



FOXES TEAM

Reference for Xnumbers.xla

Numeric Calculus in Excel

REFERENCE FOR XNUMBERS.XLA

Numeric Calculus in EXCEL

Oct 2007

Index

About this Tutorial	8
Array functions	9
What is an array-function?	9
How to insert an array function	9
How to get the help on line.....	12
Xnumbers installation.....	13
How to install	13
How to uninstall	14
Installation troubles	14
Multiprecision Floating Point Arithmetic	15
Why using extended precision numbers?	15
Multiprecision methods.....	17
How to store long number	17
Functions	19
General Description.....	19
Using Xnumbers functions	19
Using extended numbers in Excel.....	20
Functions Handbook	21
Precision.....	22
Formatting Result.....	22
Arithmetic Functions	23
Addition.....	23
Subtraction.....	23
Multiplication	24
Division	24
Inverse.....	24
Integer Division	24
Integer Remainder	25
Sum	27
Product	27
Raise to power.....	27
Square Root.....	28
N th - Root	28
Absolute.....	28
Change sign.....	28
Integer part	29
Decimal part.....	29
Truncating.....	29
Rounding	29
Relative Rounding	30
Extended Numbers manipulation	31
Digits count.....	31
Significant Digits count.....	31
Numbers comparison.....	31
Extended number check	32
Format extended number.....	32
Check digits	32
SortRange.....	33
Digits sum	33
Vector flip.....	33
Scientific format	34
Split scientific format.....	34
Converting multiprecision into double	34
Macros X-Edit	35
Macro X-Converter	36
Statistical Functions	39
Factorial.....	39
Factorial with double-step	39

Combinations.....	39
Permutations.....	40
Arithmetic Mean.....	40
Geometric Mean.....	40
Quadratic Mean.....	40
Standard Deviation.....	40
Variance.....	41
Probability distributions.....	41
Univariate Statistic.....	46
Linear Regression Coefficients.....	47
Linear Regression - Standard Deviation of Estimates.....	49
Linear Regression Formulas.....	50
Linear Regression Covariance Matrix.....	51
Linear Regression Statistics.....	52
Linear Regression Evaluation.....	53
Polynomial Regression - Coefficient.....	54
Polynomial Regression - Standard Deviation of Estimates.....	54
Polynomial Regression Statistics.....	54
Macro - Regression.....	55
Linear Regression with Robust Method.....	59
Linear Regression Min-Max.....	60
NIST Certification Test.....	61
Transcendental Functions.....	62
Logarithm natural (Napier's).....	62
Logarithm for any base.....	62
Exponential.....	62
Exponential for any base.....	62
Constant e	63
Constant $\ln(2)$	63
Constant $\ln(10)$	63
Hyperbolic Sine.....	63
Hyperbolic ArSine.....	63
Hyperbolic Cosine.....	64
Hyperbolic ArCosine.....	64
Hyperbolic Tangent.....	64
Hyperbolic ArTangent.....	64
Euler constant γ	64
Trigonometric Functions.....	65
Sin.....	65
Cos.....	65
Tan.....	66
Arcsine.....	66
Arccosine.....	66
Arctan.....	66
Constant π	66
Complement of right angle.....	67
Polynomial Rootfinder.....	68
Input parameters.....	69
Output.....	70
How to use rootfinder macros.....	71
Integer roots.....	74
Multiple roots.....	76
Polynomial Functions.....	77
Polynomial evaluation.....	77
Polynomial derivatives.....	78
Polynomial coefficients.....	79
Polynomial writing.....	80
Polynomial addition.....	80
Polynomial multiplication.....	80
Polynomial subtraction.....	81
Polynomial division quotient.....	81
Polynomial division remainder.....	81
Polynomial shift.....	85
Polynomial center.....	85
Polynomial roots radius.....	86
Polynomial building from roots.....	87
Polynomial building with multi-precision.....	89
Polynomial Solving.....	90
Integer polynomial.....	90

Polynomial System of 2 nd degree.....	91
Bivariate Polynomial	92
Orthogonal Polynomials evaluation	97
Weight of Orthogonal Polynomials.....	99
Zeros of Orthogonal Polynomials.....	99
Coefficients of Orthogonal Polynomials.....	100
Complex Arithmetic and Functions	101
How to insert a complex number	101
Complex Addition.....	102
Complex Subtraction	102
Complex Multiplication.....	102
Complex Division	102
Polar Conversion	103
Rectangular Conversion	103
Complex absolute.....	103
Complex power.....	104
Complex Roots	104
Complex Log.....	105
Complex Exp	105
Complex inverse	105
Complex negative	105
Complex conjugate	105
Complex Sin	106
Complex Cos	106
Complex Tangent	106
Complex ArcCos.....	106
Complex ArcSin.....	106
Complex ArcTan	106
Complex Hyperbolic Sine.....	106
Complex Hyperbolic Cosine.....	107
Complex Hyperbolic Tan.....	107
Complex Inverse Hyperbolic Cos.....	107
Complex Inverse Hyperbolic Sin.....	107
Complex Inverse Hyperbolic Tan.....	107
Complex digamma.....	107
Complex Exponential Integral.....	107
Complex Error Function.....	108
Complex Complementary Error Function.....	108
Complex Gamma Function	108
Complex Logarithm Gamma Function	108
Complex Zeta Function.....	108
Complex Quadratic Equation	109
Number Theory	110
Maximum Common Divisor.....	110
Minimum Common Multiple	110
Rational Fraction approximation	111
Continued Fraction	112
Continued Fraction of Square Root	113
Check Prime	113
Next Prime	113
Modular Addition.....	114
Modular Subtraction.....	114
Modular Multiplication	114
Modular Division	114
Modular Power.....	114
Perfect Square.....	116
Check odd/even.....	116
Check Integer	116
Macro - Factorize.....	117
Batch Factorization with Msieve	118
Factorization function.....	120
Macro - Prime Numbers Generator.....	121
Prime Test	121
Diophantine Equation	123
Brouncker-Pell Equation	124
Euler's Totient function	125
Integer relation.....	126
Macro Integer Relation Finder	130
Linear Algebra Functions	133
Matrix Addition	133

Matrix Subtraction	133
Matrix Multiplication	133
Matrix Inverse	133
Matrix Determinant	133
Matrix Modulus	134
Scalar Product	134
Similarity Transform	134
Matrix Power	134
Matrix LU decomposition	135
Matrix LL^T decomposition	135
Vector Product	136
Solve Linear Equation System	136
Square Delta Extrapolation	137
Macro for Multiprecision Matrix Operations	139
Integrals & Series	141
Discrete Fourier Transform	141
Discrete Fourier Inverse Transform	143
Discrete Fourier Spectrum	144
Inverse Discrete Fourier Spectrum	144
2D Discrete Fourier Transform	145
2D Inverse Discrete Fourier Transform	145
Macro DFT (Discrete Fourier Transform)	146
Macro Sampler	149
Integral function	151
Zeros of integral function	154
Function Integration (Romberg method)	155
Function Integration (Double Exponential method)	156
Function Integration (mixed method)	158
Complex Function Integration (Romberg method)	160
Data Integration (Newton-Cotes)	161
Function Integration (Newton-Cotes)	163
Integration: symbolic and numeric approaches	165
Integration of oscillating functions (Filon formulas)	166
Integration of oscillating functions (Fourier transform)	168
Infinite Integration of oscillating functions	169
Double Integral	172
Macro for Double Integration	172
Macro for Triple Integration	175
Double integration function	178
Infinite integral	181
Double Data integration	183
Series Evaluation	186
Series acceleration with Δ^2	187
Complex Series Evaluation	188
Double Series	189
Trigonometric series	190
Trigonometric double serie	192
Discrete Convolution	193
Interpolation	195
Polynomial interpolation	195
Interpolation schemas	197
Interpolation with continued fraction	199
Interpolation with Cubic Spline	201
Cubic Spline 2nd derivatives	201
Cubic Spline Coefficients	203
2D Mesh Interpolation	204
Macro Mesh Fill	205
Differential Equations	211
ODE Runge-Kutta 4	211
ODE Multi-Steps	215
Multi-step coefficients tables	216
Predictor- Corrector	218
PECE algorithm of 2 nd order	218
ODE Predictor-Corrector 4	221
ODE Implicit Predictor-Corrector	223
Differential Systems	226
OD Linear System	226
High order linear ODE	227
ODE for integral function solving	230

