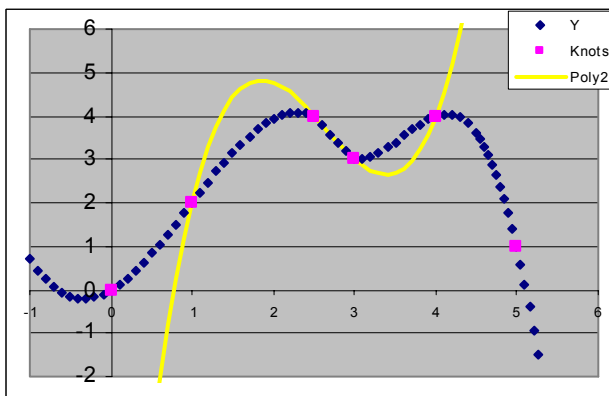
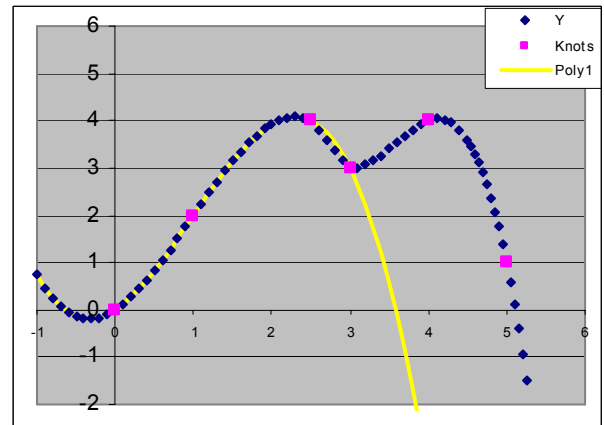
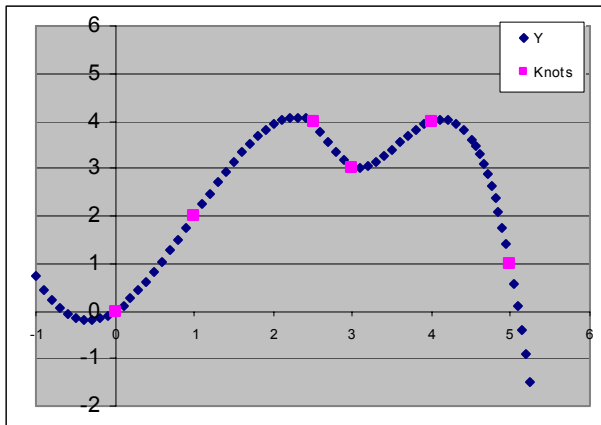
$$D(x_1, x_2, x_3, x_4) = (D(x_1, x_2, x_3) - D(x_2, x_3, x_4)) / (x_1 - x_4)$$

Example. Repeating the interpolation of the above example we get the following polynomial

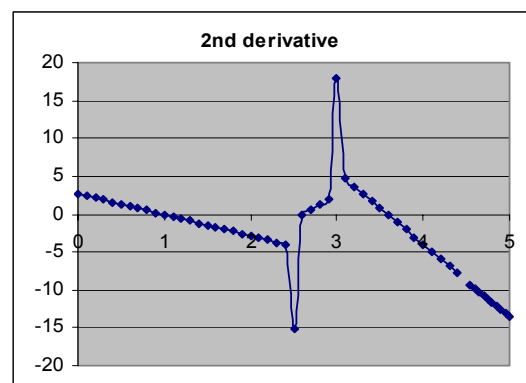
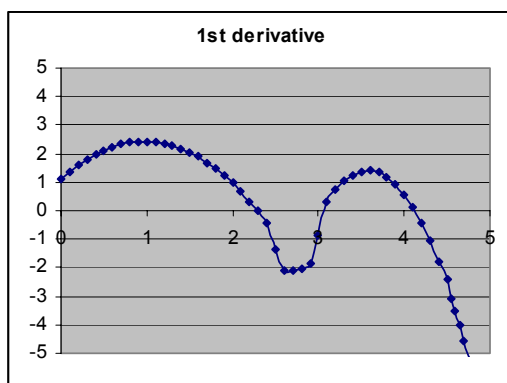
1st polynomial	$(33x + 41x^2 - 14x^3)/30$	$0 \leq x < 2.5$
2nd polynomial	$(-228 + 415x - 173x^2 + 22x^3)/18$	$2.5 \leq x < 3$
3rd polynomial	$(360 - 301x + 86x^2 - 8x^3)/5$	$3 \leq x < 5$

## Xnumbers Tutorial

In the graphs below we can see the interpolate points (dotted line) fitting to data points and the cubic polynomials (yellow) passing through the nodes of each segment. Each polynomial interpolates inside the proper segment.



Let's compute numerically the 1<sup>st</sup> and 2<sup>nd</sup> derivatives. We obtain the following graphs



In the last plot we clearly see the spikes of the 2<sup>nd</sup> derivative. In engineering applications such as mechanical motions, spurious spikes of the 2<sup>nd</sup> derivative produce unwanted vibrations transmitted to the other parts of the system: gears, bearing, etc.. This involves higher noise, wear, etc... On the contrary, spline motions can great reduce these drawbacks.

## Observations

Both methods can provide an acceptable interpolation in the entire range of  $x \in [0, 5]$ . Slight differences among interpolate values exist, but we cannot say that one is better than other because the function values between nodes is unknown and both models are conceptually equivalent.

But there is an aspect that make the difference and it is the 2<sup>nd</sup> derivative of the spline interpolation. Although this algorithm is much slower than other polynomial interpolation methods, it has the advantage of giving an exact fit to the curve without the spurious oscillations that other schemes can create.

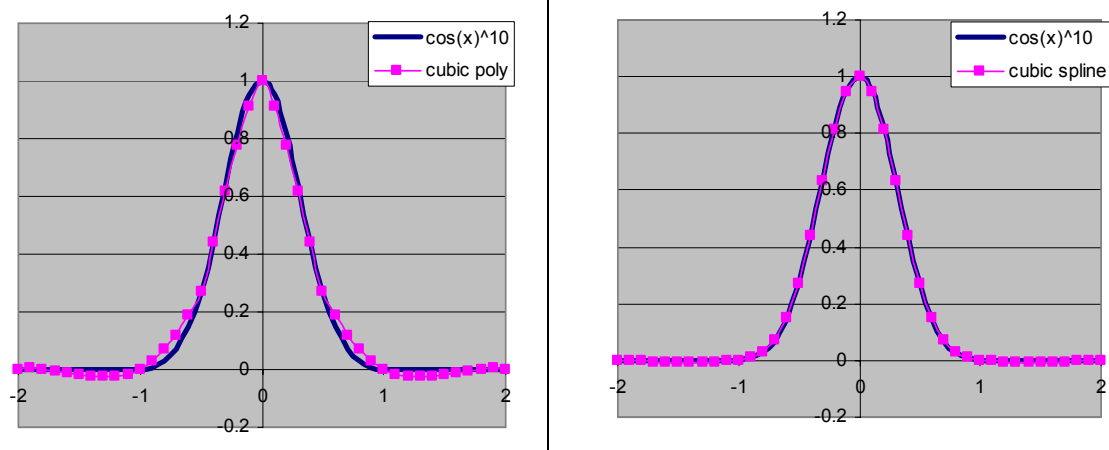
### Other test functions

In our last example we have found that both methods can provide acceptable interpolation for all range of  $x$ . Thus, there are same case, that the superiority of the spline interpolation is more evident. Gerald [2] used the “bump” test case to illustrate problems with other interpolation methods. Let’s see.

Interpolate the following knots

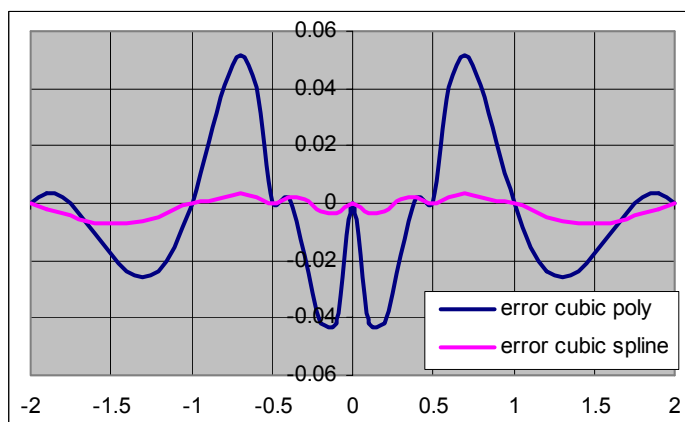
$$Y = (\cos(x))^{10}, \text{ for } x = -2, -1, -0.5, 0, 0.5, 1, 2$$

Plotting the interpolated values with a step of 0.1, we get the following graphs



The curves appear acceptable in both graphs. The second shows a closer fit near the points  $x = 1$  and  $x = -1$  where are "knees" of the curve.

But matching the error plots, we see clearly the better accuracy of the spline interpolation.



As we can see, the amplitude error of the cubic polynomial is much more than the spline.

We can show that an higher order of the interpolation polynomial, is even worst

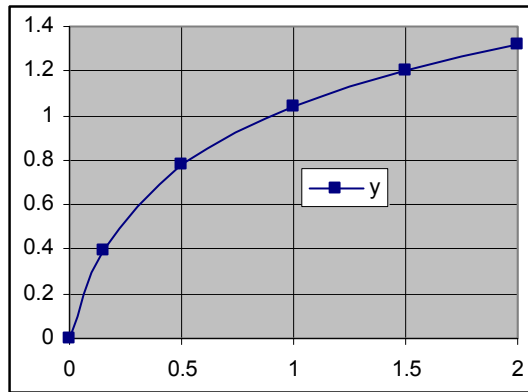
In this case, the cubic spline is the better choice

## High and low interpolation degree

Surprisingly, an high degree of the interpolation polynomial does not mean high accuracy. On the contrary, we often choose a low degree polynomial to get the maximum accuracy. Let's see this example.

Interpolate the following knots

$y(x) = 1 + \log_{10}(x+0.1)$  , for  $x = 0, 0.15, 0.5, 1, 1.5, 2$  , with step  $\Delta x = 0.1$

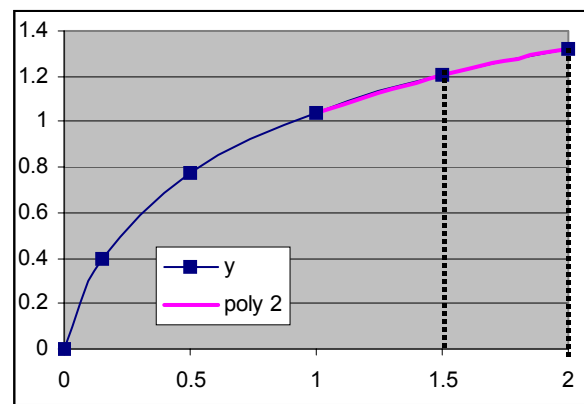
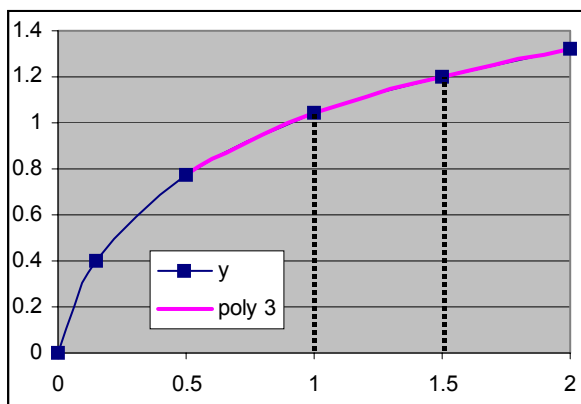
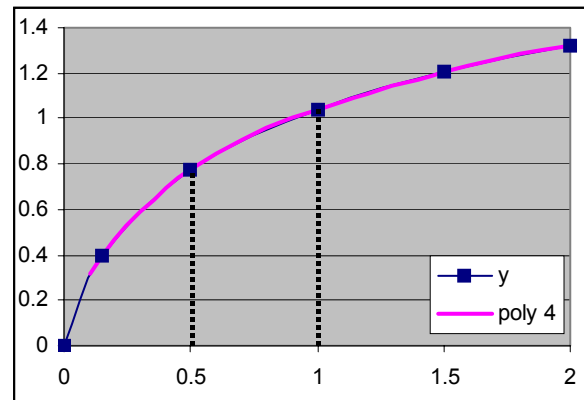
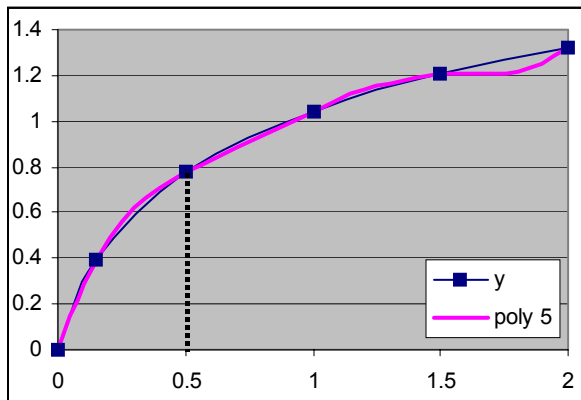


For clarity, we have draw the function line and the knots.

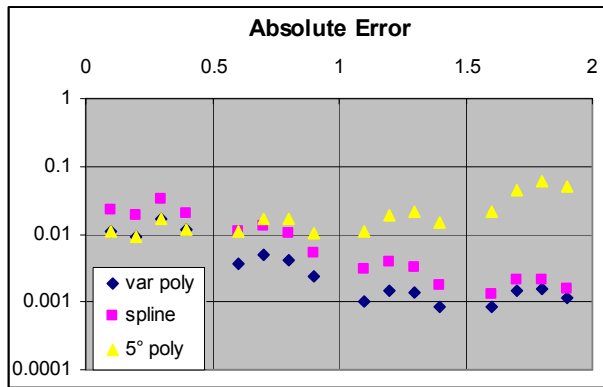
The attack strategy can be:

Interpolation range	Knots used	Poly degree
0 , 0.5	all	5°
0.5 , 1	0.15 , 0.5 , 1 , 1.5 , 2	4°
1 , 1.5	0.5 , 1 , 1.5 , 2	3°
1.5 , 2	1 , 1.5 , 2	2°

Interpolations near the zero are done with a 5<sup>th</sup> degree polynomial (the maximum), while at the end of the range we use a simpler parabolic interpolation. The graphs below show better how it works.



Now we compare the absolute errors among this interpolation schema with the spline and with the 5<sup>th</sup> degree polynomial interpolation over the entire range.



As we can see, we have a good generally accuracy of about 0.01, but the interpolation with the 5th degree polynomial is not the best. In fact, the average absolute errors obtained are:

Method	Avg error
5 <sup>th</sup> degree polynomial	0.016
cubic spline	0.0072
variable polynomial	0.0035

Note that, for  $1 < x < 2$ , the simply parabolic interpolation is absolutely more accurate than the 5<sup>th</sup> degree polynomial, and even more than the cubic spline.

What can we get from all that? As rules of thumb we can say that “Respect to the values that we want to interpolate is better to use few knots but near than many knots but distant”.

## Continued fraction interpolation

Continued fractions are often a powerful ways of interpolation when we work near the functions poles.

### Formulas

For N knots, the continued fraction expansion is:

$$\begin{aligned}
 y(x) &= a_1 + (x-x_1)/d_1 \\
 d_1(x) &= a_2 + (x-x_2)/d_2 \\
 d_2(x) &= a_3 + (x-x_3)/d_3 \\
 d_3(x) &= a_4 + (x-x_4)/d_4 \\
 &\dots\dots\dots \\
 d_{N-1}(x) &= a_N
 \end{aligned}$$

The coefficients  $a_i$  can be computed by the following iterative algorithm

```

For i = 1 to N ,  $a_i = y_i$ 
For k = 1 to N-1
For i = k+1 to N
If  $|a_i - a_{i-1}| > 10^{-14}$ 
 $a_i = (x_i - x_k) / (a_i - a_{i-1})$ 
else
 $a_i = 0$ 
    
```

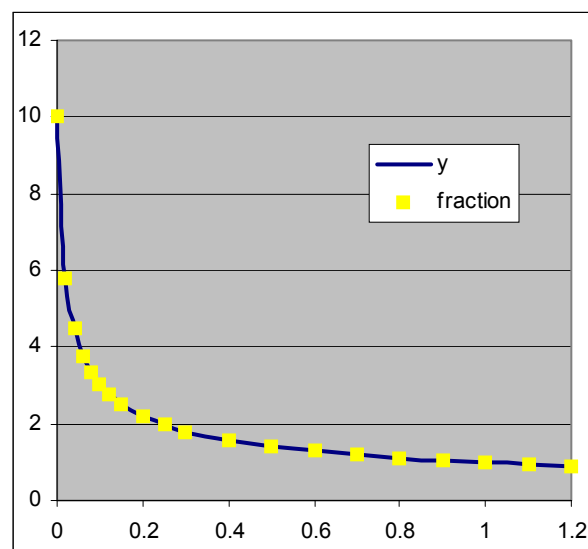
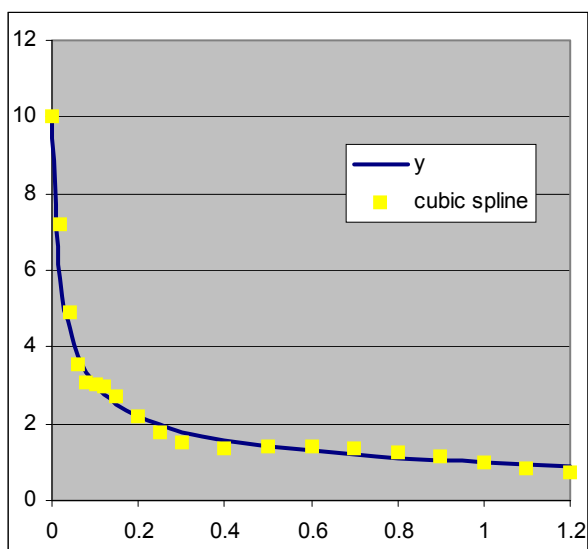
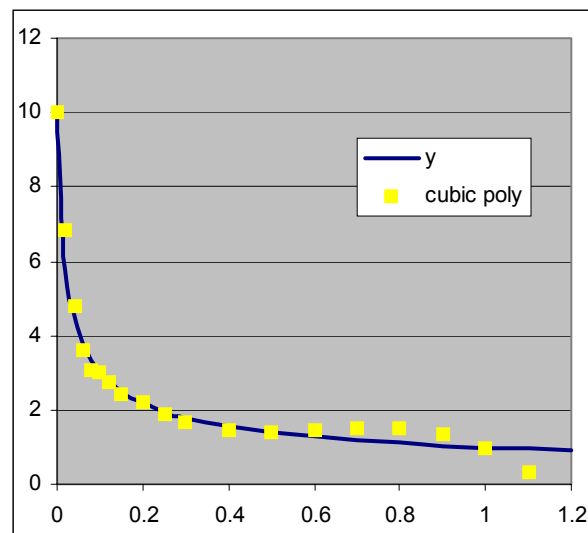
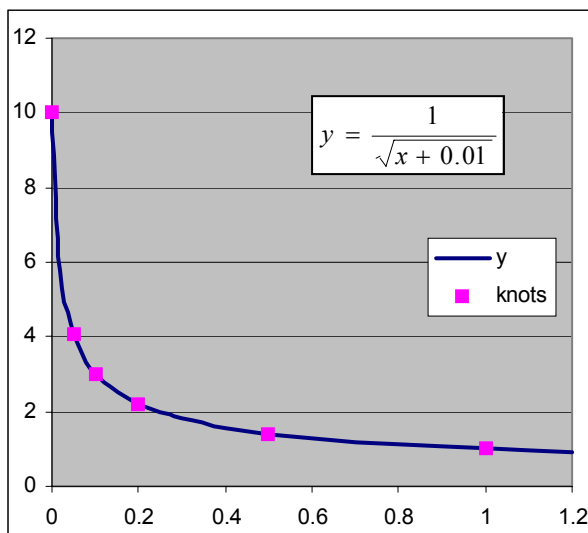
NOTE. In XNUMBERS the continued fraction coefficients can be obtained by the function **Fract\_Interp\_Coef** and the interpolation value with **Fract\_Interp**