Macro ODE Solver.....	232
Macro ODE - Slope Grid.....	233
Nonlinear Equations.....	235
Bisection.....	235
Secant.....	236
Derivatives.....	239
First Derivative.....	239
Second Derivative.....	240
Gradient.....	240
Jacobian matrix.....	241
Hessian matrix.....	241
Non-linear equation solving with derivatives.....	243
Conversions.....	245
Decibel.....	245
Base conversion.....	245
Multiprecision Base Conversion.....	246
Log Relative Error.....	247
Special Functions.....	249
Error Function $\text{Erf}(x)$	249
Exponential integral $\text{Ei}(x)$	249
Exponential integral $\text{En}(x)$	249
Euler-Mascheroni Constant γ	250
Gamma function $\Gamma(x)$	250
Log Gamma function.....	251
Gamma quotient.....	251
Gamma F-factor.....	252
Digamma function.....	252
Beta function.....	252
Incomplete Gamma function.....	253
Incomplete Beta function.....	253
Combinations function.....	253
Bessel functions of integer order.....	254
Cosine Integral $\text{Ci}(x)$	254
Sine Integral $\text{Si}(x)$	254
Fresnel sine Integral.....	255
Fresnel cosine Integral.....	255
Fibonacci numbers.....	255
Hypergeometric function.....	256
Zeta function $\zeta(s)$	256
Airy functions.....	257
Elliptic Integrals.....	257
Kummer confluent hypergeometric functions.....	258
Integral of sine-cosine power.....	259
Spherical Bessel functions of integer order.....	260
Formulas Evaluation.....	261
Multiprecision Expression Evaluation.....	261
Complex Expression Evaluation.....	265
Multiprecision Excel Formula Evaluation.....	266
Math expression strings.....	268
List of basic functions and operators.....	271
Function Optimization.....	274
Macros for optimization on site.....	274
Example 1 - Rosenbrock's parabolic valley.....	276
Example 2 - Constrained minimization.....	277
Example 3 - Nonlinear Regression with Absolute Sum.....	279
Example 4 - Optimization of Integral function.....	280
How to call Xnumbers functions from VBA.....	281
References & Resources.....	284
Analytical index.....	287

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 numerical calculus in standard and multiprecision floating point arithmetic up to 200 significant digits.

The main purpose of this document is to be a reference guide for the functions of this package, showing how to work with multiprecision arithmetic in Excel. The most part 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. Of course it speaks about math and numerical calculus but this is not a math book. You rarely find here theorems and demonstrations. You can find, on the contrary, many explaining examples.

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. Each function returning a matrix or a vector is an array functions. Function performing matrix operations such as inversion, multiplication, 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 to insert an array-function

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 vector solution **x**, that is an (3 x 1) array.

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



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

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

Excel interface showing the SYSLIN function being entered in cell G5. The formula bar shows **=SYSLIN(A5:C7,E5:E7)**. The spreadsheet shows matrix A in A5:C7 and vector b in E5:E7. The result x is shown in G5:E7.

Below the spreadsheet, the SYSLIN dialog box is shown, displaying the input ranges and the resulting solution vector x.

Mat: A5:C7 = {1;1;1;2;2;1;3;4}

v: E5:E7 = {4;2;3}

Solve Linear System.

v

Risultato formula = 6

OK Annulla

Xnumbers Tutorial

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						
9						
10						

Xnumbers Tutorial

- 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


	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		v1		v2		v3	
11		1		0		34	
12	34	-2	22	1		-46	
13		4		-1		114	
14							
15							
16							
17							

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



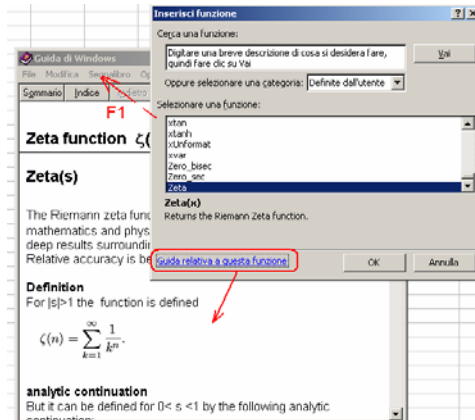
Functions returning optional values

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

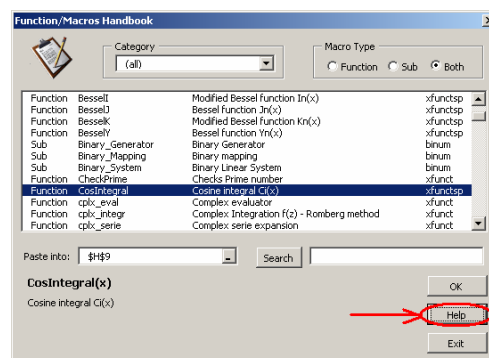
If you do not want to see this additional information simply select one cell and insert the function with the standard procedure. On the contrary, if you want to see also the extra information, you must select the extra needed cells and insert it as an array-function

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"](#)

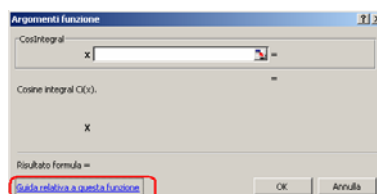


There is also another way to get the help-on-line. It is from the Xnumbers Function Handbook

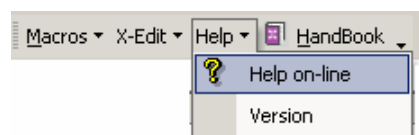


Select the function that you want and press the Help button

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

How to install

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

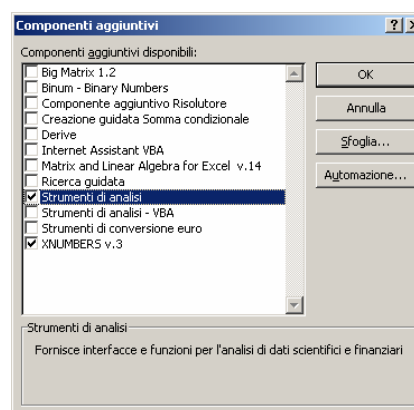
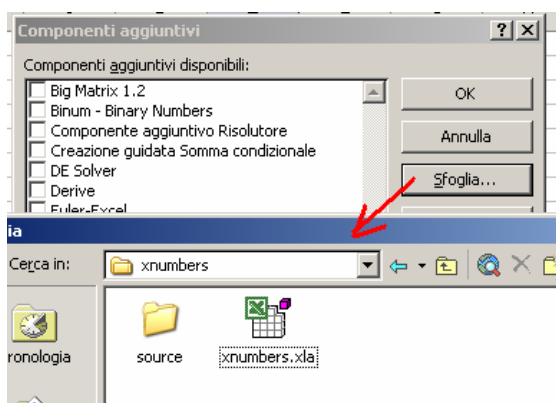
Addin file (It contains the Excel macros and functions)	Help file (It contains the help notes)	Handbook file (It contains the macros and functionn description list for the Xnumbers Handbook)
 xnumbers.xla	 Xnumbers.hlp	 xnumbers.csv

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,



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 checked addins 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".

Installation troubles

If you do not see the Xnumbers icon or the welcome popup the installation has gone wrong and probably it is due to same configuration parameter of your Windows / Excel environment

The default security setting of Excel XP 2002/2003 doesn't allow to install Xnumbers as is. To prevent errors it is necessary to open the Excel menu option:

Tools > Options > Security > Macro Security, and set: "*Average Level security*" and "*Trust Access to Visual Basic Project*".

Do not worry about it. Xnumbers contains only safe macros; they do not alter your PC in any way. If you want to remove Xnumbers simply delete all the files contained in the package.

In addition we have to point out that, nowadays, nobody use to hide virus, spy-software, trojan and related things in a macro Excel: this things belong to the prehistory of informatics! Nowadays, E-mail and Internet are by far the most important media.

Multiprecision Floating Point Arithmetic

Any computer having hardware at 32-bit can perform arithmetic operations with 15 significant digits¹. 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 the following 9th degree polynomial.

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

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:

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

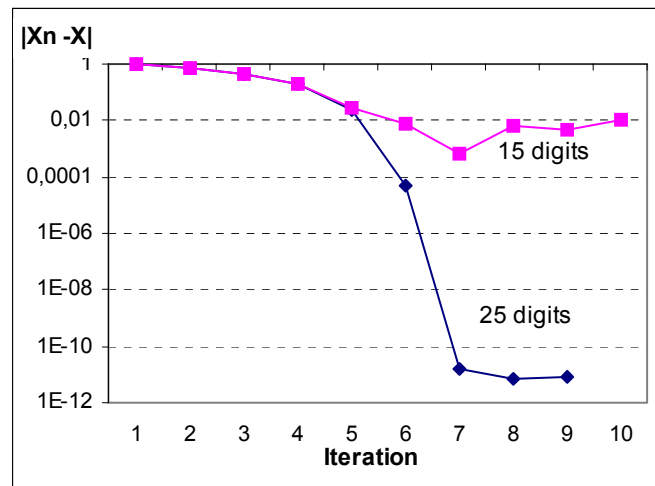
x _n	P(x) (15 digit)	P' (x) (15 digit)	-P/P'	x _n -x
32	120	428	0,280373832	1
31,71962617	43,77734375	158,9873047	0,275351191	0,7196262
31,44427498	15,69921875	60,93164063	0,257652979	0,444275
31,186622	4,78125	29,46289063	0,162280411	0,186622
31,02434159	0,65625	24,10644531	0,02722301	0,0243416
30,99711858	-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 computing. In fact, repeating the calculus of $P(x)$ and $P'(x)$ with 25 significant digits, we find the excellent convergence of this method.

¹ The basic structure is the IEEE-754 which allows for a 64 bit standard floating point double precision number. The later has a mantissa of 53 bits (one is the implied bit) which becomes equivalent to 15.4 digits. Excel reports a maximum of 15 digits. For a good, accurate note see "Excel Computation and Display Issue" by David. A. Heiser, <http://www.daheiser.info/excel/frontpage.html>

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. Especially it is very useful for numeric algorithms testing. 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:

A		B
456789	X	654321
105112		
601		

The schema below illustrates the algorithm adopted:

Xnumbers Tutorial

carry		A		B		C'		C
0	+	456789	x	654321	=	298886635269	=>	635269
298886	+	105112	x	654321	=	68777287838	=>	287838
68777	+	601	x	654321	=	393315698	=>	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 π , $\text{Log}(2)$, $\text{Log}(10)$ 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	23230505292219500000000000	=A2^4
4	2323050529221952581345121	=xmult(A2;xmult(A2;xmult(A2,A2)))
5		

As we can see, Xnumbers is powerful, but it does slow down the computation considerably. Therefore, use the multiprecision x-functions only when they are really necessary.

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 =xmuilt(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 xmuilt
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 "...".

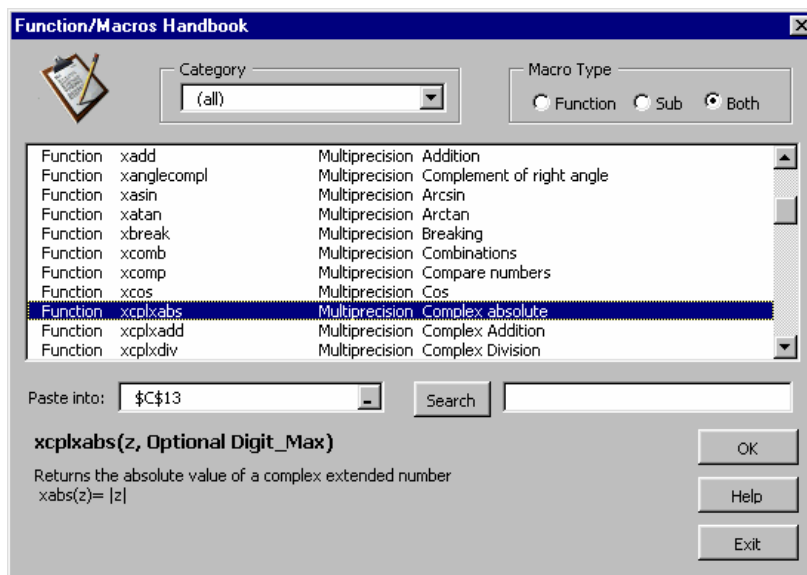
2469135780246913578 =xmuilt("1234567890123456789" , 2)

Functions Handbook



Xnumbers includes an application for searching and pasting the Xnumbers functions, cataloged by subject. This application 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 . It affects all multiprecision functions and macros.

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()**¹.

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

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

.

¹ 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
1		
2	0.129032258064516129032258064516	
3	0.333333333333333333333333333333	
4		
5	0.462365591397849462365591397849	=xadd(A2,A3)

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()**

	A	B	C
1			
2	0.129032258064516129032258064516		
3	0.333333333333333333333333333333		
4			
5	-0.204301075268817204301075268817	=xsub(A2,A3)	
6	-0.204301075268816870967741935484	=xadd(A2,-A3)	
7			

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 the exponential format.

	A	B	C	D
1	Digits Max	x		
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 $122 / 7 = 17,428571 \dots$ with 30 significant digits.

	A	B
1		
2	122	
3	7	
4	17.4285714285714285714285714285	=xdiv(A2,A3)
5		

Inverse

xinv(x, [Digit_Max])

It returns the inverse of an extended number. If $x = 0$, the function returns "?".

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

Integer Division

xdivint(a, b)

Returns the quotient of the integer division for $a > 0$, $b > 0$.

If $b = 0$ the function returns "?".

$$\text{xdivint}(a, b) = q, \text{ where: } a = b \cdot q + r, \text{ with } 0 \leq r < b$$

Xnumbers Tutorial

Another algorithm quite suitable for testing multiprecision accuracy is the π approximation by continuous fraction¹.

Initialize

$$X = 3^{1/2}, Y = 1/2, T = 6$$

Iteration

$$X = (2 + X)^{1/2}$$

$$Y = Y/X$$

$$T = 2T$$

$$P = Y \cdot T \cdot (5Y^6/112 + 3Y^4/40 + Y^2/6 + 1)$$

Accuracy: approximately 12 decimal digits every 5 iterations)

Below, step by step, a possible Excel arrangement:

	A	B	C	D	E	F
1	30	<=Digit_Max				
2			=xsqr(3;A1)			
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						
8		=xeval(\$B\$3;B5;\$A\$1)	=xdiv(C5;B6;\$A\$1)		=xeval(\$E\$3; C6;D6; \$A\$1)	
9						
10			=xmult(D5;2;\$A\$1)		=ABS(xsub(E5;E6;\$A\$1))	
11						

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.

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	1.9999958167178003620832744864	2.04530629116409511441305892323E-3	1536	3.14159265358979323762104105147	5.303E-17
14	9	1.99999985417917665522219647492	1.02265368033830454119296114608E-3	3072	3.14159265358979323844949364137	8.285E-19
15	10	1.99999973854477707409715031033	5.11326907013697501235819349761E-4	6144	3.14159265358979323846243791986	1.294E-20
16	11	1.99999993463619320041747774429	2.55663461862417314061649520852E-4	12288	3.14159265358979323846264017305	2.023E-22
17	12	1.99999998365904823334769327605	1.27831731975654740261724508019E-4	24576	3.14159265358979323846264333326	3.160E-24
18						
19						

The convergence to π is evident.

¹ This version, studied by David Sloan (2003), full of many arithmetic operations, permitted us to detect a very hidden bug in Xnumbers

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	$\Sigma =$	18.1202661917095175564128345609	=xsum(B2:B5)
8			

Product

xprod(v, [Digit_Max])

Returns the product of a vector of numbers.

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

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

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

Raise to power

xpow(x, n, [Digit_Max])

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

$xpow("0.39155749636098981077147016011", 90) = 1.9904508921478176508981155284E-7$

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

$xpow(122.5, 1000) = 122.5^{1000} = 1.36800819983562905361091390093E+2088$

Note the exponent +2088 of the third result. Such kind of numbers can be managed only with extended precision functions because they are out side of the standard limits for 32bit double precision.

For not integer power see the exponential functions xexp and xexpa

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:

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

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

Nth- Root

xroot(x, n, [Digit_Max])

Returns the nth 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 $\sqrt[9]{100}$ 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 $\text{xabs}(x) = |x|$

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

Change sign

xneg(x)

Returns the opposite of an extended number: $\text{xneg}(x) = -x$

Do not use the operator “-” (minus) for extended numbers. Otherwise Excel converts the extended numbers into double and changes its sign but, at 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 $\text{xneg}()$, 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, that is 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:

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

Truncating

xtrunc(x, [dec])

Returns the truncated number; the parameter "dec" sets the number of decimals to keep (default 0). It works like Excel function TRUNC. "dec" can be negative; in that case x is cut to the integer number, counting back from the decimal point. See the following examples.

Examples:

number	dec	number truncated
4074861.076055370173	-2	4074800
4074861.076055370173	-1	4074860
4074861.076055370173	0	4074861
4074861.076055370173	1	4074861
4074861.076055370173	2	4074861.07
4074861.076055370173	3	4074861.076

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	0.00000000000000001238521	16	0
1.238521E-17	0.00000000000000001238521	17	1.E-17
1.238521E-17	0.00000000000000001238521	18	1.2E-17
1.238521E-17	0.00000000000000001238521	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(trailing zeros are not significant).

[illegible]

Numbers comparison

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, the function assumes $b = 0$ for default and then it returns the $\text{sign}(a)$

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


Extended number check

xIsXnumber(x)

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

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

Format extended number

=xFormat(str, [Digit_Sep])

=xUnformat(str)

This function¹ separates an extended number in groups of digits by the separator 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 (default is 3)
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
```

Check digits

DigitsAllDiff(number)

This function² return TRUE if a number has all digits different.