Example. Interpolate the following dataset

$$y(x) = 1/(x+0.01)^{1/2}, \text{ for } x = 0, 0.05, 0.1, 0.2, 0.5, 1, \text{ with step } \Delta x = 0.1$$

In the graphs below we have plotted the interpolated values obtained with three different methods: cubic polynomial, cubic spline and continue fraction

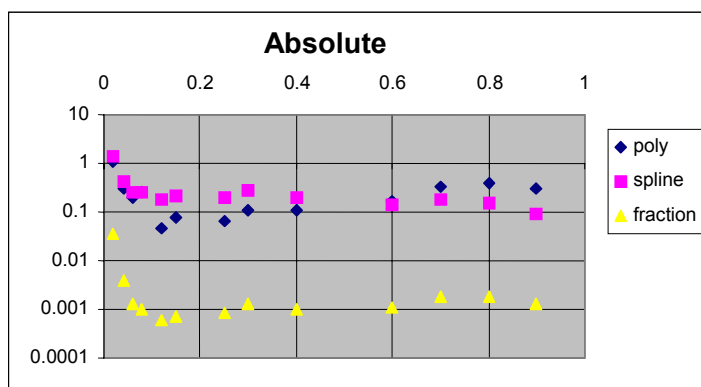
Note that  $y(x)$  have a poles in  $x = -0.01$  very near to the node  $x = 0$



We can see a good general accuracy except for the final part of the polynomial interpolation method. In this case, the worst accuracy is concentrated where the function is more flat, but, surprisingly, this perturbation is due to the distant pole in  $x = -0.01$ .

We note also that both spline and fraction methods keep a good accuracy also for point external at the interpolation range ( extrapolation for  $x > 1$  )

### Absolute Error Plot



As we can see, the average error of the continued fraction is much lower than other methods

Method	Avg. error
Cubic spline	0.22
Cubic poly	0.19
Continued fraction	0.003

## Differential Equations

Xnumbers contains functions for solving the following differential problem of the 1st order with initial conditions (Cauchy's problem):

$$y' = f(t, y) \quad , \quad y(t_0) = y_0$$

and for solving the ordinary differential system written as:

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}) \quad , \quad \mathbf{y}(t_0) = \mathbf{y}_0 \quad \Leftrightarrow \quad \begin{cases} y'_1 = f_1(t, y_1, y_2 \dots y_n) \\ y'_2 = f_2(t, y_1, y_2 \dots y_n) \\ \dots \\ y'_n = f_n(t, y_1, y_2 \dots y_n) \end{cases} , \quad \begin{cases} y_1(t_0) = y_{10} \\ y_2(t_0) = y_{20} \\ \dots \\ y_n(t_0) = y_{n0} \end{cases}$$

## ODE Runge-Kutta 4

**= ODE\_RK4(Equations, VarInit, Step, [Par, ...])**

This function integrates numerically a 1<sup>st</sup> order ordinary differential equation or a 1<sup>st</sup> order differential system, with the Runge-Kutta formula of 4<sup>th</sup> order

$$\begin{aligned} k_1 &= f(t_i, y_i) \\ k_2 &= f(t_i + 0.5 h, y_i + 0.5 h k_1) \\ k_3 &= f(t_i + 0.5 h, y_i + 0.5 h k_2) \\ k_4 &= f(t_i + h, y_i + h k_3) \\ y_{i+1} &= y_i + \frac{h}{6} (k_1 + 2k_2 + 2k_3 + k_4) \end{aligned}$$

"Equations" is a math expression string containing the equation to solve. For a system It is a vector of equations. Examples of correct equation definition are:

$$y' = -2*y*x \quad , \quad v' = 2*x-v^2+v \quad , \quad y1' = -3*y1+y2+\sin(10*t)$$

Each string may contain symbolic functions with variables, operators, parenthesis and other basic functions.

The parameter "VarInit" is a vector containing the initial values. It has two values for two variables [ to, yo ].

For a system with n+1 variables, "Varinit" is an (n+1) vector [ to, y10, y20, ..., yn0 ].

The parameter "Step" is the integration step.

The optional parameter "Par" contains the values of other extra parameters of the equations.

Let's see how it works with an example

Solve numerically the following Cauchy's problem for  $0 \leq x \leq 3$

$$y' = -2xy \quad , \quad y(0) = 1$$

We know that the exact solution is  $y = e^{-x^2}$

For performing the computation we can arrange a sheet like the following

	A	B	C	D	E	F	G
1							
2							
3							
4	x	y				h	diff. equation
5	0	1				0.1	$y' = -2*x*y$
6	0.1	0.99005					
7							
8							

As we can see, we have written in cell G5 the differential equation

$$y' = -2*x*y$$

In the range A5:B5 we have inserted the starting values of x and y. Note that we have written the labels just above their values. Labels are necessary for the correct variables assignment

Finally, in the range A6:B6 - just below the starting values - we have inserted the ODE\_RK4, that returns the value  $y(0.2) = 0.9607893...$  with a good accuracy of about  $1E-7$  (compare with the exact solution)

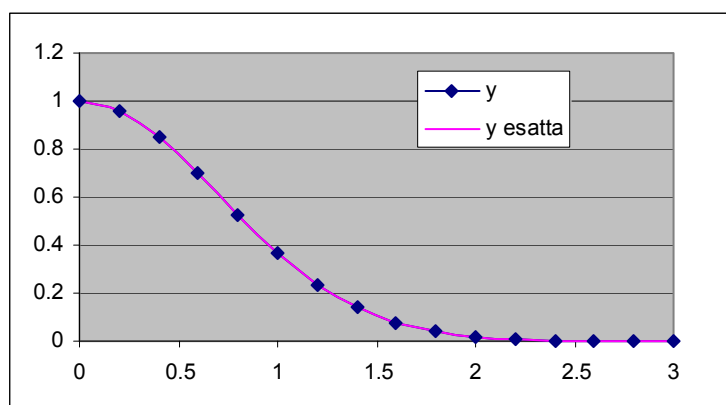
	x	y	y (exact)	error
5	0	1	1	0
6	0.1	0.99005	0.9900498	4.158E-10
7	0.2	0.960789	0.9607894	3.917E-09
8	0.3	0.913931	0.9139312	1.125E-08
9	0.4	0.852144	0.8521438	1.65E-08
10	0.5	0.778801	0.7788008	2.528E-09
11	0.6	0.697676	0.6976763	6.111E-08
12	0.7	0.612627	0.6126264	2.158E-07
13	0.8	0.527293	0.5272924	5.065E-07
14	0.9	0.444859	0.4448581	9.705E-07
15	1	0.367881	0.3678794	1.625E-06
16	1.1	0.2982	0.2981973	2.459E-06
17	1.2	0.236931	0.2369278	3.428E-06
18	1.3	0.184524	0.1845195	4.459E-06

**Tip:** In order to get all other values, select the range A6:B6 and simply drag it down.  
The cells below will be filled automatically

Only remember to fix the constant cells in the function with the \$ symbol

=ODE\_RK4(\$G\$5,A5:B5,\$F\$5)

We have also added the column with the exact values in order to check the approximation error. Both exact and approximated solutions are plotted in the following graph



The fit, in this case, seems excellent.



## Xnumbers Tutorial

If you need you can include parameters inside the differential equation

Example. Solve the following differential problem

$$y' = -k \cdot x^n \cdot y$$

$$y(0) = 1$$

where  $k = 2$  and  $n = 1$

	A	B	C	D	E
1	<b>diff. equation</b>		<b>h</b>	<b>k</b>	<b>n</b>
2	$y' = -k \cdot x^n \cdot y$		0.1	2	1
3	<b>=ODE_RK4(\$A\$2,A6:B6,\$C\$2,\$D\$2,\$E\$2)</b>				
4					
5	<b>x</b>	<b>y</b>			
6	0	1			
7	0.1	0.9900498			
8	0.2	0.9607894			
9	0.3	0.9139312			
10	0.4	0.8521438			

Note that we have added the labels "k" and "n" above the cells D2 and E2. In this way, the parser will correctly substitute the value 2 for the variable "k" and 1 for the variable "n". in the differential equation

Do not forget the labels "x" and "y" in the cells A5 and B5

Example: Solve the following linear differential equation

$$y' + \frac{1}{x} y = a \cdot x^n, \quad y(1) = 0$$

For  $n = 3$  and  $a = 1$

Rearranging, we get

$$y' = a \cdot x^n - \frac{y}{x}, \quad y(1) = 0$$

	A	B	C	D	E
1	<b>diff. equation</b>		<b>h</b>	<b>a</b>	<b>n</b>
2	$y' = a \cdot x^n - y/x$		0.1	1	3
3	<b>=ODE_RK4(\$A\$2,A6:B6,\$C\$2,\$D\$2,\$E\$2)</b>				
4					
5	<b>x</b>	<b>y</b>			
6	1	0			
7	1.1	0.1110019			
8	1.2	0.2480535			
9	1.3	0.417374			
10	1.4	0.6254631			

Note the labels "a" and "n" above the cells D2 and E2. In this way, the parser will substitute the value 1 for the variable "a" and 3 for the variable "n". in the differential equation

Do not forget the labels "x" and "y" in the cells A5 and B5

With the step  $h = 0.1$ , we have a numerical solution with a very good approximation comparing with the exact solution  $y = (x^5 - x)/5x$ , (better than  $1E-6$ )

## Xnumbers Tutorial

This function can be used to solve ordinary differential systems.

Example: Solve numerically the following differential system, where  $v(t)$  and  $i(t)$  are the voltage and the current of an electric network

$$\begin{cases} v' = i - 7 \cdot v \\ i' = -5 \cdot i + 15 \cdot v \end{cases} \quad \begin{cases} v(0) = 10 \\ i(0) = 0 \end{cases}$$

	A	B	C	D
1	$v' = i - 7 \cdot v$			h
2	$i' = -5 \cdot i + 15 \cdot v$			0.05
3				
4	{=ODE_RK4(\$A\$1:\$A\$2;A7:C7;\$D\$2)}			
5				
6	t	v	i	
7	0	10	0	
8	0.05	7.185458333	5.58875	
9	0.1	5.371308656	8.448001073	
10	0.15	4.174289895	9.701682976	
11	0.2	3.360887149	10.02694722	
12	0.25	2.788538799	9.830311635	
13	0.3	2.369953963	9.354496143	

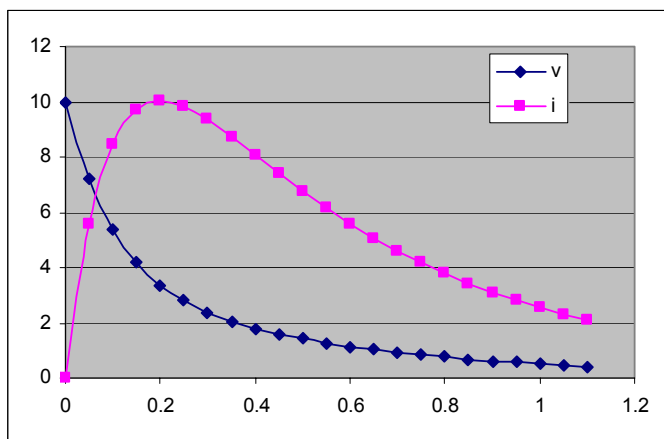
The computation can be arranged as following.

Write the variables labels in the row 6. The labels "v" and "i" must be the same that you have written in the equations. Just one row below, insert the starting values in the same order.

Select the range A8:C8 and insert the function ODE\_RK4. The first step will be returned.

Now select this row and drag it down for evaluating all the steps that you need

The graph below show the transient of  $v(t)$  and  $i(t)$  with good accuracy



Note that you can change the step "h" in order to re-compute the transient in a very fast and quick way.

Optional constant parameters can be arranged. For example if you want to add a parameter R, independent from the time "t", write:

	A	B	C	D
1	$v' = i - 7 \cdot v$		R	h
2	$i' = -R \cdot i + 15 \cdot v$		5	0.05
3				
4	{=ODE_RK4(\$A\$1:\$A\$2;A7:C7;\$D\$2;\$C\$2)}			
5				
6	t	v	i	
7	0	10	0	
8	0.05	7.185458333	5.58875	
9	0.1	5.371308656	8.448001073	

Constant parameters can be written in any part of the worksheet. You need only to add the labels with the same symbols with they appear in the differential equations. In this case, we have added the label "R" in the cell C1, upon its values.

You can add as many optional parameters that you like

## ODE Multi-Steps

Another very popular method for integrating ordinary differential equations adopts the multi-step Adams' formulas. Even if a little formally complicated, they are very fast, and adapted to build a large family of ODE integration methods

The multi-step Adams' formulas can be generally written as:

$$y_{i+1} = y_i + \frac{h}{M} \sum_{k=1}^N \beta_{N-k} \cdot y'_{i-k+1} = y_i + \frac{h}{M} (\beta_{N-1} \cdot y'_i + \beta_{N-2} \cdot y'_{i-1} + \dots + \beta_0 \cdot y'_{i-N+1})$$

$$y_{i+1} = y_i + \frac{h}{M} \sum_{k=1}^N \beta_{N-k} \cdot y'_{i-k+2} = y_i + \frac{h}{M} (\beta_{N-1} \cdot y'_{i+1} + \beta_{N-2} \cdot y'_i + \dots + \beta_0 \cdot y'_{i-N+2})$$

where  $y'_i = f(t_i, y_i)$   $t_i = t_0 + h \cdot i$

The first formula generates the explicit formulas – also called predictor formulas.  
The second formula generates the implicit formulas – also called corrector formulas.  
The number N is the order of the formula. A formula of N order requires N starting steps. Of course, formulas with high N are more accurate.

For N = 1 we get the popular Euler integration formulas

$y_{i+1} = y_i + h \cdot y'_i$	Euler's predictor (1 step)
$y_{i+1} = y_i + \frac{h}{2} \cdot (y'_{i+1} + y'_i)$	Trapezoid formula corrector (1 step)

Their errors are given by

$e \approx \frac{1}{2} h^2 y^{(2)}$	Error predictor 1 <sup>st</sup> order
$e \approx -\frac{1}{12} h^3 y^{(3)}$	Error corrector 2 <sup>st</sup> order

For N = 4 we get the popular Adams-Bashfort-Moulton predictor-corrector formulas

$y_{i+1} = y_i + \frac{h}{24} \cdot (55y'_i - 59y'_{i-1} + 37y'_{i-2} - 9y'_{i-3})$	Predictor (4 step)
$y_{i+1} = y_i + \frac{h}{24} \cdot (9y'_{i+1} + 19y'_i - 5y'_{i-1} + y'_{i-2})$	Corrector (4 step)

Their errors are given by

$e \approx \frac{251}{720} h^5 y^{(5)}$	Error predictor 4 <sup>th</sup> order
$e \approx -\frac{19}{720} h^5 y^{(5)}$	Error corrector 4 <sup>th</sup> order

There are a large set of predictor-corrector formulas

## Multi-step coefficients tables

The following tables list the coefficients for the Adams' predictor-corrector formulas up to the 9<sup>th</sup> order and relative errors

### Multi-step Predictor coefficients

N ⇒	1	2	3	4	5	6	7	8	9	10
M	1	2	12	24	720	1440	60480	120960	3628800	7257600
β 0	1	-1	5	-9	251	-475	19087	-36799	1070017	-2082753
β 1		3	-16	37	-1274	2877	-134472	295767	-9664106	20884811
β 2			23	-59	2616	-7298	407139	-1041723	38833486	-94307320
β 3				55	-2774	9982	-688256	2102243	-91172642	252618224
β 4					1901	-7923	705549	-2664477	137968480	-444772162
β 5						4277	-447288	2183877	-139855262	538363838
β 6							198721	-1152169	95476786	-454661776
β 7								434241	-43125206	265932680
β 8									14097247	-104995189
β 9										30277247

### Multi-step Corrector coefficients

N ⇒	1	2	3	4	5	6	7	8	9	10
M		2	12	24	720	1440	60480	120960	3628800	7257600
β 0		1	-1	1	-19	27	-863	1375	-33953	57281
β 1		1	8	-5	106	-173	6312	-11351	312874	-583435
β 2			5	19	-264	482	-20211	41499	-1291214	2687864
β 3				9	646	-798	37504	-88547	3146338	-7394032
β 4					251	1427	-46461	123133	-5033120	13510082
β 5						475	65112	-121797	5595358	-17283646
β 6							19087	139849	-4604594	16002320
β 7								36799	4467094	-11271304
β 8									1070017	9449717
β 9										2082753

### Error coefficient

The general error is  $e \approx -k \cdot h^{n-1} y^{(n-1)}$  where k is given by the following table

N ⇒	1	2	3	4	5	6	7	8	9	10
predictor	0.5	0.41667	0.375	0.34861	0.32986	0.31559	0.30422	0.29487	0.28698	0.28019
corrector	-	-0.0833	-0.0417	-0.0264	-0.0188	-0.0143	-0.0114	-0.0094	-0.0079	-0.0068

### The predictor-corrector algorithm

Usually the multi-step formulas, implicit and explicit, are used together to build a Predictor-Corrector algorithm. Here is how to build the 2<sup>nd</sup> order PEC algorithm (Prediction-Evaluation-Correction).