```
DigitsAllDiff(12345) = TRUE  
DigitsAllDiff(123452) = FALSE
```

Argument can be also a string. Example

```
DigitsAllDiff(12345ABCDEF GHIM) = TRUE  
DigitsAllDiff(ABCD A) = FALSE
```

¹ 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.

² This function appears by the courtesy of Richard Huxtable

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 useful¹ function returns the digits sum of an integer number (extended or not)

`sumDigits(1234569888674326778876543) = 137`

Vector flip

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
1			
2	{=flip(A4:A8)}		
3			
4	123		100
5	44		1
6	-34		-34
7	1		44
8	100		123
9			

	F	G	H	I	J
	degree	coef		degree	coef
	0	112345		4	1
	1	-2345		3	8
	2	-124		2	-124
	3	8		1	-2345
	4	1		0	112345
	{=flip(F2:G6)}				

	A	B	C	D	E	F	G	H	I
1									
2	1	2	3	4	5	6			
3									
4	6	5	4	3	2	1			
5									

¹ This function appears by the courtesy of Richard Huxtable

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

xsplitt(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

	A	B	C	D
1	number	mantissa	expon.	
2	0.00002349475639834800002345981	2.349475639834800002345981	-5	=xsplitt(A2)
3	1.2993847566004505886E-26	1.2993847566004505886	-26	=xsplitt(A3)
4	-1.233456E-807	-1.233456	-807	=xsplitt(A4)

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.

Converting multiprecision into double

=xcdbl(x)

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

2	1	4
7	12	4
-4	0	8

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

0.3076923076923076923076923	-0.02564102564102564102564103	-0.1410256410256410256410256
-0.2307692307692307692307692	0.1025641025641025641025641	0.06410256410256410256410256
0.1538461538461538461538462	-0.01282051282051282051282051	0.05448717948717948717948718

Xnumbers Tutorial

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

	A	B	C	D	E	F	G
1							
2	2	1	4		0.3077	-0.0256	-0.141
3	7	12	4		-0.2308	0.1026	0.0641
4	-4	0	8		0.1538	-0.0128	0.0545
5							
6							
7							

=xcdbl(xmatinv(A2:C4))

Note that xcdbl affects only the output. The internal computing is always performed in multiprecision.



Macros X-Edit

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

Format	Separates groups of digits
Unformat	Removes the separator character
Double Conversion	Converts multiprecision numbers into standard double precision
Round	Rounding multiprecision numbers
Relative Round	Relative rounding multiprecision numbers
Mop-Up	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

Xnumbers Tutorial

Relative rounding - significant digits 15.

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

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.

Macro X-Converter

This macro¹ (*) converts a well formed Excel formula to the equivalent in terms of Xnumber multiprecision functions (xadd, xmult, etc.).

The advantage over x-evaluate is that the code can be debugged in normal excel using small values

and in a familiar form. When the spreadsheet works it is ' converted to using nested x-calls for the precision work. The intention is to tag the conversion onto the copy worksheet function ' so that you end up with a multi-precision copy of the original.

Its main features are:

- converts a range of cells directly on site or in a new worksheet.
- skips cells having functions not convertible.
- skips array functions except MINVERSE and MMULT that are substituted with the correspondent x-functions xMatInv and xMatMult.
- Errors (if any) are shown on the panel

The digits parameter can be set in 4 different ways:

- 1) an integer number i.e. 30
- 2) a fixed reference of a cell, i.e. \$A\$1
- 3) a name of a cell, i.e. "digits"
- 4) simply nothing. In that case all the x-functions use the internal default = 30

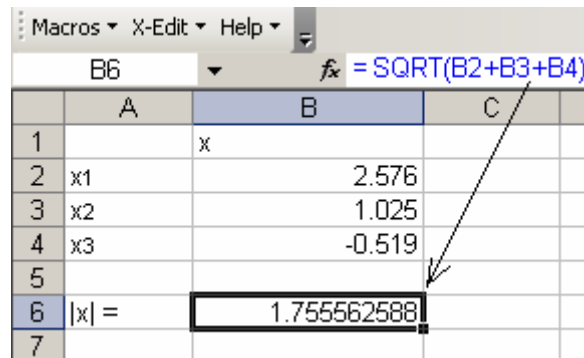
Of course not all the Excel Functions can be converted. The list of the Excel functions² converted is:

** , / , + , - , ^ , ABS , ACOS , ACOSH , ASIN , ASINH , ATAN , ATANH , AVERAGE , COMBIN , COS , COSH , EXP , FACT , INT , LN , LOG , MDETERM , MINVERSE , MMULT , MOD , PI , PRODUCT , ROUND , SIN , SINH , SQRT , STDEV , STDEVP , SUM , TAN , TANH , TRUNC , VAR , VARP*

¹ The conversion engine of this macro was originally developed by John Jones in an smart m4 macro language which makes the VB native j-code more simple (!) to write. It appears in this package thanks to his courtesy

² The functions are indicated with their original English names, that usually are different from the local names.

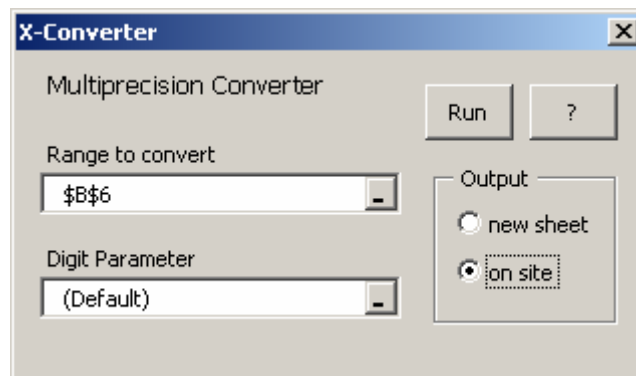
Example. Assume to have an Excel formula in the cell B6 calculating the norm of a 3dim vector like the following worksheet.



	A	B	C
1		x	
2	x1	2.576	
3	x2	1.025	
4	x3	-0.519	
5			
6	x =	1.755562588	
7			

We would transform this formula into a multiprecision one using the x-functions of the Xnumbers add-in

Select the cell B6 and start the macro X-Converter from the menu X-Edit

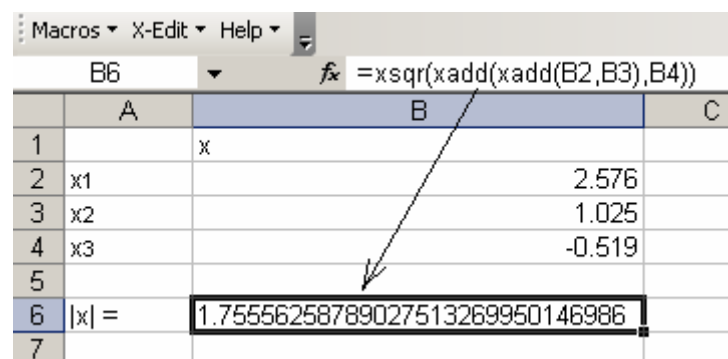


Range to convert: the range containing the one or more formulas to convert

Digit Parameter: specifies the global precision digits for all the functions under conversion. You can leave "(Default)" or blank meaning that all the function will use the internal default precision (usually 30 digits). But you can set a reference cell (example \$A\$1) where you have set your own precision. Or you can insert a name of a cell (example "mydigits") that you have defined. You can also insert directly a number (example 25)

Output Option: sets the output where the multiprecision function will be copied. Select "on site" if we want to convert the formula directly in the selected cell. Select "new sheet" if we want to perform the conversion in a new worksheet. The original formula will be preserved.

Let's select "on site" and click "run". The worksheet look will look like as the following



	A	B	C
1		x	
2	x1	2.576	
3	x2	1.025	
4	x3	-0.519	
5			
6	x =	1.75556258789027513269950146986	
7			

As we can see the original standard formula

=SQRT(B2+B3+B4)

in cell B6 has been substituted with the multiprecision formula

Xnumbers Tutorial

`=xsqr(xadd(xadd(B2,B3),B4)).`

working with the default precision digits (30)

Note that there are 3 nested functions in the converted formula.

The maximum number of nested functions in Excel is 8.

If an error raises during the cell conversion, the formula is left unchanged and the text is turned in red. The macro always shows a short list of the errors encountered during the conversion.

The macro works also with a range selection

Example. In the following worksheet we have same functions in the range A3:A18

	A	B
1	conversion test	
2		
3	1.414213562	=SQRT(2)
4	0.707106781	=1/B3
5	2.121320344	=B3+B4
6	-0.707106781	=B3-B5
7	1020030	1020030
8	-721270.13	=B6*B7
9	1.26869E+72	=B7^12
10	0.301029996	=LOG(2)
11	0.693147181	=LN(2)
12	0.03736095	=2.6*EXP(-3*B3)
13	0.385608145	=A11^2.6
14	3.535533906	{=A3:A4+A5:A6}
15	-2.22045E-16	{=A3:A4+A5:A6}
16	1.031538126	=SUM(B10:B12)
17	0.343846042	=AVERAGE(B10:B12)
18	15	=COUNTA(A3:A17)
19		

Note that the cell A18 contains the function COUNTA that are not the similar x-function

Note that the range A14:A15 contains an array function { ... }, thus the cells A14 and A15 are inseparable.

The cells A7 contain a simple constant

We want to convert them, where possible, in a new worksheet without affect the original worksheet.

For that, select all the range A3:A18 and start the X-Converter

Select "new sheet" and click "run". The new worksheet look like as the following

	A	B	C
1	conversion test		
2			
3	1.4142135623730950488016887242	=xsqr("2")	
4	0.707106781186547524400844362109	=xdiv("1",A3)	
5	2.1213203435596425732025330863	=xadd(A3,A4)	
6	-0.7071067811865475244008443621	=xsub(A3,A5)	
7	1020030	1020030	
8	-721270.130013714071314593274672	=xmult(A6,A7)	
9	1.26868948172893526046504690741E+72	=xpow(A7,"12")	
10	0.301029995663981195213738894724	=xlog("2")	
11	0.693147180559945309417232121458	=xLn("2")	
12	3.73609498351416191106264581265E-2	=xmult("2.6",xexp(xmult(xneg("3"),A3)))	
13	0.385608145012028619893099802122	=xexpbase(A11,"2.6")	
14	3.535533906	=A3:A4+A5:A6	
15	0	=A3:A4+A5:A6	
16	1.0315381260590681237415974743	=xsum(A10:A12)	
17	0.343846042019689374580532491436	=xmean(A10:A12)	
18	15	=COUNTA(A3:A17)	
19			

Note that the cell A7, A14, A15, A18 are unchanged

Note that the original cells A9 = B7^12 and A13 = A11^2.6 having the same operator symbol "^", are substituted with two different x-functions: xpow for integer power and xexpa for float power.

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 $\text{xGamma}(x)$. 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
```

	A	B	C	D
1	N	K	Combinations	digits
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			=xcomb(A10,B10)	=xdgts(C10)
12				

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 $k \gg 1000$) use the faster function $\text{xcomb_big}(n,k)$.

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.

$$M = \frac{\sum x_i}{n}$$

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.

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

Standard Deviation

xstdev(range, [Digit_Max])

xstdevp(range, [Digit_Max])

Return the deviation of n numbers, extended or not. Range is a range of cells. The optional parameter Digit_Max, from 1 to 200, sets the number of significant digits (default 30)

- **xstdev** returns the standard deviation

$$\sigma = \sqrt{\frac{\sum (x - \bar{x})^2}{n - 1}}$$

- **xstdevp** returns the population standard deviation

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

Variance

xvar(range, [Digit_Max])

xvarp(range, [Digit_Max])

Return the variance of n numbers, extended or not. Range is a range of cells. The optional parameter Digit_Max, from 1 to 200, sets the number of significant digits (default 30)

- **xvar** returns the standard variance

$$v = \frac{\sum (x - \bar{x})^2}{n - 1}$$

- **xvarp** returns the population variance

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

Probability distributions

Xnumbers contains several type of probability distribution functions

DSBeta (x, a, b, [dtype])	Beta distribution	$0 < x < 1$, $a > 0$, $b > 0$,
DSBinomial (k, n, p, [dtype])	Binomial distribution	k integer, n integer, $0 < p < 1$
DSCauchy (x, m, s, n, [dtype])	Cauchy (generalized) distribution	n integer, $s > 0$
DSChi (x, r, [dtype])	Chi distribution	r integer, $x > 0$
DSERlang (x, k, l, [dtype])	Erlang distribution	k integer, $x > 0$
DSGamma (x, k, l, [dtype])	Gamma distribution	$x > 0$, $k > 0$, $l > 0$
DSLevy (x, l, [dtype])	Levy distribution	$x > 0$, $l > 0$
DSLogNormal (x, m, s, [dtype])	Log-normal distribution	$x > 0$, $m \geq 0$, $s > 0$
DSLogistic (x, m, s, [dtype])	Logistic distribution	$x > 0$, $m \geq 0$, $s > 0$
DSMaxwell (x, a, [dtype])	Maxwell-Boltzman distribution	$x > 0$, $a > 0$
DSMises (x, k, [dtype])	Von Mises distribution	$k > 0$, $-\pi < x < \pi$
DSNormal (x, m, s, [dtype])	Normal distribution	$s > 0$
DSPoisson (k, z, [dtype])	Poisson distribution	k integer, $z > 0$
DSRayleigh (x, s, [dtype])	Rayleigh distribution	$x > 0$, $s > 0$
DSRice (x, v, s, [dtype])	Rice distribution (j=1 cumulative)	$x > 0$, $v \geq 0$, $s > 0$
DSStudent (t, v, [dtype])	Student distribution (j=1 cumulative)	v integer degree of freedom
DSWeibull (x, k, l, [dtype])	Weibull distribution (j=1 cumulative)	$x > 0$, k integer, $l > 0$

The optional parameter dtype = 0 (default) returns the density distribution f(x); dtype = 1 returns the cumulative distribution F(x).

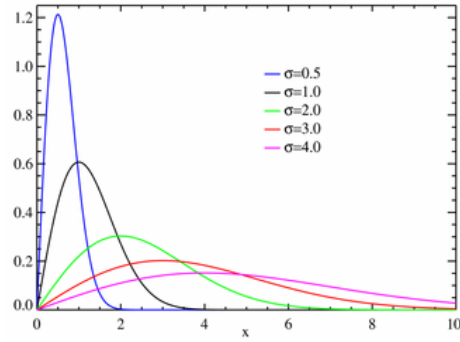
$$F(x) = P(\mathbf{x} < x) = \int_a^x f(t)dt$$

The lower limit "a" depends by the definition domain of the density function f(x).

Probability density definition

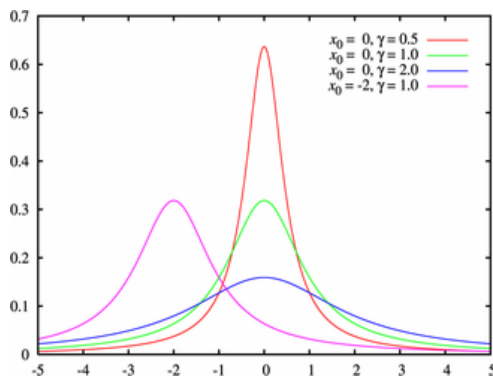
Rayleigh

$$f(x|\sigma) = \frac{x \exp\left(\frac{-x^2}{2\sigma^2}\right)}{\sigma^2}.$$



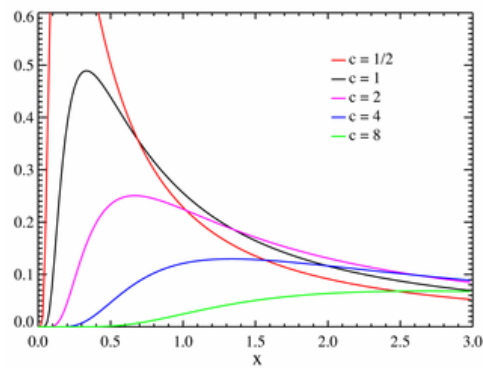
Cauchy

$$= \frac{1}{\pi} \left[\frac{\gamma}{(x - x_0)^2 + \gamma^2} \right]$$



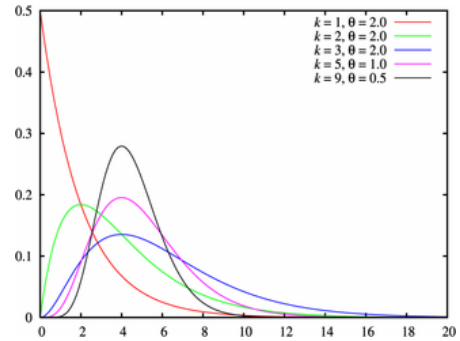
Lévy

$$f(x;c) = \sqrt{\frac{c}{2\pi}} \frac{e^{-c/2x}}{x^{3/2}}$$



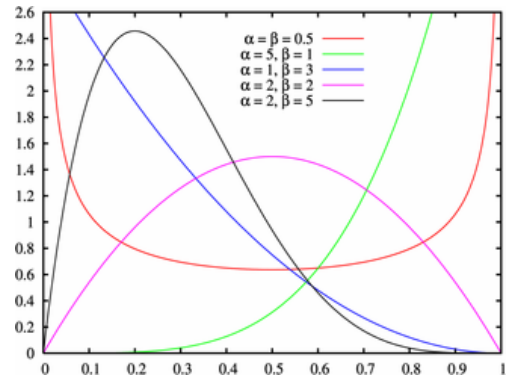
Gamma

$$f(x; k, \theta) = x^{k-1} \frac{e^{-x/\theta}}{\theta^k \Gamma(k)} \text{ for } x > 0$$



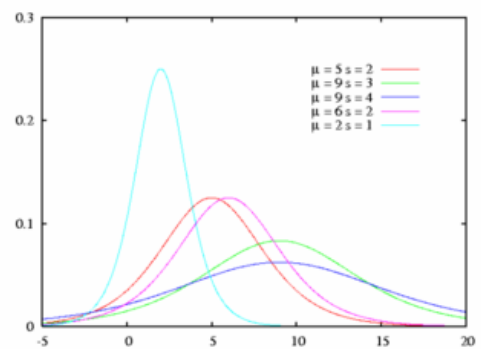
Beta

$$f(x; \alpha, \beta) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1}.$$



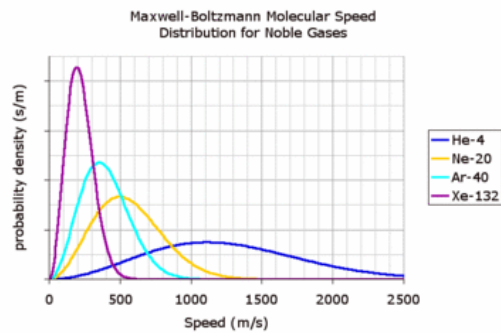
Logistic

$$f(x; \mu, s) = \frac{e^{-(x-\mu)/s}}{s (1 + e^{-(x-\mu)/s})^2}$$



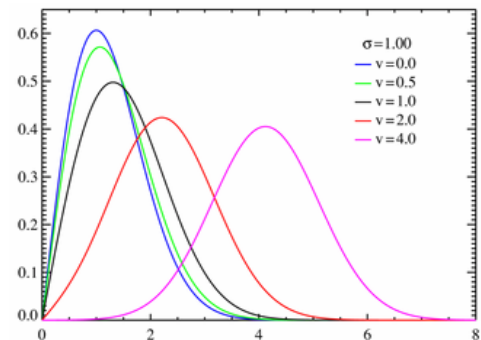
Maxwell-Boltzmann

$$f(v) = 4\pi \left(\frac{m}{2\pi kT} \right)^{3/2} v^2 \exp \left[\frac{-mv^2}{2kT} \right]$$