It uses the Euler's formula as predictor and the trapezoidal formula as corrector

Prediction	Evaluation	Correction
$y_{p1} = y_0 + h f(t_0, y_0) \Rightarrow$	$f(t_1, y_{p1}) \Rightarrow$	$y_1 = y_0 + h/2 [f(t_0, y_0) + f(t_1, y_{p1})]$
$y_{p2} = y_1 + h f(t_1, y_{p1}) \Rightarrow$	$f(t_2, y_{p2}) \Rightarrow$	$y_2 = y_1 + h/2 [f(t_1, y_1) + f(t_2, y_{p2})]$
$y_{p3} = \dots$	$\dots$	$\dots$

The value  $y_1$  can be reused to evaluate again the function  $f(t_1, y_1)$ , that can be used in the corrector formula to obtain a more accurate value for  $y_1$ .

If we indicate the first value obtained by the corrector with  $y_1^{(1)}$  and the second value with  $y_1^{(2)}$  we can arrange a new following schema

Prediction	Evaluation	Correction	Evaluation	Correction
$y_{p1} \Rightarrow$	$f(t_1, y_{p1}) \Rightarrow$	$y_1^{(1)} \Rightarrow$	$f(t_1, y_1^{(1)}) \Rightarrow$	$y_1^{(2)}$

This is the so called PECEC or  $P(EC)^2$  schema.

The group EC can also be repeated m-times or even iterated still the convergence. In these cases we have the schemas  $P(EC)^m$  and  $P(EC)^\infty$  respectively.

Note that, for  $m \gg 1$  the final accuracy depends mainly by the corrector.

Let's come back to the PEC schema.

We note that, at the step, we use the value  $f(t_1, y_{p1})$  to predict the new value  $y_{p2}$

We could increase the accuracy if we take the better approximation  $f(t_1, y_1)$ .

The new schema becomes:

Prediction	Evaluation	Correction	Evaluation
$y_{p1} \Rightarrow$	$f(t_1, y_{p1}) \Rightarrow$	$y_1 \Rightarrow$	$f(t_1, y_1) \Rightarrow$

This schema is called PECE and it is used very often being a reasonable compromise between the accuracy and the computation effort.

Using different schemas with different predictor-corrector formulas we can build a wide set of algorithms for the ODE integration. Of course they are not equivalent at all. Some of them have a high accuracy, others show a better efficiency and others have a better stability. This last characteristic may be very important for long integration intervals. In fact, the most algorithms, especially those with higher order, become unstable when the integration step grows over a limit. Algorithms that are stable for any integration step (so called A-stable algorithms) are much appreciated, but unfortunately they have a low general accuracy.

One A-stable algorithm is the  $P(EC)^\infty$  with the Euler's formula as predictor and the trapezoid formula as corrector. It is a 2<sup>nd</sup> order algorithm

## Predictor- Corrector

= ODE\_PRE(yn, f, h)

= ODE\_COR(yn, fp, f, h)

These functions perform the integration of the ordinary differential equations with the popular multi-step predictor-corrector Adams' formulas

$$y' = f(t, y) \quad , \quad y(t_0) = y_0$$

The first function returns the predictor value  $y_{n+1,p}$  while the second function returns the corrector  $y_{n+1}$ .

The parameter "yn" is the last point of the function y(t).

The parameter "f" is a vector containing the last N values of the derivative of y(t). That are the last N-1 values of the corrector.

The parameter "fp", only for the corrector, is the best approximation of the derivative of y(t) at the step n+1. Usually it is provided by a predictor formula

The parameter "h" sets the integration step

## PECE algorithm of 2<sup>nd</sup> order

Now we see how arrange a PECE algorithm of 2<sup>nd</sup> order to solve a the following differential problem.

$$y' = -xy^2 \quad , \quad y(0) = 2$$

Let's set in a cell that we like the integration step "h" and then the heading of the data table. We set separate columns for predictor and corrector values

	A	B	C	D	E
1	Predictor-Corrector (PECE) of 2° order				
2					
3	h =	0.2		=B6	
4					
5	x	yp	fp	yc	fc
6	0	2	0	2	0
7					
8		=-\$A6*B6^2			=\$A6*D6^2
9					

Build the first row.

Begin to insert the starting values ( $x_0, y_0$ ) in the cells A6 and B6 respectively, and the formula evaluations of  $f(x,y)$  in the cell C6 and E6. The corrector value is set equal to the starting value B6

	A	B	C	D	E
1	Predictor-Corrector (PECE) of 2° order				
2					
3	h =	0.2			
4	=A6+\$B\$3		=\$A7*B7^2		=\$A7*D7^2
5	x	yp	fp	yc	fc
6	0	2	0	2	0
7	0.2	2	-0.8	2	-0.8
8					

The second row is a bit more complicated. Let's see.

Select the first row A6:E6 and drag it down one row. This will copy the formula for **fp** and **fc** Insert in the cell A7 the increment formula

$$x_{i+1} = x_i + h$$

Now we have to add the predictor and corrector function

	A	B	C	D	E
1	Predictor-Corrector (PECE) of 2° order				
2					
3	h =	0.2	=ODE_PRE(D6;E6;\$B\$3)		
4					
5	x	yp	fp	yc	fc
6	0	2	0	2	0
7	0.2	=ODE_PRE(D6	-0.8	1.92	-0.73728
8					

Insert in the cell B7

=ODE\_PRE(yn, f, h)

“yn” is the last value of y(x). contained in D6. “f” is the last value of f(x,y) contained in E6. “h” is the step B3.

	A	B	C	D	E
1	Predictor-Corrector (PECE) of 2° order				
2					
3	h =	0.2			
4					
5	x	yp	fp	yc	fc
6	0	2	0	2	0
7	0.2	2	-0.8	=COR(D6;C7	-0.73728
8					
9	=ODE_COR(D6;C7;E6;\$B\$3)				

Insert in the cell D7

=ODE\_COR(yn, fp, f, h)

Where “yn” is the last value of y(x). In that case is D6. “f” is the last value of f(x,y), E6.

“fp” is the predicted. value of f(x,y), C7 in this case.

“h” is the constan step.

	A	B	C	D	E
1	Predictor-Corrector (PECE) of 2° order				
2					
3	h =	0.2			
4					
5	x	yp	fp	yc	fc
6	0	2	0	2	0
7	0.2	2	-0.8	1.92	-0.73728
8	0.4	1.772544	-1.2567649	1.72059551	-1.1841796
9	0.6	1.4837596	-1.3209255	1.470085	-1.2966899
10	0.8	1.21074701	-1.1727267	1.22314334	-1.1968637
11	1	0.9837706	-0.9678046	1.00667651	-1.0133976
12	1.2	0.80399699	-0.7756934	0.82776741	-0.8222387

Now the setting of the PECE algorithm of 2<sup>nd</sup> order is completed. Select the second row A7:E7 and drag it down in order to calculate the steps that you want.

The  $y_p$  and  $y_c$  values can be compared with the ones of the exact solution.

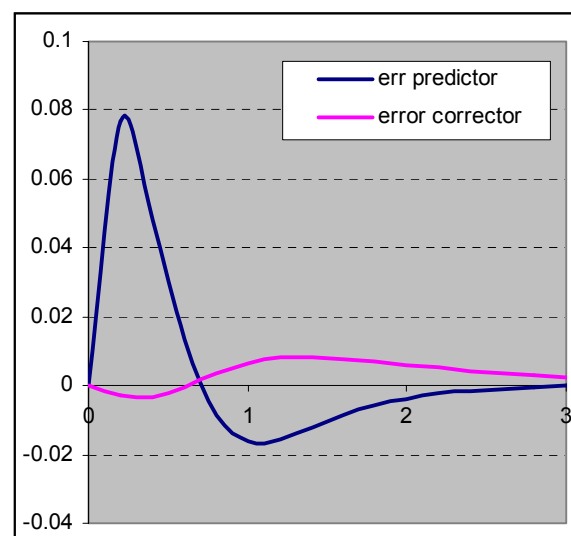
$$y = \frac{2}{1+x^2}$$

The differences:

$$d_{ip} = y_{ip} - y(x_i)$$

$$d_{ic} = y_{ic} - y(x_i)$$

are plotted in the graph at the right  
We note clearly the characteristic behavior of the predictor-corrector algorithm. The second formula refines the approximation of the first one.  
The final accuracy of PECE algorithm is practically the accuracy of the corrector



## PECE algorithm of 4<sup>th</sup> order

Now we solve the above differential equation with a 4<sup>th</sup> order PECE algorithm using the 4 steps Adams-Bashfort-Moulton formulas

$$y' = -xy^2, \quad y(0) = 2$$

To start this algorithm needs 4 steps. A good set of starting steps is:

x	y(x)
0	2
0.2	1.9230769231
0.4	1.7241379310
0.6	1.4705882353

We do not investigate here how to get the extra 3 values (they could come by Runge-Kutta method or by Taylor series approximation). The only thing that we have to point out is that these values must be sufficiently accurate in order to not degraded the global accuracy of the algorithm

The first 4 rows of the PECE algorithm are built as shown in the previous example.

	A	B	C	D	E
1	<b>Predictor-Corrector (PECE) of 4<sup>o</sup> order</b>				
2					
3	h	0.2			
4					
5	x	yp	fp	yc	fc
6	0	2	0	2	0
7	0.2	1.9230769	-0.739645	1.9230769	-0.739645
8	0.4	1.7241379	-1.1890606	1.7241379	-1.1890606
9	0.6	1.4705882	-1.2975779	1.4705882	-1.2975779
10	0.8	=ODE_PRE(E	-1.2151057	1.217386	-1.1856229
11					
12		=ODE_PRE(B9;E6:E9;\$B\$3)			
13					

The first 4 values of yp and yc are the same.

Now let's insert in the cell B10

=ODE\_PRE(yn, f, h)

where "yn" is the last value of y(x), D9 in that case.

"f" is a vector of the last four values of f(x,y), E6:E9 in this case.

"h" is the step B3.

	A	B	C	D	E
1	<b>Predictor-Corrector (PECE) of 4<sup>o</sup> order</b>				
2					
3	h	0.2			
4					
5	x	yp	fp	yc	fc
6	0	2	0	2	0
7	0.2	1.9230769	-0.739645	1.9230769	-0.739645
8	0.4	1.7241379	-1.1890606	1.7241379	-1.1890606
9	0.6	1.4705882	-1.2975779	1.4705882	-1.2975779
10	0.8	1.2324293	-1.2151057	=ODE_COR(D9;C	-1.1856229
11					
12		=ODE_COR(D9;C10;E7:E9;\$B\$3)			
13					

Insert in the cell D10

=ODE\_COR(yn, fp, f, h)

where "yn" is the last value of y(x). In that case D9.

"f" is a vector of the last 3 values of f(x,y), E7:E9.

"fp" is the predicted value of f(x,y), C10 in this case.

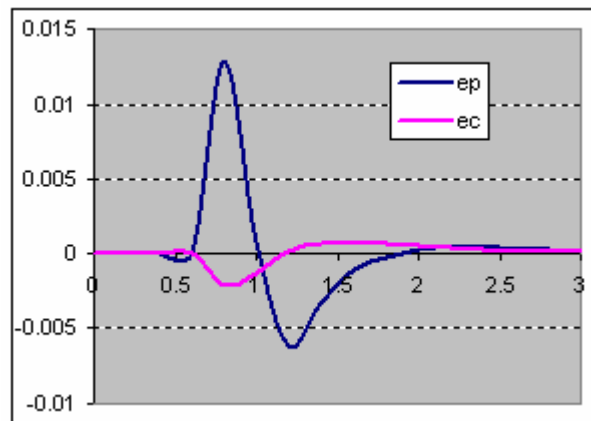
"h" is the step B3

Now the setting of the PECE algorithm of 4<sup>th</sup> order is completed. Select the 5<sup>th</sup> row and drag it down in order to calculate the steps you want.

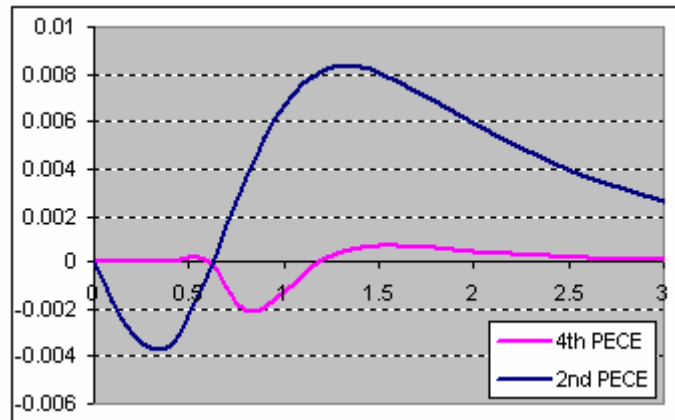


## Xnumbers Tutorial

The predictor-corrector error curves are shown in the following graph



In order to compare the accuracy of the solutions of this algorithm with the 2<sup>nd</sup> order algorithm of the previous example let's draw both the error curves in a same graph



As we can see, the 4<sup>th</sup> order algorithm is evidently more accurate than the 2<sup>nd</sup> order. On the other hand, the first one requires an extra work for providing 3 starting points.

## Nonlinear Equations

### Bisection

**=Zero\_bisec(a, b, func, [step])**

Approximates the zero of a monovariable function  $f(x)$  with the bisection method

$$f(x) = 0$$

This function needs two starting points  $[a, b]$  bracketing the zero.

Parameter "func" is a math expression string containing the symbolic function  $f(x)$

Examples of correct function definitions are:

$-2 \cdot \ln(x)$  ,  $2 \cdot \cos(x) - x$  ,  $3 \cdot x^2 - 10 \cdot \exp(-4 \cdot x)$  , etc.

The optional parameter "step" sets the maximum number of steps allowed. If omitted the function iterates still the convergence. Step = 1 is useful to study the method step-by-step

At the first step, the function returns a new segment

$$[a_1, b_1] \text{ where } a_1 < x_0 < b_1$$

At the second step, the function return a new segment

$$[a_2, b_2] \text{ where } a_1 < a_2 < x_0 < b_2 < b_1.$$

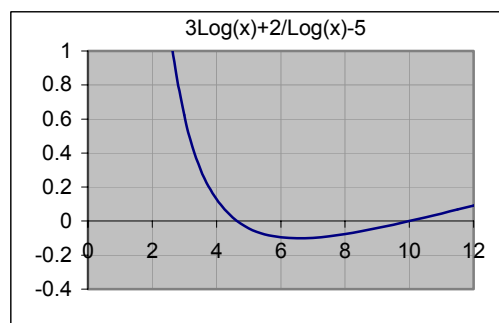
The interval  $[a_n, b_n]$ , with  $n \gg 1$ , will be very closed to the value  $x_0$

Example: Find the approximated zero of the following equation and show the first steps of the bisection method.

$$3 \cdot \log_{10}(x) + \frac{2}{\log_{10}(x)} - 5 = 0$$

The plot indicates two zeros: one trivial  $x = 10$  and another into the interval  $2 < x < 9$

Starting the algorithm with  $a = 2$  and  $b = 9$  we get  $x_0 = 4.6415888361278$



	B6		f <sub>x</sub> {=Zero_bisec(B5;C5;\$B\$2)}		
	A	B	C	D	E
1					
2	<b>f(x) =</b>	3*log(x)+2/log(x)-5			
3					
4	<b> b-a </b>	<b>a</b>	<b>b</b>		
5	7	2	9		
6	6.217E-15	4.641588834	4.641588834		
7					

The root approximates the exact zero  $x_0 = 100^{1/3}$  with error  $< 1E-14$

We can also solve this equation step-by-step in order to investigate how this algorithm works

	A	B	C	D	E
1					
2	$f(x) =$	$3 \cdot \log(x) + 2 / \log(x) - 5$			
3					
4	$ b-a $	a	b		
5	7	2	9		
6	3.5	2	5.5	{=Zero_bisec(B5;C5;\$B\$2;1)}	
7	1.75	3.75	5.5	{=Zero_bisec(B6;C6;\$B\$2;1)}	
8	0.875	4.625	5.5	{=Zero_bisec(B7;C7;\$B\$2;1)}	
9	0.4375	4.625	5.0625	{=Zero_bisec(B8;C8;\$B\$2;1)}	
10	0.21875	4.625	4.84375	{=Zero_bisec(B9;C9;\$B\$2;1)}	
11	0.109375	4.625	4.734375	{=Zero_bisec(B10;C10;\$B\$2;1)}	
12	0.0546875	4.625	4.6796875	{=Zero_bisec(B11;C11;\$B\$2;1)}	
13	0.0273438	4.625	4.6523438	{=Zero_bisec(B12;C12;\$B\$2;1)}	
14					

As we can see, the convergence is quite low but very robust because the zero always remains bracketed between the interval limits  $[a, b]$ . The error estimation is also very quick. Simply take the difference  $|b-a|$

## Secant

**=Zero\_sec(a, b, func, [step], [DgtMax])**

Approximates the zero of a monovariate function  $f(x)$  with the secant method

$$f(x) = 0$$

This function needs two starting points  $[a, b]$  bracketing the zero.

Parameter "func" is a math expression string containing the symbolic function  $f(x)$

Examples of correct function definitions are:

$-2 \cdot \ln(x)$  ,  $2 \cdot \cos(x) - x$  ,  $3 \cdot x^2 - 10 \cdot \exp(-4 \cdot x)$  , etc.

The optional parameter "step" sets the maximum number of steps allowed. If omitted the function iterates until convergence. Step = 1 is useful to study the method step-by-step

The optional parameter "DgtMax" sets the maximum number of multi-precision digits. If omitted the function works in double precision.