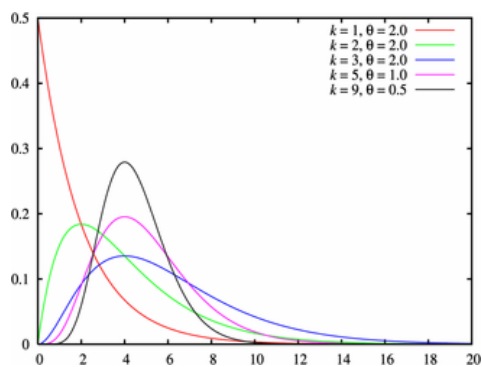
Rice

$$\frac{x}{\sigma^2} \exp \left(\frac{-(x^2 + v^2)}{2\sigma^2} \right) I_0 \left(\frac{xv}{\sigma^2} \right)$$



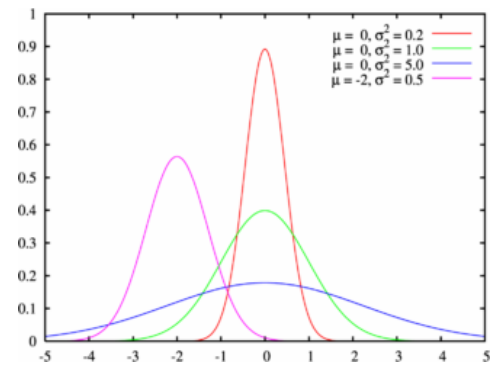
Erlang

$$f(x; k, \lambda) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!} \quad \text{for } x > 0.$$



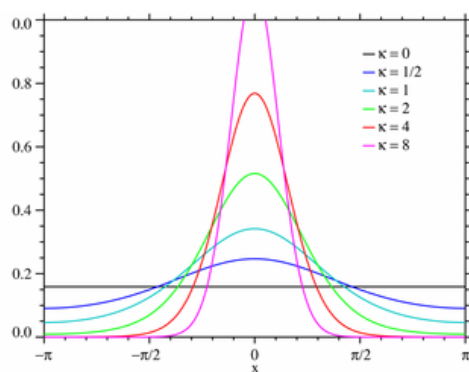
Normal

$$\frac{1}{\sigma\sqrt{2\pi}} \exp \left(-\frac{(x-\mu)^2}{2\sigma^2} \right)$$



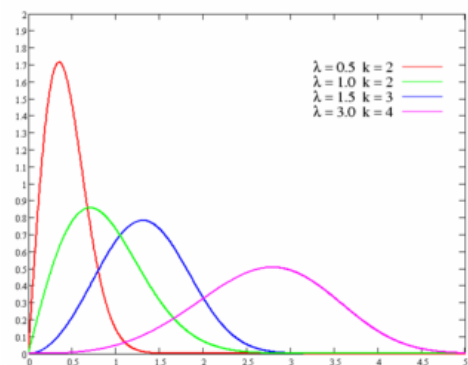
Von Mises

$$f(x|\mu, \kappa) = \frac{e^{\kappa \cos(x-\mu)}}{2\pi I_0(\kappa)}$$



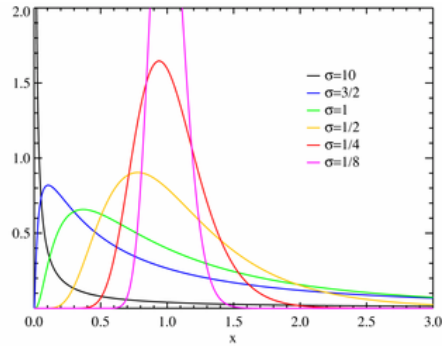
Weibull

$$f(x; k, \lambda) = \frac{k}{\lambda} \left(\frac{x}{\lambda} \right)^{k-1} e^{-(x/\lambda)^k}$$



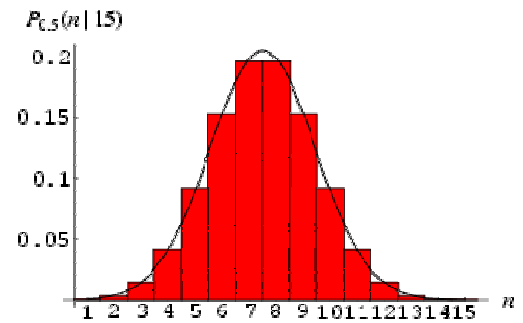
Log-normal

$$f(x; \mu, \sigma) = \frac{1}{x\sigma\sqrt{2\pi}} e^{-(\ln x - \mu)^2 / 2\sigma^2}$$



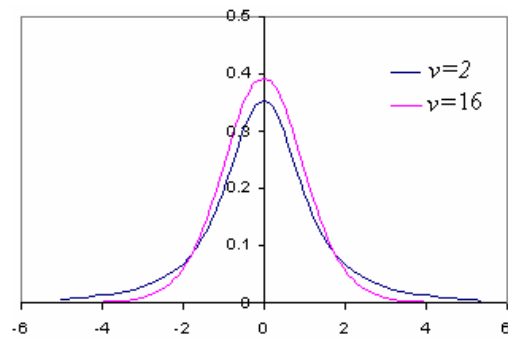
Binomial

$$f(k, n, p) = \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k}$$



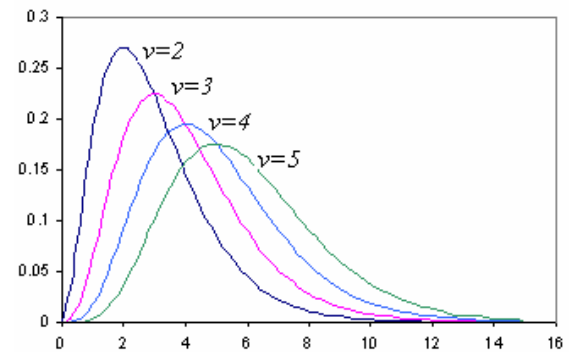
Student

$$f(t, \nu) = \frac{\Gamma(\frac{1}{2}(\nu+1))}{\sqrt{2\pi\nu} \cdot \Gamma(\frac{1}{2}\nu) (1 + \frac{1}{2}t^2)^{(\nu+1)/2}}$$



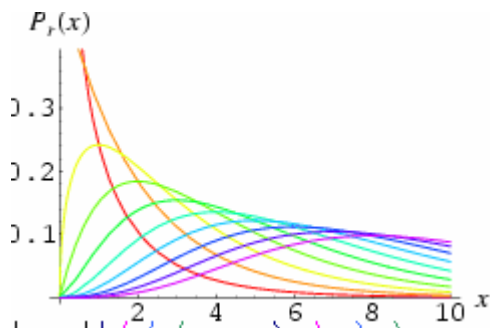
Poisson

$$f(x, \nu) = \frac{x^\nu e^{-x}}{\nu!}$$



Chi squared

$$f(x, \nu) = \frac{x^{\nu/2-1} e^{-x/2}}{2^{\nu/2} \Gamma(\frac{1}{2}\nu)}$$



Univariate Statistic

xstatis(range, [digit_max])

This function returns the univariate statistic summary of a range of data.
The statistics computed are:

Total of numerical elements of the range	N
Arithmetic Mean	$\frac{\sum x}{n}$
Standard Deviation	$\sqrt{\frac{\sum (x - \bar{x})^2}{n-1}}$
Popul. Standard Deviation	$\sqrt{\frac{\sum (x - \bar{x})^2}{n}}$
Autocorrelation lag1	$\frac{\sum_{i=1}^{n-1} (x_i - \bar{x})(x_{i+1} - \bar{x})}{\sum_{i=1}^n (x_i - \bar{x})^2}$

This function returns a vector of 5 elements. Use the ctrl+shift+enter key sequence for insert it.

	A	B	C
1	x		
2	4.8549	N =	13
3	4.7410	Mean =	5.11784615384615384615384615384
4	4.7286	St.Dev =	0.326007220307016654743770830397
5	5.3545	St.Dev.P =	0.313217605889345330137069891571
6	5.3548	Autocorr =	0.109153925269627793828582278206
7	4.7313		
8	5.2673		
9	4.6654		
10	5.4927		

{=xstatis(A2:A14)}

Linear Regression Coefficients

xRegLinCoef(Y, X, [DgtMax], [Intcpt])

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

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

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

DgtMax sets the precision (default 30).

Intcpt = true/false. If true (default) the algorithm calculates the intercept; otherwise the intercept is set to 0

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

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 **xRegLinCoef** 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	x	y		Coefficients
2	0.1	1991	a0 =	1990.9999102920727213168454672
3	0.2	1991.001046	a1 =	0.00528310949144214233068544754729
4	0.35	1991.001831		
5	0.4	1991.002092		
6	0.45	1991.002354		
7	0.6	1991.003138		
8	0.7	1991.003661		
9	0.8	1991.004184		
10	0.9	1991.004707		
11	1	1991.00523		
12	1.5	1991.007845		
13	1.8	1991.009414		
14	2	1991.01046		
15	3	1991.01569		

`{=xRegLinCoef(B2:B15,A2:A15)}`

Regression factor r^2

`=xRegLinStat(B2:B15,A2:A15,D2:D3)`

Let's see other examples

Example of univariate regression

	A	B	C	D	E
1	x	y		Coefficients	Standard Deviation of Estimate
2	1	581		341.142857142857142857142857143	101.221090607014649907881434777
3	2	1105		317.964285714285714285714285714	22.6337239353950208373564988561
4	3	1270			
5	4	1622			
6	5	1800			
7	6	2413			
8	7	2500			
9					
10					
11					

$\{=xRegLinCoef(B2:B8;A2:A8)\}$
 $\{=xRegLinErr(B2:B8;A2:A8;H2:H3)\}$
R-Squared
Standard Deviation
 $\{=xRegLinStat(B2:B8;A2:A8;H2:H3)\}$

Example of multivariate regression

	A	B	C	D	E	F
1	x1	x2	y		Coefficients	Standard Deviation of Estimate
2	126	-12	161		132.309698996655518394648829431	123.019270145209770704197670702
3	127	-7	247		1.78227424749163879598662207358	0.960580958961737000073465119491
4	128	0	359		16.2357859531772575250836120401	0.163664455051402049614429771709
5	129	15	606			
6	130	8	495			
7	131	13	576			
8						
9						
10						
11						

$\{=xRegLinCoef(C2:C7;A2:B7)\}$
 $\{=xRegLinStat(C2:C7;A2:B7;E2:E4)\}$
R-Squared
Standard Deviation
 $\{=xRegLinErr(C2:C7;A2:B7;E2:E4)\}$

Example of linear regression with intercept = 0

	A	B	C	D	E	F
1						
2	x1	x2	y		coefficients	standard deviation
3	126	-12	110	a0 = 0	0	0
4	127	-7	192	a1 = 2.39041181513326453093336753225	1.17589688276687264102746957126E-3	
5	128	0	306	a2 = 15.9401417771759702965663934394	1.45076092729588532465738405068E-2	
6	129	15	548			
7	130	8	438			
8	131	13	520			

$\{=xRegLinCoef(C3:C8;A3:B8, FALSE)\}$
 $\{=xRegLinErr(C3:C8;A3:B8;E3:E5, FALSE)\}$

Note that the x-regression functions always returns a vector of m+1 values $[a_0, a_1, \dots, a_m]$ independently if the intercept is 0 or not.

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							

Formulas shown in the image:

- `=xRegLinCoef(D2:D7,A2:C7)` (pointing to cell G2)
- `xRegLinStat(D2:D7,A2:C7,G2:G5)` (pointing to cell G6)

Linear Regression - Standard Deviation of Estimates

xRegLinErr(Y, X, Coeff, [DgtMax], [Intcpt])

Returns the standard deviation of the linear regression of estimate

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

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

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

DgtMax sets the digits precision of computing.

Intcpt = true/false. If true (default) the algorithm calculates the intercept deviation; otherwise intercep a₀ is set to 0

See above examples

Polynomial Regression

The same function finding the linear regression can easily be adapted to the polynomial regression. In the example below we will find the best fitting polynomial of 3rd 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					

	A	B	C	D	E	F
1	y	x	x ²	x ³		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		
7	3625	15	225	3375		
8	4378	16	256	4096		
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 =	0.99999999996282
13						
14						
15						

Formulas shown in the image:

- `=xRegLin_Coeff(A2:A12;B2:D12)` (pointing to cell F2)
- `REGR.LIN` (pointing to cell F2)
- `=B12^2` (pointing to cell C12)
- `=B12^3` (pointing to cell D12)
- `=flip(MatT(REGR.LIN(A2:A12;B2:D12)))` (pointing to cell F12)

The model for this data set is

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \text{ where } [a_0, a_1, a_2, a_3] \text{ are the unknown coefficients}$$

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 above.

The polynomial coefficients can be computed by xRegLinCoef.

The exact result is $y = 10 + x + x^2 + x^3$

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 $a = [a_1, a_2, \dots a_m]$ are the solution of the following linear system

$$[C] \cdot a = b$$

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

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

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 the 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 becomes nearly singular

Linear Regression Covariance Matrix

xRegLinCov (Y, X, Coeff, [DgtMax], [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 the (m+1) vector of the linear regression coefficients

Cross Covariance Matrix

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

Intercept calculated

Intercept = 0

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

$$X = \begin{bmatrix} x_{11} & \dots & x_{m1} \\ x_{12} & \dots & x_{m2} \\ \dots & \dots & \dots \\ x_{n2} & \dots & x_{mn} \end{bmatrix}$$

The covariance matrix C is

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

The covariance matrix C is

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

Where:

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

Where:

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

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])

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 $[a_0, a_1, a_2 \dots a_m]$.

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

= xRegLinEval(Coeff, X)

Evaluates the multivariate linear regression in multi precision arithmetic.

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

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

The functions return the linear combination.

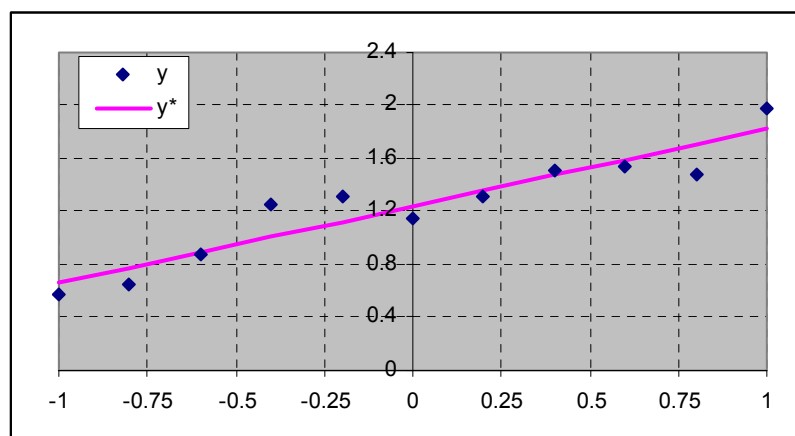
$$y = a_0 + a_1x_1 + a_2x_2 + \dots a_nx_n$$

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	0.657727	a0	1.24	
-0.6	0.88	-0.8	0.65	0.774182	a1	0.582272727	
-0.4	1.25	-0.6	0.88	0.890636	=xcdbl(xRegLinCoef(B2:B12,A2:A12))		
-0.2	1.32	-0.4	1.25	1.007091			
0	1.14	-0.2	1.32	1.123545	=xcdbl(xRegLinEval(\$E\$2:\$E\$3,A12))		
0.2	1.31	0	1.14	1.24			
0.4	1.51	0.2	1.31	1.356455			
0.6	1.54	0.4	1.51	1.472909			
0.8	1.48	0.6	1.54	1.589364			
1	1.98	0.8	1.48	1.705818			
		1	1.98	1.822273			

In this worksheet, each value of linear regression y^* is computed by the function xRegLinEval. The regression coefficients are computed by xRegLinCoef. The results are converted in double by the function xcdbl.

Selecting the range A1:C12 and plotting the data we get the following regression graphs



Polynomial Regression - Coefficient

xRegPolyCoef(Y, X, Coeff, [Degree], [DgtMax], [Intcpt])

Computes the polynomial regression with the least squares method in multi precision arithmetic.

$$y = a_0 + a_1x + a_2x^2 \dots + a_mx^m$$

with $m > 1$

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

X is a vector (n x 1) of the independent variable

Degree is the degree $m > 1$ of the polynomial

DgtMax sets the precision (default 30).