At the first step, the function returns a new segment

$$[a_1, b_1] \text{ where } a_1 < x_0 < b_1$$

At the second step, the function returns a new segment

$$[a_2, b_2] \text{ where } a_1 < a_2 < x_0 < b_2 < b_1.$$

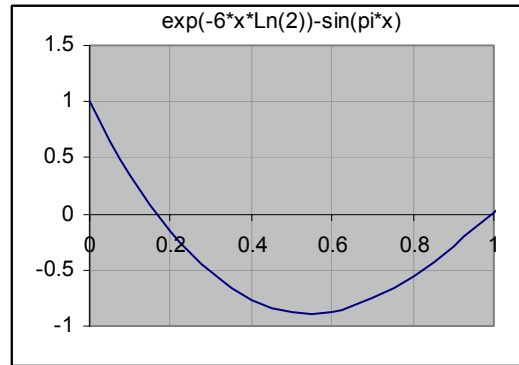
The interval  $[a_n, b_n]$ , with  $n \gg 1$ , will be very close to the value  $x_0$

Use the CTRL+SHIFT+ENTER sequence to paste this function

Example: Find the approximated zero of the following equation and show the first steps of the secant method.

$$\exp(-3x \cdot \ln(2)) - \sin(\pi \cdot x) = 0$$

The plot indicates one zero in the interval  $0 < x < 0.5$



Starting the algorithm with  $a = 0$  and  $b = 0.5$  we get  $x_0 = 0.166666666666667$

B6		fx {=Zero_sec(B5;C5;\$B\$2)}			
	A	B	C	D	
1					
2	<b>f(x) =</b>	exp(-6x*ln(2))-sin(pi*x)			
3					
4	<b> b-a </b>	<b>a</b>	<b>b</b>		
5	0.5	0	0.5		
6	0	0.166666667	0.166666667		
7					

The root approximates the exact zero  $x_0 = 1/6$  with error  $< 1E-15$

Let's see now the iteration trace setting the parameter step = 1

	A	B	C	D	E	F
1						
2	<b>f(x) =</b>	exp(-6x*ln(2))-sin(pi*x)				
3						
4	<b> b-a </b>	<b>a</b>	<b>b</b>			
5	0.5	0	0.5			
6	0.2333333	0.5	0.266666667	{Zero_sec(B5;C5;\$B\$2;1)}		
7	0.2088422	0.266666667	0.057824454	{Zero_sec(B6;C6;\$B\$2;1)}		
8	0.1241313	0.057824454	0.181955763	{Zero_sec(B7;C7;\$B\$2;1)}		
9	0.0131587	0.181955763	0.168797096	{Zero_sec(B8;C8;\$B\$2;1)}		
10	0.0021775	0.168797096	0.166619598	{Zero_sec(B9;C9;\$B\$2;1)}		
11	4.721E-05	0.166619598	0.166666809	{Zero_sec(B10;C10;\$B\$2;1)}		
12	1.422E-07	0.166666809	0.166666667	{Zero_sec(B11;C11;\$B\$2;1)}		
13	9.471E-12	0.166666667	0.166666667	{Zero_sec(B12;C12;\$B\$2;1)}		
14	0	0.166666667	0.166666667	{Zero_sec(B13;C13;\$B\$2;1)}		
15						

As we can see the convergence of this method is much faster than the one of the bisection method. On the other hand, it is not guaranteed that the zero remains bracketed into the interval.

## Derivatives

### First Derivative

**=Diff1(x, fx, [lim])**

Approximates the first derivative of a mono-variable function  $f(x)$  at the given point  $x$

$$f'(x) = \frac{d}{dx} f(x)$$

The parameter "Fx" is a math expression string containing the symbolic function  $f(x)$   
Examples of function definition are:

$-2*\ln(x)$  ,  $2*\cos(x)$  ,  $3*x^2-10*\exp(-4*x)$  ,  $x^2+4*x+1$  , etc.

The optional parameter "Lim" (default = 0) sets the way how the limit approach to  $x$ . If  $\lim = 1$ , it approaches from the right; if  $\lim = -1$ , it approaches from the left; if  $\lim = 0$ , it approaches centrally. That is, it returns the following derivatives

$$f'(x) = \begin{cases} \lim_{h \rightarrow 0^-} f(x) = f'(x^-) \\ \lim_{h \rightarrow 0} f(x) = f'(x) \\ \lim_{h \rightarrow 0^+} f(x) = f'(x^+) \end{cases}$$

This function uses the following formulas to approximate each derivative

$$f'(x^-) \cong \frac{1}{12h} (25f(x) - 48f(x-h) + 36f(x-2h) - 16f(x-3h) + 3f(x-4h))$$

$$f'(x) \cong \frac{1}{12h} (f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h))$$

$$f'(x^+) \cong \frac{1}{12h} (25f(x) - 48f(x+h) + 36f(x+2h) - 16f(x+3h) + 3f(x+4h))$$

Example. Evaluate numerically the left, right and central derivatives of the given function at the point  $x = 0$ , and check if the given function is differentiable in that point

$$f(x) = \frac{x}{x^2 + |x| + 1}$$

	A	B	C	D	E	F
1	x	0		y'(x-)	y'(x)	y'(x+)
2	y(x) =	x / (x^2 +  x  + 1)		1	1	1
3						
4		=diff1(B1;B2;-1)	=diff1(B1;B2)	=diff1(B1;B2;1)		

As we can see all derivatives are equal, so the function is differentiable in  $x = 0$

## Second Derivative

### =Diff2(x, fx)

It approximates the second derivative of mono-variable function  $f(x)$  at the given point

$$f''(x) = \frac{d^2}{dx^2} f(x)$$

The parameter "Fx" is a math expression string containing the symbolic function  $f(x)$   
Examples of function definition are:

$-2*\ln(x)$  ,  $2*\cos(x)$  ,  $3*x^2-10*\exp(-4*x)$  ,  $x^2+4*x+1$  , etc.

Example: Evaluate the first and second derivatives at the point  $x = 2$  for the following function

$$f(x) = \frac{x+3}{x^2+1}$$

	A	B	C	D
1	x	2		
2	y(x) =	(x+3)/(x^2+1)	=diff1(B1;B2)	
3				
4	y'(x) =	-0.6	=diff2(B1;B2)	
5	y''(x) =	0.56		
6				

## Gradient

### =Grad(p, func)

Approximates the gradient of a multivariate function  $f(x, y, z)$  at the given point

$$\nabla f(x, y, z) = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right]$$

The parameter "p" is the vector of the variables  $[x, y, z]$

The parameter "Func" is an expression string containing the function  $f(x, y, z)$ .

Examples of function definition are:

$-2*\ln(x+3y)$  ,  $2*\exp(-x)*\cos(3*t)$  ,  $3*x^2-y^2+z^2$  ,  $(x^2+y^2)^{(1/3)}$  , etc.

For performance problem, the number of variables is restricted to 4, "x", "y", "z", "t".

The variables values must be always passed in this order.

Example. Evaluate the gradient of the following function at the point  $P(1, 1)$

$$f(x, y) = \frac{1}{x^2 + 5y^2}$$

	A	B	C	D
1	x =	1		
2	y =	1		
3	f(x,y) =	1/(x^2+5*y^2)	{=Grad(B1:B2;B3)}	
4				
5	f'x(x, y) =	-0.055555556		
6	f'y(x, y) =	-0.277777778		
7				

## Jacobian matrix

### =Jacobian (p, func)

Approximates the Jacobian's matrix of a multivariate vector-function  $\mathbf{F}(x, y, z)$  at the given point  $p(x, y, z)$

$$F(x, y, z) = \begin{bmatrix} f_1(x, y, z) \\ f_2(x, y, z) \\ f_3(x, y, z) \end{bmatrix} \quad J(x, y, z) = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial z} \\ \frac{\partial f_3}{\partial x} & \frac{\partial f_3}{\partial y} & \frac{\partial f_3}{\partial z} \end{bmatrix}$$

The parameter "p" is the vector of the variables [x, y, z]

The parameter "Func" is an expression string containing the function  $f(x, y, z)$ .

Examples of function definition are:

$-2 \cdot \ln(x+3y)$ ,  $2 \cdot \exp(-x) \cdot \cos(3 \cdot t)$ ,  $3 \cdot x^2 - y^2 + z^2$ ,  $(x^2 + y^2)^{(1/3)}$ , etc.

For performance problem, the number of variables is restricted to 4, "x", "y", "z", "t".  
The variables values must be always passed in this order.

Example. Evaluate the Jacobian's matrix of the following vector-function at the point  $P(1, 1)$

$$f_1(x, y, z) = \frac{1}{x^2 + 5y^2 + z^2} \quad f_2(x, y, z) = z \cdot \ln(x + 2y) \quad f_3(x, y, z) = 4xyz$$

	A	B	C	D	E	F	G
1	x=	0.5		{=Jacobian(B1:B3;B4:B6)}			
2	y=	2					
3	z=	1					
4	f1(x,y,z)=	1/(x^2+5*y^2+z^2)		-0.0022145	-0.0442907	-0.0044291	
5	f2(x,y,z)=	z*ln(x+2y)	J(x,y,z)=	0.2222222	0.4444444	1.5040774	
6	f3(x,y,z)=	4*x*y*z		8	2	4	
7							

## Hessian matrix

### =Hessian (p, func)

Approximates the Hessian' matrix of a multivariate function  $f(x, y)$  at the given point  $p(x, y)$

$$H(x, y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

The parameter "p" is the vector of the variables [x, y]

The parameter "Func" is an expression string containing the function  $f(x, y, z)$ .

Examples of function definition are:

## Xnumbers Tutorial

$-2*\ln(x+3y)$ ,  $2*\exp(-x)*\cos(3*t)$ ,  $3*x^2-y^2+z^2$ ,  $(x^2+y^2)^{(1/3)}$ , etc.

For performance problem, the number of variables is restricted to 4, "x", "y", "z", "t".

The variables values must be always passed in this order.

This function returns a square a matrix (n x n) of the second derivatives

Note: the derivatives approximation is about to 1E-10

Example. Approx. the Hessian's matrix of the following function at the point (2,1,1)

$$f(x,y,z) = \frac{1}{x^2 + 5y^2 + z^2}$$

	A	B	C	D	E	F
1	x =	2				
2	y =	1		0.012	0.08	0.016
3	z =	1	H(x,y,z) =	0.08	0.1	0.04
4	f(x,y,z) =	1/(x^2+5*y^2+z^2)		0.016	0.04	-0.012
5						
6		{=hessian(B1:B3;B4)}				
7						



216

## Xnumbers Tutorial

This excellent result has been obtained in spite of the approximated precision ( $1e-13$ ) of the derivative. The reason is simple: the accuracy of the derivative does not influence the final accuracy of the root. We note that the derivative, after very few iterations, remains constant: we might substitute this value with an even more approximated values, i.e.  $f' = 1.57$ , for all iterations. The final accuracy will not change. We will need only more few steps, at the most.

But this method show its power overall for non-linear systems. For a 2 variables problem the Newton-Raphson method becomes

$$\begin{cases} f(x, y) = 0 \\ g(x, y) = 0 \end{cases} \Rightarrow \begin{pmatrix} x \\ y \end{pmatrix}_{n+1} = \begin{pmatrix} x \\ y \end{pmatrix}_{n+1} - \begin{bmatrix} f_x & f_y \\ g_x & g_y \end{bmatrix}_n^{-1} \begin{pmatrix} f \\ g \end{pmatrix}_n$$

The (2 x 2) matrix is the Jacobian calculated at the point  $(x_n, y_n)$ . In Xnumbers it can be evaluated by the function **Jacobian**

Example. Solve the following system

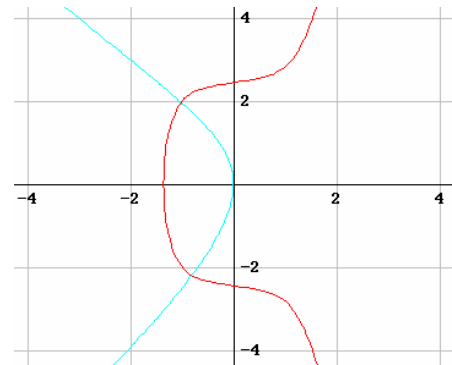
$$\begin{cases} -x^5 + y^2 - x - 6 = 0 \\ e^{x-y} + x + y - 1 = 0 \end{cases}$$

setting:

$$f(x, y) = -x^5 + y^2 - x - 6$$

$$g(x, y) = e^{x-y} + x + y - 1$$

the contour plots of the functions  $f = 0$  and  $g = 0$  show two intersection points: one near the point  $(-1, 2)$  and one near  $(-1, -2)$



	A	B	C	D	E
3					
4	$f(x,y) =$	$-x^5 + y^2 - x - 6$			
5	$g(x,y) =$	$\exp(x-y) + x + y - 1$			$\{=B7:B8+E10:E11\}$
6					
7	$x0$	-1.019648		$x$	-1.019648
8	$y0$	1.9693081		$y$	1.9693081
9					
10	$f$	3.967E-15		$dx$	4.413E-15
11	$g$	-1.05E-14		$dy$	6.169E-15
12					
13	Jacobian		$\{=MMULT(MINVERSE(A14:B15),-B10:B11)\}$		
14		-6.404695	3.9386162		
15		1.05034	0.94966		$\{=Jacobian(B7:B8,B4:B5)\}$

The function  $f(x,y)$  and  $g(x,y)$  are evaluated and converted in double precision by the nested functions

$=xcdbl(xeval(B4,B7:B8))$

$=xcdbl(xeval(B5,B7:B8))$

At the begin insert the starting point  $(-1, 2)$  in the cells B7, B5.

The new point is calculated in the cells E7:E8. Copy this range and re-insert in the range B7:B8. At each iteration the increments  $dx, dy$  of the range E10:E11 becomes more and more small.

Starting from  $(-1, 2)$  and  $(-1, -2)$  the iteration algorithm leads to the correct solutions

x	y
-1	2
-1.0201151219	1.9698273171
-1.0196483063	1.9693084022
-1.0196480758	1.9693081215
-1.0196480758	1.9693081215

x	y
-1	-2
-0.7964138633	-2.3053792051
-0.8079928505	-2.2042117521
-0.8107932120	-2.1997452584
-0.8108021826	-2.1997248438

## Conversions

### Decibel

**=dBel(A, [MinLevel])**

Converts a positive number A into decibel

$$A_{\text{dB}} = 20 \log_{10}(A)$$

If zero, A is substituted with the value contained in the parameter "MinLevel" (default 1E-15)

Example

A	A dB
1	0
0.5	-6.0206
0.1	-20
0.05	-26.021
0.01	-40
0.001	-60
0.0001	-80
0	-300

### Base conversion

**cvDecBin(DecNum)**

base 10  $\Rightarrow$  base 2

**cvBinDec(BinNum)**

base 2  $\Rightarrow$  base 10

**cvDecBase(DecNum, Base)**

base 10  $\Rightarrow$  any base (2-16)

**cvBaseDec(BaseNum, Base)**

any base (2-16)  $\Rightarrow$  base 10

**baseChange(number, old\_base, new\_base)**

any base (2 - 36)  $\Rightarrow$  any base (2 - 36)

These functions perform the number conversion between different bases.

Example: Converts the decimal number  $n = 902023485$  into bases 2 and 3.

`cvDecBin(902023485) = 110101110000111100100100111101` (base 2)

`cvDecBase(902023485, 3) = 2022212022112121020` (base 3)

Example: Converts the hexadecimal number  $n = 35CFFF3D$  into decimal

`cvBaseDec(35CFFF3D) = 902823741` (base 10)

You can also convert directly base-to-base, nesting two functions.

Example convert  $n = 35CFFF3D$  from base 16 into 8

`cvDecBase(cvBaseDec(35CFFF3D, 16), 8) = 6563777475` (base 8)

For this scope you can also use the `baseChange` function<sup>18</sup>

In spite of its digits limitation (15), this function has several interesting features. It converts any number into many different bases (up to 36). The digits greater than 9 are indicated as A, B, C, D, E, F, G, H, etc. It converts also decimal numbers. It formats the result consistently with the source cell. Let's see how it works

<sup>18</sup> [The function baseChange appears thanks to the courtesy of Richard Huxtable](#)

	A	B	C	D
1	<b>Examples using the changeBase function</b>			
2	<b>Convert a number into many different bases</b>			
3				
4	old base =>	10	14.2000000	14.20
5				
6	new bases =>	25	E.5000000	E.50
7	new bases =>	24	E.4J4J4J4	E.4J
8	new bases =>	23	E.4DI94DI	E.4D
9	new bases =>	18	E.3AE73AE	E.3A
10	new bases =>	16	E.3333333	E.33
11	new bases =>	15	E.3000000	E.30
12				
13	=baseChange(C4;B4;B11)			
14				

The cell C4 is formatted with 7 digits and also its results have the same format; the cell D4 is formatted with 2 decimals and its result has the same format.

## Log Relative Error

= mjkLRE(q, c, NoSD)

= xLRE(q, c, NoSD, [DgtMax])

This function<sup>19</sup> returns the log relative error (LRE) for an estimated value ( $q$ ) and a certified value ( $c$ ), which has a specified number of significant digits (NoSD). The LRE is a measure of the number of correct significant digits only when the estimated value is “close” to the exact value. Therefore, each estimated quantity must be compared to its certified value to make sure that they differ by a factor of less than two, otherwise the LRE for the estimated quantity is zero.

### Definition

The base-10 logarithm of the relative error is defined as:

$$\text{if } c = 0 \begin{cases} \text{LRE} = 0 & \text{if } |q| > 1 \\ \text{LRE} = \min(-\log(q), \text{NoSD}) & |q| < 1 \end{cases}$$

$$\text{if } c \neq 0 \begin{cases} \text{LRE} = \min(-\log(|q - c| / |c|), \text{NoSD}) & \text{if } c \neq q \text{ and } 1/2 \leq |q/c| \leq 2 \\ \text{LRE} = \text{NoSD} & \text{if } c = q \\ \text{LRE} = 0 & \text{if else} \end{cases}$$

Example:

Assume that you want to compare an approximate value with a 15 digits certified value of pi-Greek. LRE metric can show this in a easy way

Certified value C = 3.14159265358979

Approx. value Q = 3.14159265300001

mjkLRE(C, Q, 15) = 9.7

<sup>19</sup> These functions appear by courtesy of Michael J. Kozluk. This algorithm was first programmed into an Excel user function, by Michael, in standard 32 bit precision. As it works fine also for comparing long extended numbers (NoSD > 15), we have now developed its multiprecision version xLRE().

## Xnumbers Tutorial

This means that two values are close for about 10 significant digits. LRE metric rejects non significant digits. Look at this example:

Certified value  $C = 0.000133333333333333$

Approx. value  $Q = 0.000133333333333311$

$\text{mjkLRE}(C, Q, 15) = 12.8$

As we see, the two numbers appear exact up to the 17<sup>th</sup> digit, but the relative error is about  $1\text{E-}13$

LRE is very useful when you work with long string of extended numbers. For example, compare this approximation of "e" (Napier's number)

Certified value  $C = 2.71828182845904523536028747111$

Approx. value  $Q = 2.71828182845904523536028747135$

$\text{xLRE}(C, Q, 30) = 28.1$

At the first sight it is hard to say, but the LRE function shows immediately a precision of about 28 digits

## Special Functions

The computation of special functions is a fundamental aspect of numerical analysis in virtually all areas of engineering and the physical sciences.

All these special functions have a high-fixed-precision. Because most of these special functions are in the form of infinite series or infinite integrals, their solutions are quite complicated, and we have spent many times for selecting and testing many different algorithms in order to achieve the highest possible accuracy in 32 bit arithmetic.

### Error Function Erf(x)

---

#### erfun(x)

Returns the error function

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

Accuracy: about  $10^{-14}$  per  $x > 0$

### Exponential integral Ei(x)

---

#### exp\_integr(x)

Returns the exponential integral

$$Ei(x) = - \int_{-x}^{\infty} \frac{e^{-t}}{t} dt$$

Accuracy: about  $10^{-14}$  for  $x > 0$

### Exponential integral En(x)

---

#### exp\_integr\_n(x, n)

Returns the exponential integral of n-th order

$$En(x) = - \int_1^{\infty} \frac{e^{-xt}}{t^n} dt$$

Accuracy: about  $10^{-14}$  for  $x > 0$  and  $n > 0$

### Euler-Mascheroni Constant $\gamma$

---

#### xGm([Digit\_Max])

Returns the Euler-Mascheroni gamma constant.

The optional parameter Digit\_Max sets the maximum digits (default 30, max 415)

$$\gamma = \lim_{n \rightarrow \infty} \left( 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} - \log(n) \right)$$

Example: compute the gamma constant with 40 significant digits

xGm(40) = 0.5772156649015328606084804798767149086546

## Gamma function $\Gamma(x)$

---

### xGamma(x)

Returns the gamma function.

$$\Gamma(x) = \int_0^{\infty} t^{x-1} e^{-t} dt$$

This routine uses an excellent Lanczos series approximation<sup>20</sup>

$$\Gamma(x) \cong \frac{\sqrt{2\pi}}{e^g} \left( \frac{x+g+\frac{1}{2}}{e} \right)^{x+\frac{1}{2}} \cdot \left( c_0 + \sum_{i=1}^{14} \frac{c_i}{x+i} \right)$$

where: g = 607/128 and c<sub>i</sub> are the Lanczos' coefficients.

Relative accuracy is better than 10<sup>-14</sup>, except very near to the poles x=0, -1, -2, -3...

This function works also with large argument because it uses the multiprecision format to avoid the overflow for arguments greater than 170.

Example,

x	xgamma(x)	Rel. Error
0.001	9.99423772484596E+2	1.02E-15
0.01	9.94325851191507E+1	1.00E-15
0.1	9.51350769866874	9.33E-16
1	1	0
10	3.6288E+5	0
100	9.33262154439441E+155	5.64E-16
1,000	4.02387260077093E+2564	1.92E-15
10,000	2.84625968091705E+35655	1.58E-15
100,000	2.82422940796034E+456568	2.75E-15
1,000,000	8.26393168833122E+5565702	2.54E-15

Note that relative accuracy is better than 5\*10<sup>-15</sup> in any case

You can convert in double only the values with x ≤ 170, otherwise you will get #VALUE! (error). You can manipulate these large values only by the "x-functions", or, separating mantissa and exponent (see xsplit())

FACTORIAL: Thanks to its effcience and accuracy, this function can also be used to calculate the factorial of a big integer number, using the relation

$$n! = \Gamma(n+1)$$

---

<sup>20</sup> This accurate algorithm has been extracted from a very good note by Paul Godfrey, Intersil , C.2001

Example:

<code>xfact(10002) =</code>	<code>2.84711361574652325360317551421E+35667</code>	30 digits, slower
<code>xgamma(10003) =</code>	<code>2.84711361574651E+35667</code>	15 digits, faster

## Log Gamma function

---

### **xGammaln(x)** **xGammalog(x)**

These function return the natural and decimal logarithm of the gamma function.

```
xgammaln(100000) = 1051287.7089736568948
xgammalog(100000) = 456568.45089997090835
```

Relative accuracy is better than  $10^{-(14+|\log(x)|)}$  for  $x>0$

These functions are added only for compatibility with Excel and other math packages. In fact they are useful to avoid overflow in standard precision arithmetic for large arguments of gamma function. However if you use directly the `xgamma()` and multiprecision arithmetic, you need no more to use these functions.

## Gamma quotient

---

### **xGammaQ(x1, x2)**

Performs the division of two gamma functions.

$$q = \Gamma(x_1) / \Gamma(x_2)$$

Relative accuracy is better than  $10^{-14}$ , for  $x_1>0$  and  $x_2>0$

Example: suppose you have to calculate for  $v=1,000,000$  the following quotient

$$q = \frac{\Gamma(\frac{v+1}{2})}{\Gamma(\frac{v}{2})}$$

Taking  $x_1 = 500,000.5$  and  $x_2 = 500,000$ , we have easily

```
xgammaq(500000.5, 500000) = 707.106604409874 (rel error = 5.96E-16)
```

Note that if you have used the standard `GAMMALN()` function, you should have:

```
EXP(GAMMALEN(500000.5) - GAMMALEN(500000)) = 707.106604681849
(rel error = 3.846E-10)
```

As we can see, In this case, the error is more than 500,000 times bigger than the previous one!



## Gamma F-factor

---

### xGammaF(x1, x2)

Returns the gamma factor of the Fischer distribution.

$$k = \frac{\Gamma\left(\frac{x_1 + x_2}{2}\right)}{\Gamma\left(\frac{x_1}{2}\right) \cdot \Gamma\left(\frac{x_2}{2}\right)}$$

Relative accuracy is better than  $10^{-14}$ , for  $x_1 > 0$  and  $x_2 > 0$

## Digamma function

---

### digamma(x)

Returns the logarithmic derivative of the gamma function

$$\Psi(x) = \frac{d}{dx} \ln(\Gamma(x)) = \frac{\Gamma'(x)}{\Gamma(x)}$$

Relative accuracy is better than  $10^{-14}$ , for  $x > 0$

Example

digamma(x)	value	rel. error
0.01	-100.560885457869	3.24E-15
0.1	-10.4237549404111	2.23E-15
1	-0.577215664901532	1.49E-15
10	2.25175258906672	4.92E-16
100	4.60016185273809	5.65E-16
1000	6.90725519564881	2.97E-16

Note that  $\Psi(1) = -\gamma$  (Eulero- constant)

## Beta function

---

### xbeta(x, y)

Returns the beta function

$$B(x, y) = \int_0^1 t^{x-1} (1-t)^{y-1} dt$$

Relative accuracy is better than  $10^{-14}$ , for  $x > 0$  and  $y > 0$

## Combinations function

---

### **xcomb\_big(n, k)**

Returns the combination, or binomial coefficients, for large integer numbers

$$C_{n,k} = \binom{n}{k} = \frac{n!}{(n-k)!k!}$$

Relative accuracy is better than  $10^{-14}$ , for  $n \gg 0$  and  $k \gg 0$

This function uses the gamma function to calculate the factorials. It is much faster than xcomb function. For this reason is adapted for large integer values (10,000 - 1,000,000)

xcomb(5000,2493) =	1.5627920156854189438574778889E+1503	(30 digits, slow)
xcomb_big(5000,2493) =	1.56279201568542E+1503	(15 digits , fast)

## Bessel functions

---

<b>BesselJ (x, [n])</b>	Bessel function of 1° kind, order n: $J_n(x)$
<b>BesselY (x, [n])</b>	Bessel function of 2° kind, order n: $Y_n(x)$
<b>BesseldJ (x, [n])</b>	First derivative of Bessel functions of 1° kind, order n: $J'_n(x)$
<b>BesseldY (x, [n])</b>	First derivative of Bessel functions of 2° kind, order n: $Y'_n(x)$
<b>Bessell (x, [n])</b>	Modified Bessel function of 1° kind, order n: $I_n(x)$
<b>BesselK (x, [n])</b>	Modified Bessel function of 2° kind, order n: $K_n(x)$
<b>BesseldI (x, [n])</b>	First derivative of mod. Bessel functions of 1° kind, order n: $I'_n(x)$
<b>BesseldK (x, [n])</b>	First derivative of mod. Bessel functions of 2° kind, order n: $K'_n(x)$

Relative accuracy is better than  $10^{-13}$ , for  $x > 0$  and n any integer

These routines<sup>21</sup> have a high general accuracy. Look at the following example. We have compared results obtained from our BesselJ with the standard Excel similar function

x	J0(x) (BesselJ)	Rel. Error	J0(x) (Excel standard)	Rel. Error
0.1	0.997501562066040	1.11E-16	0.997501564770017	2.71E-09
0.5	0.938469807240813	1.06E-15	0.938469807423541	1.95E-10
1	0.765197686557967	7.25E-16	0.765197683754859	3.66E-09
5	-0.177596771314338	2.66E-15	-0.177596774112343	1.58E-08
10	-0.245935764451374	1.06E-13	-0.245935764384446	2.72E-10
50	0.055812327669252	3.98E-15	0.055812327598901	1.26E-09

As we can see, the general accuracy improving is more than 200,000 times!

## Cosine Integral Ci(x)

---

### CosIntegral(x)

Returns the Cosine integral defined as:

$$ci(x) = -\int_x^{\infty} \frac{\cos(t)}{t} dt$$

Relative accuracy is better than  $10^{-13}$ , for  $x > 0$

---

<sup>21</sup> All these special functions are provided thanks to the FORTRAN 77 Routines Library for Computation of Special Functions developed by Shanjie Zhang and Jianming Jin . The programs and subroutines contained in this library are copyrighted. However, authors kindly gave permission to the user to incorporate any of these routines into his programs.

## Sine Integral Si(x)

---

### SinIntegral(x)

Returns the sine integral defined as:

$$\text{si}(x) = \int_0^x \frac{\sin(t)}{t} dt$$

Relative accuracy is better than  $10^{-13}$ , for  $x > 0$

## Fresnel sine Integral

---

### Fresnel\_sin(x)

Returns the Fresnel's sine integral defined as:

$$S(x) = \int_0^x \sin\left(\frac{1}{2} \pi t^2\right) dt$$

Relative accuracy is better than  $10^{-13}$ , for  $x > 0$

Remember also the following relation

$$k \cdot \int_0^x \sin(t^2) dt = S(k \cdot z) \quad \text{where:} \quad k = \sqrt{\frac{2}{\pi}}$$

## Fresnel cosine Integral

---

### Fresnel\_cos(x)

Returns the Fresnel's cosine integral defined as:

$$C(x) = \int_0^x \cos\left(\frac{1}{2} \pi t^2\right) dt$$

Relative accuracy is better than  $10^{-13}$ , for  $x > 0$

Remember also the following relation

$$k \cdot \int_0^x \cos(t^2) dt = C(k \cdot z) \quad \text{where:} \quad k = \sqrt{\frac{2}{\pi}}$$

## Fibonacci numbers

---

### xFib(n, [DgtMax])

Returns the Fibonacci's numbers defined by the following recurrent formula:

$$F_1 = 1, \quad F_2 = 2, \quad F_n = F_{n-1} + F_{n-2}$$

Example:

xFib(136) = 11825896447871834976429068427

xFib(4000) = 3.99094734350044227920812480949E+835

## Hypergeometric function

---

### Hypergeom(a, b, c, x)

Returns the Hypergeometric function

The parameter "a" is real, "b" is real, "c" is real and different from 0, -1, -2, -3 ...

The variable "x" is real with  $|x| < 1$

Relative accuracy is better than  $10^{-14}$ , for  $-1 < x < 1$

The hypergeometric function is the solution of the so called *Gaussian-hypergeometric differential equation*

$$x(1-x)y'' + (c - (a+b+1)x)y' + ab y = 0$$

An integral form of the hypergeometric function is

$$F(a, b, c, x) = \frac{\Gamma(c)}{\Gamma(b)\Gamma(c-b)} \int_0^1 t^{b-1} (1-t)^{c-b-1} (1-tx)^{-a} dt$$

More known is the series expansion that converges for  $|x| < 1$

$$F(a, b, c, x) = 1 + \frac{ab}{c} \frac{x}{1!} + \frac{a(a+1)b(b+1)}{c(c+1)} \frac{x^2}{2!} + \frac{a(a+1)(a+2)b(b+1)(b+2)}{c(c+1)(c+2)} \frac{x^3}{6!} + \dots$$

Special result are:

$$F(p, 1, 1, x) = (1-x)^{-p}$$

$$F(1, 1, 2, -x) = \frac{\ln(1+x)}{x}$$

$$F\left(\frac{1}{3}, \frac{2}{3}, \frac{5}{6}, \frac{27}{32}\right) = 1.6$$

## Zeta function $\zeta(s)$

---

### Zeta(s)

The Riemann zeta function  $\zeta(s)$  is an important special function of mathematics and physics which is intimately related with very deep results surrounding the prime number, series, integrals, etc.

Relative accuracy is better than  $1E-14$ , for any  $s \neq 1$

For  $|s| > 1$  the function is defined

$$\zeta(n) = \sum_{k=1}^{\infty} \frac{1}{k^n}.$$

Analytic continuation. The Riemann zeta function can be defined for  $0 < s < 1$  by the following analytic continuation:

$$\zeta(s) = \frac{1}{1 - 2^{1-s}} \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{n^s}.$$

For  $s < 0$  the function is defined by the following relation:

$$\zeta(1-s) = 2(2\pi)^{-s} \cos\left(\frac{1}{2}s\pi\right) \Gamma(s) \zeta(s)$$

Some known exact results are:  $\zeta(2) = \pi^2/6$  ,  $\zeta(4) = \pi^4/90$

Zeta function is very useful in computing series. Look at this example:

$$\sum_{k=0}^{\infty} \frac{1}{(k+2)^2} = \sum_{k=2}^{\infty} \frac{1}{k^2} = \sum_{k=1}^{\infty} \frac{1}{k^2} - 1 - \frac{1}{2^2} = \zeta(2) - \frac{5}{4}$$

So, the final result is  $\pi^2/6 - 5/4$

## Formulas Evaluation

### Multiprecision Expression Evaluation

These functions realize a little math shell, putting together the power of multiprecision numeric computation with the ease of symbolic calculus. Sometime we may want to perform the computation using symbolic formulas.

We would pass these strings to a routine for evaluation, returning the numerical results with a given accuracy. These functions perform this useful task.

**xeval( Formula, [Var], [DgtMax], [Angle],)**

**xevall( Formula, [Var1, Var2 ...] )**

These functions return the evaluation of a math expression in multiprecision arithmetic. They use the same algorithm<sup>22</sup> and have the same variable accuracy. They differ only for the input parameters.

The parameter "Formula" is a math expression string containing variables, operators, parenthesis and other basic functions. Examples.

```
3+1/(x^2+y^2), sin(2*pi*t)+4*cos(2*pi*t), (x^4+2x^3+6x^2-12x-10)^(1/2)
```

The optional parameter "Var" is an array containing one or more value for variables substitution. Before computing, the parser substitutes each symbolic variable with its correspondent value. It can be a single value, an array of values or, even an array of values + labels (see examples).

The optional parameter "Var1", "Var2"... are single values or array as "Var" but without labels, because the function **xevall** automatic finds by itself the appropriate labels. (See example)

The optional parameter "DgtMax" – from 1 to 200 - sets the maximum number of precision digits (default=30). Setting DgtMax = 0 will force the function to evaluate in faster standard precision.

The optional parameter "Angle" sets the angle unit "RAD" (default) "DEG", "GRAD".of for trigonometric computation:

Example:

```
xeval("(1+sqr(2))/2+5^(1/3)") = 2.91708272786324451375395323463
xeval("(1+cos(x))/2+x^y" , {5, 1.2}) = 7.5404794000376872941836369067
xeval("(a+b)*(a-b)", {2, 3}) = (2+3)*(2-3) = -5
```

All the function parameters can also be passed by reference of cell

Example. Tabulate the following function for x = 1, 1.5, 2, ... with 30 significant digits

<sup>22</sup> The algorithm is divided into two steps: parsing and evaluation. The first step is performed by the MathParser class. The evaluation is performed with the x-functions of XNUMBERS.

## Xnumbers Tutorial

$$f(x) = \frac{1+x}{\sqrt{1+x^2}}$$

	A	B	C	D
1				
2	<b>f(x) =</b>	(1+x) / sqr(1+x^2)		
3				
4	<b>x</b>	<b>f(x)</b>		
5	1	1.41421356237309504880168872421	=xeval(\$B\$2;A5)	
6	1.5	1.38675049056307280504585433364	=xeval(\$B\$2;A6)	
7	2	1.34164078649987381784550420123	=xeval(\$B\$2;A7)	
8				

Note how the use of this function is simple and straight comparing with the nested formulas

```
=xdiv(xadd(1,A6),xsqr(xadd(1,xpow(A6,2))))
```

Calculating functions with more than one variable a bit complication arises, because we have to pay attention which values are assigned to the variables. Let's see this example

Calculate the following bivariate function for x = 2.4, y = 5.5

$$f(x, y) = \frac{\ln(y) + xy}{\sqrt{1+x^2}}$$

In order to pass to the parameter "Var" the correct value for each variable we select the variables range B2:C3 including the labels "x" and "y" (header). The labels must contain the same symbols contained into the formula string

	A	B	C	D
1				
2	<b>function</b>	<b>x</b>	<b>y</b>	<b>result</b>
3	(ln(y)+x*y) / sqr(y)	2.4	5.5	6.35540594073030048985687628091
4				=xeval(A3;B2:C3)
5				
6				
7				

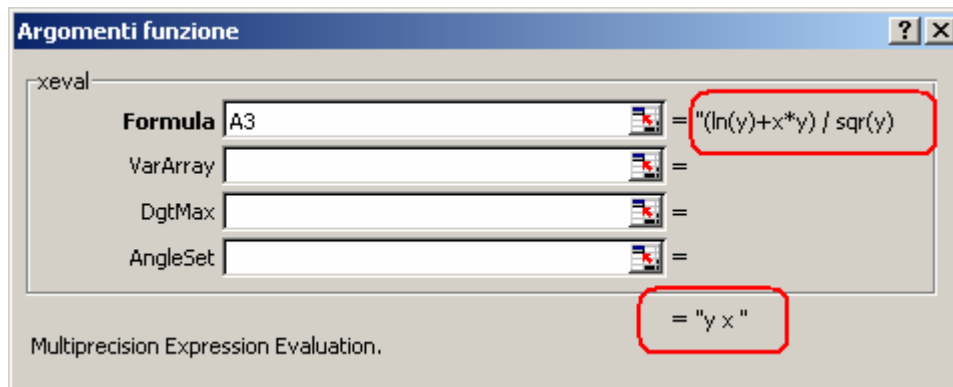
variables range with header B2:C3

Note If we pass the range B3:C3 without the labels, the function assigns the values to the variables in the same order that they appear in the formula, from left to right. In our example the first variables is "y" and the second is "x", so the function assigns the first value 2.4 to "y" and the second value 5.5 to "x"

To by-pass the variable order rule, the function uses the trick of the "variables labels". On the contrary, for one or none variable it is impossible to make confusion so the header can be omitted.

Variables order. The function returns the variables order in the Excel function insertion panel





In our example we see the string “y x”, that means you have to pass the first value for the variable “y” and the second value for the variables “x”

The variables order is by-passed by labels. Using labels you do not need to worry about the variables order

Let's see another example. Calculate with 30 digits precision, the following function

$$f(x, a, b) = a \sin(x) - b \cos(2x)$$

for  $x = 30^\circ$  deg ,  $a = 1$  ,  $b = -2$

[illegible]

Note that we have set the angle unit = “DEG”

Note also that in this case the variable order would be “a, x, b”, but with the aid of the labels the function can associate the exact parameters in the right way.

Sometime it is not possible to add a label near its value (in the middle of a table, for example). Neither all parameters are located adjacent each others. For these cases we can use the second evaluator function **xevall**

This function accepts separate parameters or separate array. We do not need to pass variables labels. The function automatically finds all labels present on the active worksheet

Of course all this has a cost. the function **xevall** is about 10 times slower then **xeval**.

Let's see how it works with an example. Tabulate the given trigonometric function, from  $t = 0$  to  $0.5$ , with step =  $0.1$  and an error less than  $1E-20$

$$f(t) = a \cdot \cos(\pi \cdot t) + b \cdot \cos(\pi \cdot t)$$

where  $a = 0.5$  and  $b = -2$

B8		=xeval(\$B\$1;A8;\$C\$4:\$D\$4;\$C\$7:\$D\$7)			
	A	B	C	D	E
1	f(t) = a*cos(pi*t)+b*sin(pi*t)				
2					
3	t	f(t)	a	b	
4	0.1	-0.142505730602318062146	0.5	-2	
5	0.2	-0.771062007397472546279			
6	0.3	-1.32414136260365828361	DgtMax	AngleSet	
7	0.4	-1.74760453540283343216	21	RAD	
8	0.5	-2			

The above sheet shows a possible arrangement. If we look the last cell B8 we discover that the parameters are:

Var1 the cell A8 containing the value of the independent variable “t”

Var2 the range “C4:D4”, containing the values of the parameters “a” and “b”

Var3 range “C7:D7”, the the internal parameter “DgtMax” and “AngleSet”

The internal “DgtMax” and “AngleSet” parameters are reserved word and must write as is.

Note also that the cell A8 has no label, but the function performs the correct assignment to the “t” variable.

**Label Rules.** Labels must stay always at the top or at the left of the corresponding values. Labels can have any alphanumeric name starting with any letter and not containing blank. In the example:

t = 0.1, a = 0.5 , DgtMax = 30

t		a	
0.1		0.5	
0.2			
0.3		DgtMax	30
0.4			
0.5			

The function **xeval** only assigns a column (or a row) of values to the correspondent variable on top (or at left)

## Complex Expression Evaluation

**=cplxeval( Formula, [Var1, Var2 ...] )**

This function<sup>23</sup> evaluates a math expression in complex arithmetic.

The parameter "Formula" is a math expression string containing variables, operators, parenthesis and other basic functions.

$(3+8j)*(-1-4j)$  ,  $(1+i)*\ln(1+3i)$  ,  $((x+3i)/(x+4-2i))^{(1-i)}$

The optional parameter "Var1", "Var2",... can be single or complex value. See *How to insert a complex number* for better details

Example: Evaluate the given complex polynomial for  $z = 2 - i$

$$z^2 + (3+i)z + (2+5i)$$

<sup>23</sup> This function uses the `clsMathParserC` class by A. De Grammont and L. Volpi

## Xnumbers Tutorial

	A	B	C	D
1	<b>f(z)</b>	<b>z</b>		<b>Result</b>
2	$z^2 + (3+2j)z + 2+5j$	2		13
3		-1		2
4				
5	{=cplx_eval(A2;B2:B3)}			

Note that we use the complex rectangular format only in the symbolic math formula. When we pass a complex variable we must always use the double cell format  
Note also that we can write “i” or “j” as well for imaginary symbol, the parser can recognize both of them.

For complex numbers labels are not supported. When we have formulas with two or more variables, we must provide the values for variable substitutions in the exact order that they appear in the formula, starting from left to right. The formula wizard will easily help you. Look at this example.

Example. Compute the expression for the given complex values

$$F(s) = \frac{e^{ks}}{(s-a)(s-b)} \quad s = 1 + j, a = 1 - 4j, b = 3 + 6j, k = -0.5$$

In the cell B2 we have inserted the string

“exp(k\*s) / ((s-a) (s-b))”

	A	B	C	D
1				
2	F(s) =	exp(k*s)/((s-a)*(s-b))		
3				
4	k	-0.5		
5	a	1	-4	
6	b	3	6	
7				
8	s re	s imm	F(s) re	F(s) imm
9	1	1	0.0223654	-0.00268531
10				
11	{=cplx_eval(B2;B4;A9:B9;B5:C5;B6:C6)}			
12				

Argomenti funzione

cplx\_eval

Formula B2 = "exp(k\*s)/((s-a)\*(s-b))"

Var =

= "k s a b"

Complex evaluator.

When we enter the formula, the parser recognizes the variables symbols and shows us the exact order in which we have to pass to the function itself. In this case: **k, s, a, b**

## Math expression strings

Functions like **Integr**, **Series**, **xeval**, **xevall**, **cpplxeval** operate with symbolic math expressions by the aid of **clsMathParser** and **claMathparserC** evaluators (two internal class modules).

These programs (for real and complex numbers) accept in input any string representing an arithmetic or algebraic expression with a list of variable values and return a single numeric result.

Typical math expressions are:

$1+(2-5)*3+8/(5+3)^2$	<code>sqr(2)+asin(x)</code>
$(a+b)*(a-b)$	<code>x^2+3*x+1</code>
$1.5*\exp(-t/12)*\cos(\pi*t + \pi/4)$	<code>(1+(2-5)*3+8/(5+3)^2)/sqr(5^2+3^2)</code>
$2+3x+2x^2$	<code>0.25x + 3.5y + 1</code>
$\text{sqr}(4^2+3^2)$	<code>1/(1+e# ) + Root(x,6)</code>
$(-1)^{(2n+1)}*x^n/n!$	<code> x-2 + x-5 </code>
<code>And((x&lt;2),(x&lt;=5))</code>	<code>sin(2*pi*x)+cos(2*pi*x)</code>

**Variables** can be any alphanumeric string and must start with a letter

`x, y, a1, a2, time, alpha, beta`

Also the symbol "\_" is accepted to build variable names in "programming style".

`time_1, alpha_b1, rise_time`

Capitals are accepted but ignored. Names such as "Alpha", "alpha", "ALPHA" indicate the same variable.

**Implicit multiplication** is not supported because of its intrinsic ambiguity. So "xy" stands for variable named "xy" and not for  $x*y$ . The multiplication symbol "\*" generally cannot be omitted. It can be omitted only for coefficients of the classic math variables x, y, z. It means that string like 2x and 2\*x are equivalent

`2x, 3.141y, 338z^2`  $\Leftrightarrow$  `2*x, 3.141*y, 338*z^2`

On the contrary, the following expressions are illegal in this context.

`2a, 3(x+1), 334omega`

**Constant numbers** can be integer, decimal, or exponential

`2, -3234, 1.3333, -0.00025, 1.2345E-12`

**Logical expression** are supported

`"x<1", "x+2y >= 4", "x^2+5x-1>0", "t<>0", and(x>0;x<1)`

Logical expressions always returns 1 (True) or 0 (False). Multiple logical expression, like "0<x<1", are not supported; you must enter:

`and(x>0,x<1) or (x>0)*(x<1)`

**Math Constants** supported are: Pi Greek ( $\pi$ ), Euler-Napier

`pi = 3.14159265358979 or pi# = 3.14159265358979`  
`e# = 2.71828182845905`

### Angle expression

This version supports angles in RAD radians, DEG degree, or GRAD degree. For example if you set the unit "DEG", all angles will be read and converted into degrees

```
sin(120) => 0.86602540378444
asin(0.86602540378444) => 120
rad(pi/2) => 90      , grad(400) => 360  , deg(360) => 360
```

Angles can also be write in DMS format like for example 45° 12' 13"

```
sin(29°59'60") => 0.5
```

**Complex number** can be indicated in a formula string as an ordered couple of number enclosed into parenthesis "(..)" and divided by a comma "," like for example:

```
(2, 3)      (a, b)      (-1, -0.05)      (-1.4142135623731, -9.94665E-18)
```

On the other hand, complex numbers can also be indicate by the common rectangular form:

```
3+3j      a+bj      -1 - 0.05j      -1.4142135623731 - 9.94665E-18j
```

You note that the second form is suitable for integer numbers, while, on the contrary, for decimal or exponential number the first one is clearer. The parenthesis form is more suitable also in nested results like

```
((2+3*4), (8-1/2)) that gives the complex number (14, 7.5)
```

**Note:** Pay attention if you want to use the rectangular convention in nested formulas.

```
wrong (2+3*4)+(8-1/2)j .      correct (2+3*4)+(8-1/2)*j      .
```

Do not omit the product symbol "\*" before j because the parser recognize it as an expression, not a complex number. The product symbol can be omitted only when before the letter "j" is a constant number

**Note:** You can use both "j" and "i" for indicating the imaginary number  $\sqrt{-1}$

## List of basic functions and operators

Function	Description	Note
+	addition	
-	subtraction	
*	multiplication	
/	division	$35/4 = 8.75$
%	percentage	$35\% = 3.5$ , $1000+35\% = 1035$
\	integer division	$35 \setminus 4 = 8$
^	raise to power	$3^{1.8} = 7.22467405584208$
	absolute value	$ -5 =5$ (the same as abs)
!	factorial	$5! = 120$ (the same as fact)
abs(x)	absolute value	$\text{abs}(-5) = 5$
atn(x)	inverse tangent	
cos(x)	cosine	argument in radians
sin(x)	sine	argument in radians
exp(x)	exponential	$\exp(1) = 2.71828182845905$
fix(x)	integer part	$\text{fix}(-3.8) = 3$
int(x)	integer part	$\text{int}(-3.8) = 4$
dec(x)	decimal part	$\text{dec}(-3.8) = -0.8$
ln(x)	logarithm natural	argument $x > 0$
log(x)	logarithm decimal	argument $x > 0$
rnd(x)	random	returns a random number between x and 0
sgn(x)	sign	returns 1 if $x > 0$ , 0 if $x = 0$ , -1 if $x < 0$
sqr(x)	square root	$\text{sqr}(2) = 1.4142135623731$ , also $2^{1/2}$
cbr(x)	cube root	$\forall x$ , example $\text{cbr}(2) = 1.2599$ , $\text{cbr}(-2) = -1.2599$
tan(x)	tangent	argument (in radians) $x \neq k \cdot \pi/2$ with $k = \pm 1, \pm 2 \dots$
acos(x)	inverse cosine	argument $-1 \leq x \leq 1$
asin(x)	inverse sine	argument $-1 \leq x \leq 1$
cosh(x)	hyperbolic cosine	
sinh(x)	hyperbolic sine	
tanh(x)	hyperbolic tangent	
acosh(x)	inverse hyperbolic cosine	argument $x \geq 1$
asinh(x)	inverse hyperbolic sine	
atanh(x)	inverse hyperbolic tangent	argument $-1 < x < 1$
root(x,n)	n-th root (the same as $x^{1/n}$ )	Argument $n \neq 0$ , $x \geq 0$ if n even , $\forall x$ if n odd
mod(a, b)	division quotient	
fact(x)	factorial	argument $0 \leq x \leq 170$
comb(n,k)	combinations	$\text{comb}(6,3) = 20$
min(a, b)	min between two numbers	
max(a, b)	max between two numbers	
mcd(a, b)	maximum common divisor between two numbers	$\text{mcm}(4346,174) = 2$
mcm(a, b)	minimum common multiple between two numbers	$\text{mcm}(4346,174) = 378102$
gcd(a, b)	greatest common divisor between two numbers	The same as mcd
lcm(a, b)	lowest common multiple between two numbers	The same as mcm
erf(x)	error Gauss's function	argument $x > 0$
gamma(x)	gamma	argument $0 < x < 172$
gammain(x)	logarithm gamma	argument $x > 0$
digamma(x)	digamma	argument $x > 0$
beta(x,y)	beta	argument $x > 0$ $y > 0$
zeta(x)	zeta Riemman's function	argument $x < -1$ or $x > 1$
ei(x)	exponential integral function	argument $x > 0$
csc(x)	cosecant	argument (in radians) $x \neq k \cdot \pi$ with $k = 0, \pm 1, \pm 2 \dots$
sec(x)	secant	argument (in radians) $x \neq k \cdot \pi/2$ with $k = \pm 1, \pm 2 \dots$
cot(x)	cotangent	argument (in radians) $x \neq k \cdot \pi$ with $k = 0, \pm 1, \pm 2 \dots$
acsc(x)	inverse cosecant	
asec(x)	inverse secant	

## Xnumbers Tutorial

acot(x)	inverse cotangent	
csch(x)	hyperbolic cosecant	argument $x > 0$
sech(x)	hyperbolic secant	argument $x > 1$
coth(x)	hyperbolic cotangent	argument $x > 2$
acsch(x)	inverse hyperbolic cosecant	
asech(x)	inverse hyperbolic secant	argument $0 \leq x \leq 1$
acoth(x)	inverse hyperbolic cotangent	argument $x < -1$ or $x > 1$
rad(x)	radians conversion	converts radians into current unit of angle
deg(x)	degree DEG. conversion	converts DEG degree into current unit of angle
grad(x)	degree GRAD. conversion	converts GRAD. degree into current unit of angle
round(x,d)	round a number with d decimal	round(1.35712, 2) = 1.36
>	greater than	return 1 (true) 0 (false)
>=	equal or greater than	return 1 (true) 0 (false)
<	less than	return 1 (true) 0 (false)
<=	equal or less than	return 1 (true) 0 (false)
=	equal	return 1 (true) 0 (false)
<>	not equal	return 1 (true) 0 (false)
and	logic and	and(a, b) = return 0 (false) if $a=0$ or $b=0$
or	logic or	or(a, b) = return 0 (false) only if $a=0$ and $b=0$
not	logic not	not(a) = return 0 (false) if $a \neq 0$ , else 1
xor	logic exclusive-or	xor(a, b) = return 1 (true) only if $a \neq b$
nand	logic nand	nand(a, b) = return 1 (true) if $a=1$ or $b=1$
nor	logic nor	nor(a, b) = return 1 (true) only if $a=0$ and $b=0$
nxor	logic exclusive-nor	nxor(a, b) = return 1 (true) only if $a=b$

Symbol "!" is the same as "Fact", symbol "/" is the integer division, symbols "|x|" is the same as Abs(x)  
 Logical function and operators returns 1 (true) or 0 (false)



## Function Optimization

### Macros for optimization on site

These macros has been ideated for performing the optimization task directly on the worksheet. This means that you can define any function that you want simply using the standard Excel built-in functions.

**Objective function.** For example: if you want to search the minimum of the bivariate function

$$f(x, y) = \left(x - \frac{51}{100}\right)^2 + \left(y - \frac{35}{100}\right)^2$$

insert in the cell E4 the formula "`=(B4-0.51)^2+(C4-0.35)^2`", where the cells B4 and C4 contain the current values of the variables x and y respectively. Changing the values of B4 e/o C4 the function value E4 also changes consequently.

	A	B	C	D	E	F
1						
2						
3		x	y		f(x,y)	
4		0	0		0.3826	
5						
6						
7		Minimize changing these cells			Function to minimize =(B4-0.51)^2+(C4-0.35)^2	
8						

For optimization, you can choose two different algorithms

<p><b><u>Downhill-Simplex</u></b><sup>24</sup></p> <p>The Nelder–Mead downhill simplex algorithm is a popular derivative-free optimization method. Although there are no theoretical results on the convergence of this algorithm, it works very well on a wide range of practical problems. It is a good choice when a one-off solution is wanted with minimum programming effort. It can also be used to minimize functions that are not differentiable, or we cannot differentiate. It shows a very robust behavior and converges for a very large set of starting points. In our experience is the best general purpose algorithm, solid as a rock, it's a "jack" for all trades.</p>	<p>For mono and multivariate functions without constrains</p>
<p><b><u>Divide-Conquer 1D</u></b></p> <p>For monovariate function only, it is an high robust derivative free algorithm. It is simply a modified version of the bisection algorithm Adapt for every function, smooth or discontinue. It converges for very large segments. Starting point not necessary</p>	<p>For monovariate function only. It needs the segment where the max or min is located</p>

Example assume to have to minimize the following function for  $x > 0$

<sup>24</sup> The Downhill-Simplex of Nelder and Maid routine appears by the courtesy of Luis Isaac Ramos Garcia



## Xnumbers Tutorial

$$f(x) = e^{-3x} \sin(3x) + e^{-x} \cos(4x)$$

We try to search the minimum in the range  $0 < x < 10$   
Choose a cell for the variable  $x$ , example B6, and insert the function

`= SIN(3*B6)*EXP(- 2*B6) + COS(4*B6)*EXP(-B6)`

in a cell that you like, for example C6.

After this, add the constrain values into another range, for example B3:C3  
The values of the variables at the start are not important

	C6		$f_x$	=SEN(3*B6)*EXP(- 2*B6) + COS(4*B6)*EXP(-B6)		
	A	B	C	D	E	F
1						
2		a	b			
3		0	10			
4						
5		x	f(x)			
6		10	-3.0281E-05			
7						
8						
9						

constrains:  $a < x < b$

Objective function

Variable to change

Select the cell of the function C6 and start the macro "1D divide and conquer", filling the input field as shown

**Functions Optimizator Tool**

Optimization on site      1D-Divide-Conquer algorithm

Objective Function:  $f(x, y \dots)$

     ☒ Min    ☐ Max

Variables:  $x, y \dots$

     Points:

constrains box:

**Stopping limit.** Set the maximum evaluation points allowed.

**Max/Min.** The radio buttons switches between the minimization and maximization algorithm

The "**Downhill-Simplex**" macro is similar except that:

- The constrain box is optional.
- It accepts up to 9 variables (range form 1 to 9 cells)
- The algorithm starts from the point that you give in the variable cells. If the constrain box is present, the algorithm starts from a random point inside the box

Let's see how it works.

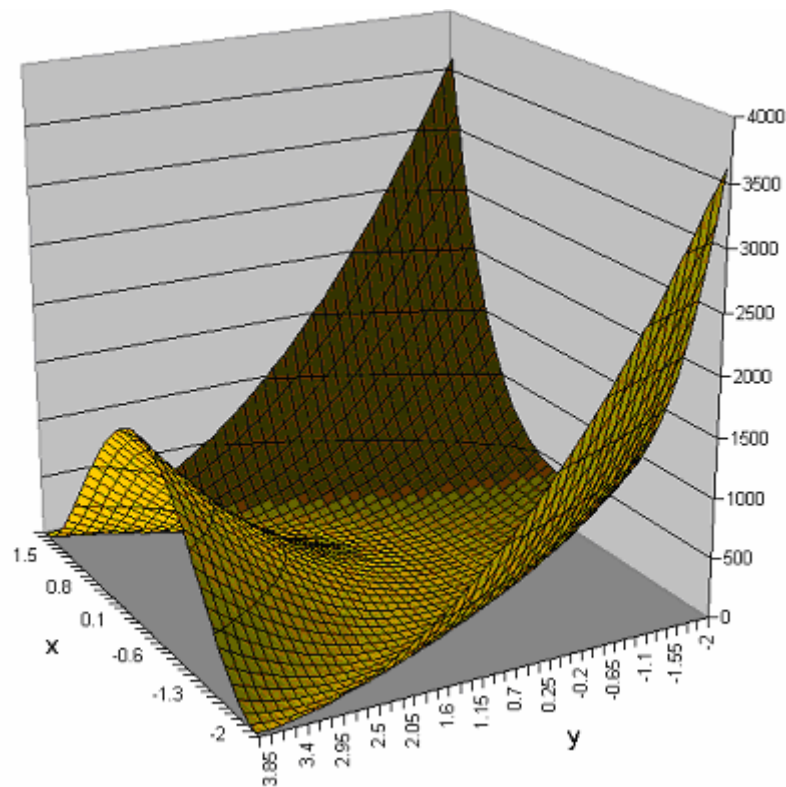
The following examples are extracted from "*Optimization and Nonlinear Fitting*", Foxes Team, Nov. 2004

### Example 1 - Rosenbrock's parabolic valley

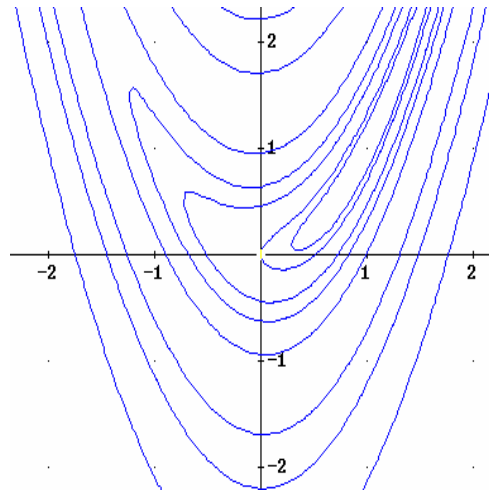
This family of test functions is well known to be a minimizing problem of high difficult

$$f(x, y) = m \cdot (y - x^2)^2 + (1 - x)^2$$

The parameter "m" tunes the difficult: high value means high difficult in minimum searching. The reason is that the minimum is located in a large flat region with a very low slope. The following 3D plot shows the Rosenbrock's parabolic valley for m = 100



The following contour plot is obtained for m = 10



The function is always positive except in the point (1, 1) where it is 0. it is simple to demonstrate it, taking the gradient

$$\nabla f = 0 \Rightarrow \begin{cases} 4m \cdot x^3 + 2x(1 - 2m \cdot y) - 2 = 0 \\ 2m(y - x^2) = 0 \end{cases}$$

From the second equation, we get

$$2m(y - x^2) = 0 \Rightarrow y = x^2$$

Substituting in the first equation, we have

$$4m \cdot x^3 + 2x(1 - 2m \cdot x^2) - 2 = 0 \Rightarrow 2x - 2 = 0 \Rightarrow x = 1$$

So the only extreme is the point (1, 1) that is the absolute minimum of the function

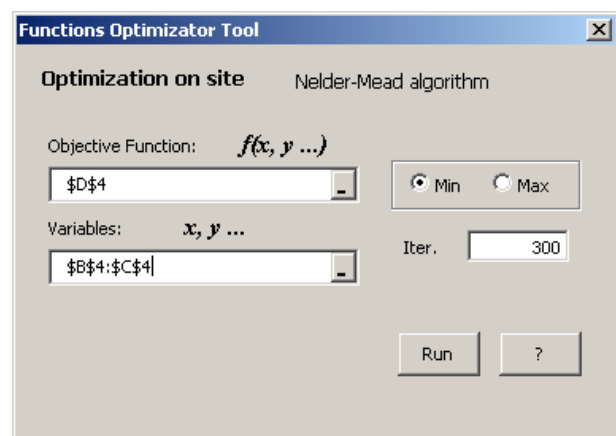
To find numerically the minimum, let's arrange a similar sheet.

We can insert the function and the parameters as we like

Select the cell D4 - containing the objective function - and start the macro "**Downhill-Simplex**". The macro fills automatically the variables-field with the cells related to the objective function. But, In that case, the cell A4 contains the parameter m that must not change. So insert the range B4:C4 into the variables field.

	A	B	C	D
1	<b>Minimum searching</b>			
2	Rosenbrock's parabolic valley			
3	<b>m</b>	<b>x</b>	<b>y</b>	<b>f(x,y)</b>
4	10	0	0	1
5				
6	=A4*(C4-B4^2)^2+(1-B4)^2			

The cells B4:C4 will change for minimizing the objective function in the cell D4



Starting from the point (0, 0) we obtain the following good results

m	Algorithm	x	y	error	time
10	Simplex	1	1	2.16E-13	2 sec
100	Simplex	1	1	4.19E-13	2 sec

Where the error is calculated as  $|x-1|+|y-1|$

## Example 2 - Constrained minimization

Example: assume to have to minimize the following function

$$f(x, y) = x^2 + 2xy - 4x + 4y^2 - 10y + 7$$

with the ranges constrains

$$0 \leq x \leq 2, \quad 0 \leq y \leq 0.5$$

The Excel arrangement can be like the following

The screenshot shows an Excel spreadsheet with the following data:

	A	B	C
1	x	y	f(x,y)
2	0	0	7
3			
4	x	y	
5	0	0	min
6	2	0.5	max

The Functions Optimizer Tool dialog box is open, showing the Nelder-Mead algorithm. The Objective Function is set to  $f(x, y \dots)$  and the variable cell is  $\$C\$2$ . The Variables are set to  $x, y \dots$  and the variable range is  $\$A\$2:\$B\$2$ . The constraints box is set to  $\$A\$5:\$B\$6$ . The 'Run' button is visible.

Compare with the exact solution  $x = 1.5, y = 0.5$

Note that the function has a free minimum at  $x = 1, y = 1$

Repeat the example living empty the constrains box input, for finding those free extremes.

### Example 3 - Nonlinear Regression with Absolute Sum

This example explains how to perform a nonlinear regression with an objective function different from the "Least Squared". In this example we adopt the "Absolute Sum".

We choose the exponential model

$$f(x, a, k) = a \cdot e^{-k \cdot x}$$

The goal of the regression is to find the best couple of parameters (a, k) that minimizes the sum of the absolute errors between the regression model and the given data set.

$$AS = \sum |y_i - f(x_i, a, k)|$$

The objective function AS depends only by parameter a, k. Giving in input this function to our optimization algorithm we hope to solve the regression problem

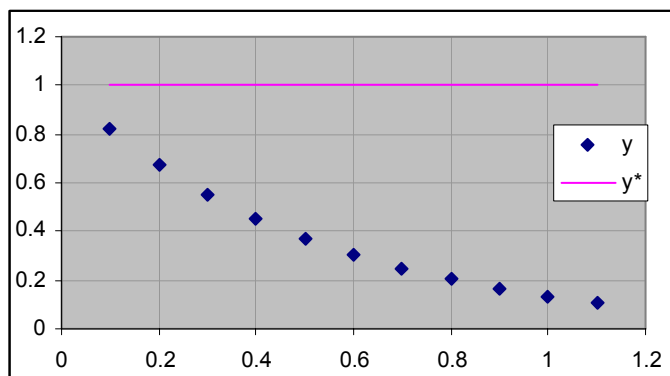
A possible arrangement of the worksheet may be:

	A	B	C	D	E	F	G
1							
2	x	y	y*	y-y*	a	k	Σ error
3	0.1	0.8187308	1	0.1812692	1	0	6.98380414
4	0.2	0.67032	1	0.32968			
5	0.3	0.5488116	1	0.4511884			
6	0.4	0.449329	1	0.550671			
7	0.5	0.3678794	1	0.6321206			
8	0.6	0.3011942	1	0.6988058			
9	0.7	0.246597	1	0.753403			
10	0.8	0.2018965	1	0.7981035			
11	0.9	0.1652989	1	0.8347011			
12	1	0.1353353	1	0.8646647			
13	1.1	0.1108032	1	0.8891968			

Formulas shown in the image:

- Cell G3: `=SUM(D3:D13)`
- Cell E3: `=$E$3*EXP($F$3*A13)`
- Cell D3: `=ABS(B13-C13)`

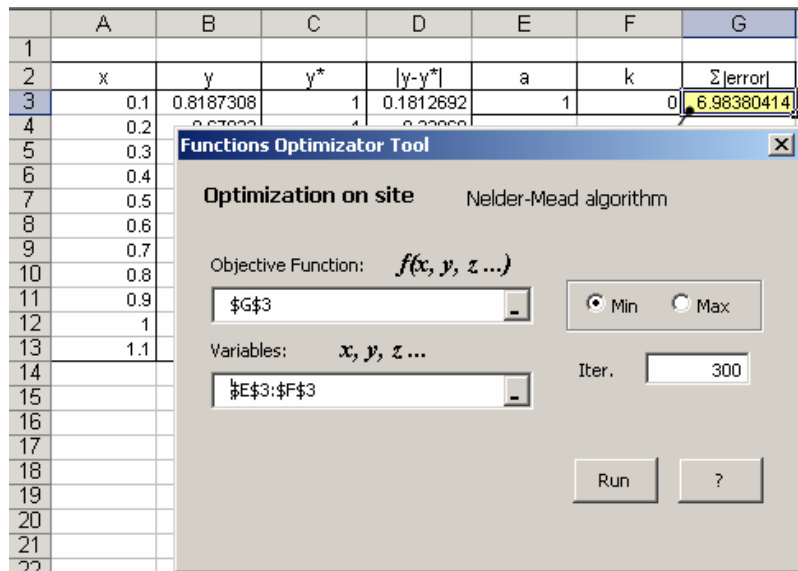
We hope that changing the parameters "a" and "k" into the cells E2 and F3, the objective function (yellow cell) goes to its minimum value. Note that the objective function depends indirectly by the parameters a and k.



The starting condition is the following, where y indicates the given data and y\* is the regression plot (a flat line at the beginning)

Start the Downhill-Simplex and insert the appropriate range as shown in the picture

## Xnumbers Tutorial

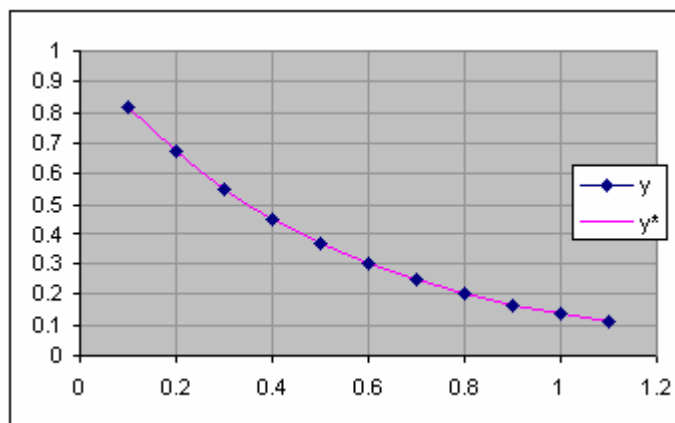


Starting from the point (1, 0) you will see the cells changing quickly until the macro stops itself leaving the following "best" fitting parameters and the values of the regression  $y^*$

Best fitting parameters

a	k
1	-2

The plot of the  $y^*$  function and the samples  $y$  are shown in the graph. As we can see the regression fits perfectly the given dataset.



## References

*"Computation of Special Functions"* by Shanjie Zhang and Jianming Jin - John Wiley and Sons, Inc

Computation of Special Functions" is a valuable book/software package containing more than 100 original computer programs for the computation of most special functions currently in use. These include many functions commonly omitted from available software packages, such as the Bessel and modified Bessel functions, the Mathieu and modified Mathieu functions, parabolic cylinder functions, and various prolate and oblate spheroidal wave functions. Also, unlike most software packages, this book/disk set gives readers the latitude to modify programs according to special demands of the sophisticated problems they are working on.

All the programs and subroutines contained in this library are copyrighted. However, authors kindly gave permission to the user to incorporate any of these routines into his or her programs.

We have cured to preserve the clean original code, modifying as less as possible, to adapt it from FORTRAN 77 to the VBA language.

*"Lanczos Implementation of the Gamma Function"* by Paul Godfrey, Intersil Corp, 2001

A note on the computation of the convergent Lanczos complex Gamma approximation.

Abstract: The convergent approximation of the Gamma function as developed by Lanczos is examined. It is found that the coefficients in the series expansion can be written as the product of 4 matrices. This allows greater accuracy in computing the function.[...] A 15 term expansion was found that provides an accuracy of about 15 significant digits.

Resource Library of Wolfram Research by Eric Weisstein

*"Numerical Recipes in FORTRAN 77- The Art of Scientific Computing"* - 1986-1992 by Cambridge University Press. Programs Copyright (C) 1986-1992 by Numerical Recipes Software

*clsMathParser - A Class for Math Expressions Evaluation in Visual Basic*, Leonardo Volpi , Michael Ruder, Thomas Zeuschler, Lieven Dossche, . 3.2 Jan. 2003, by .Volpi

*clsMathParserC -A Class for Complex Math Expressions Evaluation in Visual Basic*, Arnaud de Grammont, Leonardo Volpi, v. 3.2 , Jan. 2003

*F F T (Fast Fourier Transform)*, Paul Bourke, June 1993, <http://astronomy.swin.edu.au>

*2 Dimensional FFT* , Paul Bourke, June 1998, <http://astronomy.swin.edu.au>

*"Solutions Numeriques des Equations Algebriques"*, E., Durand, Tome I, Masson,Paris ,1960.

*"A modified Newton method for Polynomials"* , W.,Ehrlich, Comm., ACM, 1967, 10, 107-108.

*"Ein Gesamtschrittverfahren zur Berechnung der Nullstellen eines Polynoms"*, O., Kerner, Num.Math., 1966, 8, 290-294.

*"Iteration methods for finding all the zeros of a polynomial simultaneously"*, O. Aberth, Math. Comp. ,1973, 27, 339-344.

*"The Ehrlich-Aberth Method for the nonsymmetric tridiagonal eigenvalue problem"*, D. A. Bini, L. Gemignani, F. Tisseur, AMS subject classifications. 65F15

*"Progress on the implementetion of Aberth's method"*, D. A. Bini, G. Fiorentino, , 2002, The FRISCO Consortium (LTR 21.024)

*"Nonlinear regression"*, Gordon K. Smyth Vol. 3, pp 1405/1411, in Encyclopedia of Environmetrics (ISBN 0471 899976), Edited by John Wiley & Sons, Ltd, Chichester, 2002

*"Optimization"*, Gordon K. Smyth Vol. 3, pp 1481/1487, in Encyclopedia of Environmetrics (ISBN 0471 899976), Edited by John Wiley & Sons, Ltd, Chichester, 2002

*"Process Modeling"*, The National Institute of Standards and Technology (NIST) website for Statistical Reference Datasets, (<http://www.itl.nist.gov/div898/handbook/pmd/pmd>)

## Xnumbers Tutorial

"*Metodos numericos con Matlab*", J. M Mathewss et al., Prentice Hall

"*Genetic and Nelder-Mead*", E. Chelouan, et al, EJOR 148(2003) 335-348

"*Convergence properties*", J.C. Lagarias, et al, SIAM J Optim. 9(1), 112-147

"*Optimization for Engineering Systems*", Ralph W. Pike, 2001, Louisiana State University  
(<http://www.mpri.lsu.edu/bookindex>)

"*Zeros of Orthogonal Polynomials*" , Leonardo Volpi, Foxes Team  
(<http://digilander.libero.it/foxes/Documents>)

"*How to tabulate the Legendre's polynomials coefficients*" Leonardo Volpi, Foxes Team  
(<http://digilander.libero.it/foxes/Documents>)

"*Gli algoritmi della crittografia a chiave pubblica*", Giovanni Fraterno, settembre 2000  
(<http://digilander.libero.it/crittazione>)

"*Improving Exact Integral from Symbolic Algebra System*", R.J. Fateman and W. Kaham,  
University of California, Berkeley, July 18,2000

"*NIST/SEMATECH e-Handbook of Statistical Methods*", January 26, 2005  
(<http://www.itl.nist.gov/div898/handbook>)

"*DE-Quadrature (Numerical Automatic Integrator) Package*", by Takuya OOURA, Copyright(C) 1996,  
Research Institute for Mathematical Sciences, Kyoto University, Kyoto 606-01, Japan  
(<http://momonga.t.u-tokyo.ac.jp/~oura>)

"*A Comparison of three high-precision quadrature schemes*", David H. Bailey and Xiaoye S. Li,  
Copyright 2003, University of California  
(<http://repositories.cdlib.org/lbnl/LBNL-53652>)

"*Asymptotic Bit Cost of Quadrature Formulas Obtained by Variable Transformation*" P. Favati,  
Appl. Math. Lett. Vol. 10, No. 3, pp. 1-7, 1997, Pergamon Copyright 1997

"*Matrix Analysis and Applied Linear Algebra*" Carl Meyer, 2000, SIAM, Philadelphia

"*Handbook of Mathematical Functions*", by M. Abramowitz and I. Stegun, Dover Publication  
Inc., New York.



## Credits

### Software developed

Xnumbers contains code developed by the following authors that kindly contributed to this collection.

<a href="#">Luis Isaac Ramos Garcia</a>	Orthogonal Polynomials and Downhill Simplex Nelder-Mead routine
<a href="#">Olghierd Zieba</a>	Cubic Spline and documentation
<a href="#">Alfredo Álvarez Valdivia</a>	Robust Method fitting routines
<a href="#">Michael J. Kozluk</a>	Log Relative Error function, Linear regression debugging, and documentation (StRD benchmark)
<a href="#">Ton Jeursen</a>	Format function, Xnumber debugging; Xnumber for Excel95
<a href="#">Richard Huxtable</a>	ChangeBase and Prime functions
<a href="#">Michael Ruder</a>	MathParser improvement and debugging
<a href="#">Thomas Zeutschler</a>	MathParser improvement and debugging
<a href="#">Lieven Dossche</a>	Class MathParser development
<a href="#">Arnaud de Grammont</a>	Complex MathParser developement
<a href="#">Rodrigo Farinha</a>	MathParser improvement and debugging
<a href="#">Vladimir Zakharov</a>	Installation and initialization improvement

### Software translated

Xnumbers contains VB code translated from the following packages:

<a href="#">Takuya OOURA</a>	DE-Quadrature (Numerical Automatic Integrator) Package
<a href="#">Shanjie Zhang</a> <a href="#">Jianming Jin</a>	FORTTRAN routines for computation of Special Functions

Many thanks to everybody

# Analytical Index

## A

Absolute; 31  
 Adams; 202; 203  
 Adams-Bashfort-Moulton; 202  
 Addition; 25  
 Aitken; 156; 174  
 Arccos; 66  
 Arcsine; 66  
 Arctan; 66  
 Arithmetic Mean; 42

## B

Base conversion; 218  
 baseChange; 218  
 Bessel functions; 226  
 BesseldI; 226  
 BesseldJ; 226  
 BesseldK; 226  
 BesseldY; 226  
 Bessell; 226  
 BesselJ; 226  
 BesselK; 226  
 BesselY; 226  
 Beta function; 224  
 Bisection; 209  
 Bivariate Polynomial; 98

## C

Cebychev; 86  
 Central Polynomial; 77  
 Change sign; 31  
 Check digits; 36  
 Check odd/even; 121  
 Check Prime; 120  
 CheckPrime; 120  
 Cholesky; 129  
 Circle of the Roots; 79  
 Coefficients of Orthogonal Polynomials;  
 107  
 Coefficients Transformation; 78  
 Combinations; 41  
 Combinations function; 225  
 Compare numbers; 34  
 Complement of right angle; 67  
 Complex absolute; 111  
 Complex Addition; 109  
 Complex ArcCos; 113

Complex ArcSin; 114  
 Complex ArcTan; 114  
 Complex Complementary Error Function;  
 115  
 Complex conjugate; 113  
 Complex Cos; 113  
 Complex digamma; 115  
 Complex Division; 110  
 Complex Error Function; 115  
 Complex Exp; 112  
 Complex Exponential Integral; 115  
 Complex Expression Evaluation; 233  
 Complex Function Integration (Romberg  
 method); 152  
 Complex Gamma Function; 116  
 Complex Hyperbolic Cosine; 114  
 Complex Hyperbolic Sine; 114  
 Complex Hyperbolic Tan; 114  
 Complex inverse; 112  
 Complex Inverse Hyperbolic Cos; 114  
 Complex Inverse Hyperbolic Sin; 114  
 Complex Inverse Hyperbolic Tan; 115  
 Complex Log; 112  
 Complex Logarithm Gamma Function; 116  
 Complex Multiplication; 109  
 Complex negative; 113  
 Complex power; 111  
 Complex Quadratic Equation; 117  
 Complex Roots; 111  
 Complex Series Evaluation; 175  
 Complex Sin; 113  
 Complex Subtraction; 109  
 Complex Tangent; 113  
 Complex Zeta Function; 116  
 Constant "e"; 62  
 Constant Ln(10); 62  
 Constant Ln(2); 62  
 Constant pi; 66  
 Convert Extended Number; 38  
 Convolve; 180  
 corrector; 202  
 Corrector; 203; 205  
 Cos; 65  
 Cosine Integral Ci(x); 226  
 CosIntegral; 226  
 cplxabs; 111  
 cplxacos; 113  
 cplxacosh; 114  
 cplxadd; 109  
 cplxasin; 114

cplxasinh; 114  
 cplxatan; 114  
 cplxatanh; 115  
 cplxconj; 113  
 cplxcos; 113  
 cplxsinh; 114  
 cplxdigamma; 115  
 cplxdiv; 110  
 cplxexp; 115  
 cplxEquation2; 117  
 cplxerf; 115  
 cplxerfc; 115  
 cplxeval; 233  
 cplxExp; 112  
 cplxgamma; 116  
 cplxgammaLn; 116  
 cplxintegr; 152  
 cplxinv; 112  
 cplxLn; 112  
 cplxmult; 109  
 cplxneg; 113  
 cplxpol; 110  
 cplxpow; 111  
 cplxrect; 110  
 cplxroot; 111  
 cplxserie; 175  
 cplxsin; 113  
 cplxsinh; 114  
 cplxsub; 109  
 cplxtan; 113  
 cplxatanh; 114  
 cplxzeta; 116  
 Crout; 128  
 cspline\_coeff; 186  
 cspline\_eval; 184  
 cspline\_interp; 184  
 cspline\_pre; 185  
 Cubic Spline 2nd derivatives; 185  
 Cubic Spline Coefficients; 186  
 cvBaseDec; 218  
 cvBinDec; 218  
 cvDecBase; 218  
 cvDecBin; 218

**D**

Data Conditioned Linear Regression  
   Coefficients; 53  
 Data Conditioning; 52  
 Data Integration (Newton Cotes); 154  
 Data Integration (Romberg method); 146  
 dBel; 218  
 Decibel; 218  
 Decimal part; 31  
 DFSP; 138  
 DFSP\_INV; 139  
 DFT; 136; 141  
 DFT\_INV; 137  
 Diff1; 212  
 Diff2; 213

digamma; 224  
 Digamma function; 224  
 Digit\_Max; 24; 39  
 Digits count; 34  
 Digits sum; 36  
 DigitsAllDiff; 36  
 Diophantine; 125  
 Diophantine Equation; 125  
 DiophEqu; 125  
 Discrete 2D Fourier Transform; 139  
 Discrete Convolution; 180  
 Discrete Fourier Inverse Transform; 137  
 Discrete Fourier Spectrum; 138  
 Discrete Fourier Transform; 136  
 Division; 26  
 Double Exponential; 148  
 Double Integral; 166  
 Double integration function; 168  
 Double Integration macro; 166  
 Double Series; 176  
 Downhill; 241  
 DPOLYN; 82

## E

*Eratostene*; 122  
 errfun; 221  
 Error Function Erf(x); 221  
 Euler; 202  
 Euler's constant gamma; 64  
 Euler-Mascheroni Constant; 221  
 exp\_integr; 221  
 exp\_integr\_n; 221  
 Exponential; 61  
 Exponential any base; 61  
 Exponential integral Ei(x); 221  
 Exponential integral En(x); 221  
 Extended Number Check; 35

## F

Factor; 122  
 Factorial; 41  
 Factorial with double-step; 41  
 Factorize; 121  
 Factorize function; 122  
 Fermat; 123  
 Fermat's Prime Test; 123  
 FFT; 136; 141  
 FFT\_INV; 137  
 FFT2D; 139  
 FFT2D\_INV; 140  
 Fibonacci numbers; 227  
 First Derivative; 212  
 Flip; 37  
 Format Extended Number; 35  
 Fourier; 164  
 Fourier\_cos; 162  
 Fourier\_cos; 164  
 Fourier\_cos; 165  
 Fourier\_sin; 162

fract; 119  
 Fract\_Interp; 182  
 Fract\_Interp\_Coef; 182  
 Fresnel cosine Integral; 227  
 Fresnel sine Integral; 227  
 Fresnel\_cos; 227  
 Fresnel\_sin; 227  
 Function Integration (Double Exponential method); 148  
 Function Integration (mixed method); 150  
 Function Integration (Newton-Cotes formulas); 156  
 Function Integration (Romberg method); 147

## G

Gamma F-factor; 224  
 Gamma function; 222  
 Gamma quotient; 223  
 GCD; 118  
 Geometric Mean; 42  
 Grad; 213  
 Gradient; 213  
 Greatest Common Divisor; 118

## H

Hermite; 86  
 Hessian; 214  
 Hessian matrix; 214  
 Hyperbolic Arc Cosine; 63  
 Hyperbolic Arc Sine; 62  
 Hyperbolic Arc Tangent; 63  
 Hyperbolic Cosine; 63  
 Hyperbolic Sine; 62  
 Hyperbolic Tangent; 63  
 Hypergeom; 228  
 Hypergeometric function; 228

## I

Infinite integral; 170  
 Infinite Integration of oscillating functions; 163  
 Integer Division; 27  
 Integer part; 31  
 Integer polynomial; 94  
 Integer Remainder; 27  
 Integer roots; 74  
 integr; 164; 165  
 Integr; 150  
 Integr\_2D; 168  
 Integr\_fcos; 160  
 Integr\_fsin; 160  
 Integr\_nc; 156  
 Integr\_ro; 147  
 Integr2D; 166  
 Integral\_Inf; 170  
 Integration of oscillating functions (Filon formulas); 160

Integration of oscillating functions (Fourier transform); 162  
 IntegrDataC; 154  
 IntegrDataR; 146  
 Interp\_Mesh; 188  
 InterpL; 187  
 InterpL\_Coef; 187  
 Interpolation 2D; 188  
 Interpolation with continue fraction; 182  
 Interpolation with Cubic Spline; 184  
 IntRombergMat; 146  
 Inverse; 26  
 Inverse 2D Discrete Fourier Transform; 140  
 Inverse Discrete Fourier Spectrum; 139  
 isXnumbers; 35

## J

Jacobian; 214  
 Jacobian matrix; 214

## L

LCM; 118  
 Least Common Multiple; 118  
 Legendre; 86  
 Linear Regression Coefficients; 44  
 Linear Regression Covariance Matrix; 48  
 Linear Regression Evaluation; 50  
 Linear Regression Formulas; 47  
 Linear Regression Min-Max; 56  
 Linear Regression Statistics; 49  
 Linear Regression with Robust Method; 55  
 LINEST; 46  
 Log Gamma function; 223  
 Log Relative Error; 219  
 Logarithm in any base; 61  
 Logarithm natural (Napier's); 61  
 LRE; 54

## M

Macro Sampler; 143  
 Macros for optimization on site; 239  
 Macros X-Edit; 39  
 Math expression strings; 235  
 MathParser; 237  
 matrix; 134  
 Matrix Addition; 126  
 Matrix Determinant; 127  
 Matrix Inverse; 126  
 Matrix LLT decomposition; 129  
 Matrix LU decomposition; 128  
 Matrix Modulus; 127  
 Matrix Multiplication; 126  
 Matrix Power; 128  
 Matrix Subtraction; 126  
 Maximum Common Divisor; 118  
 MCD; 118  
 MCM; 118  
 Minimum Common Multiple; 118

mjklRE; 219  
mjLRE; 54  
modular power; 120  
Modular Power; 120  
Multiplication; 26  
Multiprecision Expression Evaluation; 230  
Multiprecision Matrix operations; 134  
Multi-variables Interpolation; 187

### N

Next Prime; 120  
NextPrime; 120  
Non Linear Equation Solving; 216  
N-Root; 30

### O

Objective function; 239  
*ODE Multi-Steps*; 202  
ODE Runge-Kutta 4; 198  
ODE\_COR; 205  
ODE\_PRE; 205  
ODE\_RK4; 198  
Orthogonal polynomials; 102  
Orthogonal Polynomials; 102  
Orthogonal Polynomials evaluation; 103  
oscillating; 163

### P

Partial; 99  
PECE; 205; 207  
Perfect Square; 121  
permutation; 42  
Permutations; 42  
Polar Conversion; 110  
*Pollard*; 122  
Poly\_ChebyshevT; 103  
Poly\_ChebyshevU; 103  
Poly\_Gegenbauer; 103  
Poly\_Hermite; 103  
Poly\_Jacobi; 103  
Poly\_Laguerre; 103  
Poly\_Legendre; 103  
Poly\_Weight\_ChebyshevT; 106  
Poly\_Weight\_ChebyshevU; 106  
Poly\_Weight\_Gegenbauer; 106  
Poly\_Weight\_Hermite; 106  
Poly\_Weight\_Jacobi; 106  
Poly\_Weight\_Laguerre; 106  
Poly\_Weight\_Legendre; 106  
PolyAdd; 84  
PolyBuild; 91  
PolyBuildCfx; 93  
PolyCenter; 89  
PolyDiv; 85  
PolyInt; 94  
PolyInterp; 95  
PolyInterpCf; 95  
PolyMult; 84

POLYN; 80  
POLYN2; 98  
Polynomial addition; 84  
Polynomial building from roots; 91  
Polynomial building with multi-precision; 93  
Polynomial center; 89  
Polynomial coefficients; 83  
Polynomial derivatives; 82  
Polynomial division quotient; 85  
Polynomial division remainder; 85  
Polynomial evaluation; 80  
Polynomial interpolation; 95  
Polynomial multiplication; 84  
Polynomial roots radius; 90  
Polynomial shift; 89  
Polynomial solving; 94  
Polynomial subtraction; 85  
Polynomial System of 2nd degree; 97  
Polynomial writing; 84  
PolyRadius; 90  
PolyRem; 85  
PolyShift; 89  
PolySolve; 94  
PolySub; 85  
Polyterms; 88  
PolyTerms; 83  
predictor; 202  
Predictor; 203  
Predictor; 205  
Prime; 120  
Prime Numbers Generator; 123  
Prime\_Test\_Fermat; 123  
PrimeGenerator; 123  
Product; 29

### Q

Quadratic Mean; 43

### R

Raise to power; 30  
Rational Fraction approximation; 119  
Rectangular Conversion; 110  
RegLin\_Coeff; 44  
RegLin\_Eval; 50  
RegLinMM; 56  
RegLinRM; 55  
regression; 244  
Relative Rounding; 33  
RLCondCoef; 53  
Root Error Estimation; 72  
Rounding; 32

### S

Scalar Product; 127  
Scientific Format; 37  
Secant; 210  
Second Derivative; 213  
Serie\_trig; 177

Serie2D\_trig; 178  
 Series acceleration with  $\Delta^2$ ; 174  
 Series Evaluation; 173  
 sign; 34  
 Significant Digits count; 34  
 Similarity Transformation; 127  
 Simplex; 241  
 Sin; 65  
 Sine Integral Si(x); 227  
 SinIntegral; 227  
 Solve Linear Equation System; 130  
 Solve Linear Equation System with Iterative  
   method; 131  
 SortRange; 36  
 Split scientific format; 37  
 Square Delta Extrapolation; 132  
 Square Root; 30  
 Standard Deviation; 43  
 Sub-Tabulation; 52  
 Subtraction; 25  
 Sum; 29  
 sumDigits; 36  
 Summary of Linear Regressions; 51  
 SYSLIN\_ITER\_G; 131  
 SYSPOLY2; 97

T

Tan; 66  
*tanh-sinh transformation*; 148  
 trial division; 122  
 Trigonometric double serie; 178  
 Trigonometric series; 177  
 Truncating; 32

V

Variance; 43  
 Vector Inversion; 37  
 Vector Product; 129

W

Weight of Orthogonal Polynomials; 106

X

x2pi; 66  
 xabs; 31  
 xacos; 66  
 xacosh; 63  
 xadd; 25  
 xanglecompl; 67  
 xasin; 66  
 xasinh; 62  
 xatan; 66  
 xatanh; 63  
 xbeta; 224  
 xcdbl; 38  
 xcomb; 41  
 xcomb\_big; 225  
 xcomp; 34  
 xcos; 65  
 xcosh; 63  
 xcplxabs; 111  
 xcplxadd; 109  
 xcplxconj; 113  
 xcplxdiv; 110  
 xcplxExp; 112  
 xcplxinv; 112  
 xcplxLn; 112  
 xcplxmult; 109  
 xcplxneg; 113  
 xcplxpolar; 110  
 xcplxpow; 111  
 xcplxrect; 110  
 xcplxroot; 111  
 xcplxsub; 109  
 xcvexp; 37  
 xdec; 31  
 xDgt; 34  
 xdiv; 26  
 xdivint; 27  
 xdivrem; 27  
 xe; 62  
 xeu; 64  
 xeval; 29; 230  
 xevall; 230  
 xexp; 61  
 xfact; 41  
 xfact2; 41  
 xFib; 227  
 xFormat; 35  
 xfrac; 119  
 xFract\_Interp; 182  
 xFract\_Interp\_Coef; 182  
 xGamma; 222  
 xGammaF; 224  
 xGammaln; 223  
 xGammalog; 223  
 xGammaQ; 223  
 xGm; 221  
 xgmean; 42  
 xint; 31  
 xinv; 26  
 xlsOdd; 121  
 xlsSquare; 121  
 xLn; 61  
 xLn10; 62  
 xLn2; 62  
 xLog; 61  
 xLRE; 219  
 xMat\_BAB; 127  
 xMat\_LL; 129  
 xMat\_LU; 128  
 xMatAbs; 127  
 xMatAdd; 126  
 xMatDet; 127  
 xMatInv; 126  
 xMatMult; 126  
 xMatPow; 128

xMatSub; 126  
xMCD; 118  
xMCM; 118  
xmean; 42  
xmult; 26  
xneg; 25; 31  
xpi; 66  
xpi2; 66  
xpi4; 66  
xpow; 30  
xProdScal; 127  
xProdVect; 129  
xqmean; 43  
xRegLin\_Coeff; 44  
xRegLin\_Coeff; 46  
xRegLin\_Eval; 50  
xroot; 30  
xround; 32  
xroundr; 33  
xSerie2D; 176  
xsin; 65

xsinh; 62  
xsplit; 37  
xsqr; 30  
xstdev; 43  
xsub; 25  
xsum; 29  
xSYSLIN; 130  
xtan; 66  
xtanh; 63  
xtrunc; 32  
xUnformat; 35  
xvar; 43

### Z

Zero\_bisec; 209  
Zero\_sec; 210  
Zeros of Orthogonal Polynomials; 106  
Zeta; 228  
Zeta function; 228



© 2005, by Foxes Team  
ITALY  
Sept 2005