Intcpt = true/false. If true (default) the algorithm calculates the intercept; otherwise the intercept is set to 0

The function returns the coefficient vector $[a_0, a_1, a_2 \dots a_m]$

Example. Find the 3rd degree polinomial fitting the given data set (xi, yi)

	A	B	C	D	E	F
1	x	y				
2	1	24	{=xRegPolyCoef(B2:B14,A2:A14,3,,L10)}		{=xRegPolyErr(B2:B14,A2:A14,D4:D7,3,,L10)}	
3	2	56	coefficients		standard deviation	
4	3	108	b0	4.03156194738238620665496023566	3.93572228656587965379608410321	
5	6	444	b1	12.4511740382652405748905676935	0.696169136290454934758454633511	
6	7	636	b2	4.02434861263165101820651131901	5.05779995028438221550849706861E-2	
7	9	1176	b3	0.999765891057977359253494613619	6.42560050080475987018775204523E-4	
8	10	1500				
9	11	1964				
10	24	16446				
11	67	319596		R-squared	0.999999988155770127987863535579	
12	-1	-4		Standard deviation	11.0894643064806792369955295039	
13	-2	-12				
14	-8	-354			{=xRegPolyStat(B2:B14,A2:A14,D4:D7,3)}	
15						
16						

Polynomial Regression - Standard Deviation of Estimates

xRegPolyErr (Y, X, Coeff, [Degree], [DgtMax], [Intcpt])

Returns the standard deviation of the polynomial regression estimate

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

X is a vector (n x 1) of the independent variable

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

DgtMax sets the precision (default 30)..

Intcpt = true/false. If true (default) the algorithm calculates the intercept; otherwise the intercept is set to 0

see above example

Polynomial Regression Statistics

xRegPolyStat (Y, X, Coeff, [Degree], [DgtMax], [Intcpt])

Computes the R-squared factor of the polynomial regression and the standard deviation of the regression.

R^2	R-Squared coefficient of the polynomial regression
$S_{y,x}$	Standard Deviation of the polynomial regression

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

X is a vector (n x 1) of the independent variable

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

DgtMax sets the precision (default 30).

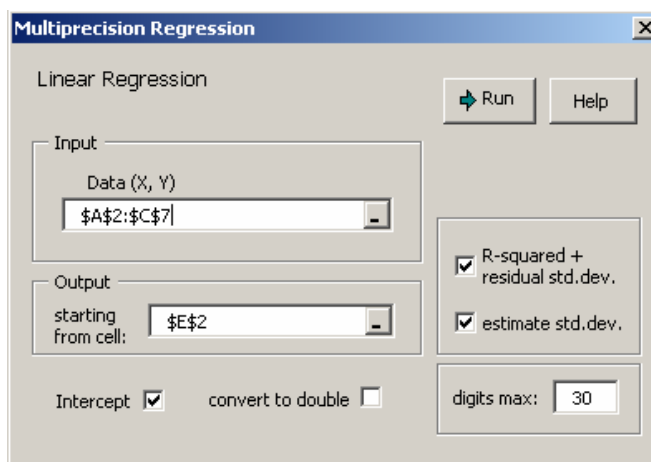
Intcpt = true/false. If true (default) the algorithm calculates the intercept deviation; otherwise this is set to 0

If you wanto to see the standard deviation select two cells and give the CTRL+SHIFT+ENTER sequence.

Macro - Regression

Xnumbers contains two macros performing the linear and polynomial regression in multiprecision arithmetic. From the Xnumbers menu, select

- **Macro \ Regression \ Linear**
- **Macro \ Regression \ Polynomial**



Data XY is an array containing the column of the variable X at the left with the adjacent column of Y at the right. For a m-multivariate regression also the columns of X must be exactly m. Select this range before starting the macro. In this case the input box will be automatically filled

Data XY for univariate and polynomial regression

	A	B
1	x	y
2	-12	161
3	-7	247
4	0	359
5	15	606
6	8	495
7	13	576
8		

Data XY for multivariate regression

	A	B	C
1	x1	x2	y
2	126	-12	161
3	127	-7	247
4	128	0	359
5	129	15	606
6	130	8	495
7	131	13	576
8			

Output cell indicates the starting upper left cell of the output area.

Intercept. If checked the macro calculates the intercept otherwise it is set to 0

Convert to double. Converts the multiprecision in double before the output

R-squared + residual std. dev. Calculate the correspondent statistics

Estimate std. dev. Calculate the standard deviation of estimate

Digits Max. from 1 to 200, sets the precision (default 30)

Degree. The panel of the Polynomial Regression macro is similar to the Linear except for the input degree box that allows to chose the polynomial degree.

degree:

An example of the regression output is.

	A	B	C	D	E	F
1	x1	x2	y		Coefficients	Standard Deviation of Estimate
2	126	-12	161	a0 =	132.309698996655518394648829431	123.019270145209770704197670702
3	127	-7	247	a1 =	1.78227424749163879598662207358	0.960580958961737000073465119491
4	128	0	359	a2 =	16.2357859531772575250836120401	0.163664455051402049614429771709
5	129	15	606			
6	130	8	495		R-Squared	0.999944298440953362332372934176
7	131	13	576		Standard Deviation	1.74672810873406813031254410884
8						

To obtain this result follow this steps:.

- Select the range A2:C7 and start the macro.
- Select all the option boxes and deselect the "double conversion" in order to see all the 30 digits.

Note. The text has been added by hand only for clarity. The macro do not write them. We do it.

The coefficient a_0 (intercept) will be always output. If the intercept is switched off, the result in the cell F2 will be 0

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 11 values with step 0.1, from 0 to 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

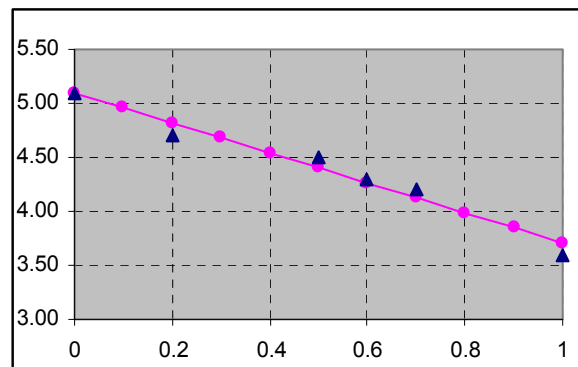
First of all, we find the linear regression coefficients

$[a_0, a_1]$

Then we re-calculate the values

$$y_i = a_0 + a_1 h, \quad i = 1 \dots 10, \quad h = 0.1$$

	A	B	C	D	E	F	G	H
1	Table A			Coefficients			Table B	
2	x	y		a0	5.0953125		x	y
3	0	5.1		a1	-1.390625		0	5.09531
4	0.2	4.7					0.1	4.95625
5	0.5	4.5					0.2	4.81719
6	0.6	4.3					0.3	4.67813
7	0.7	4.2					0.4	4.53906
8	1	3.6					0.5	4.4
9							0.6	4.26094
10							0.7	4.12188
11							0.8	3.98281
12							0.9	3.84375
13							1	3.70469



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 back-transformed 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 useful for increasing the global accuracy of the linear regression

Linear Regression with Robust Method

RegLinRM(x, y, [Method])

This function¹ 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

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

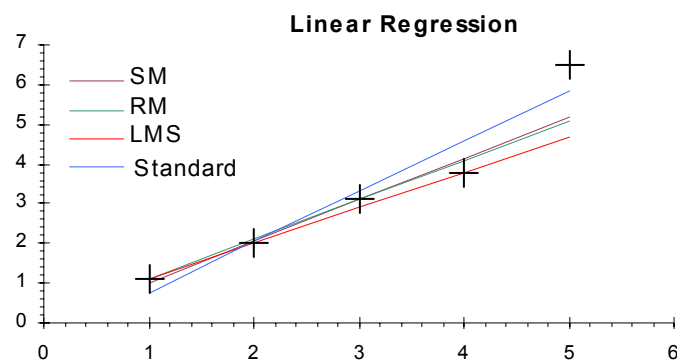
Use CTRL+SHIFT+ENTER to paste it.

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)



¹ 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

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

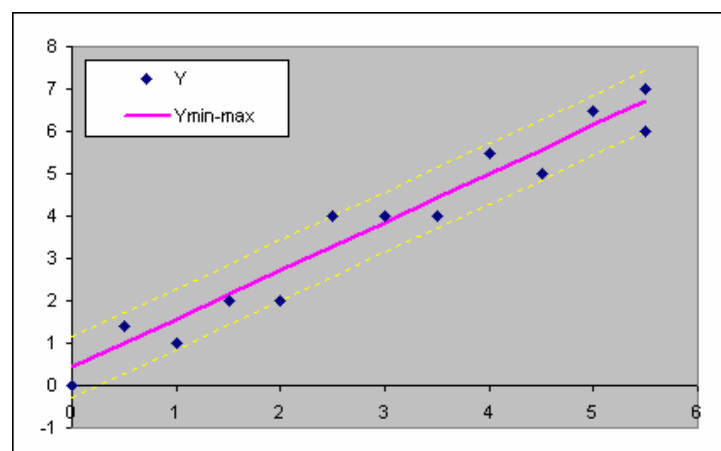
Example. Find the better fitting line that minimize the absolute error

	A	B	C	D	E	F	G
1	x	y	Ymin-max	Error			
2	0	0	0.42857	0.42857		Coefficients	
3	0.5	1.4	1.00000	-0.40000			
4	1	1	1.57143	0.57143		a0	a1
5	1.5	2	2.14286	0.14286		0.428571	1.142857
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		ErrMax =	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). ($E_{\max} \cong 0.7$)

NIST Certification Test

The multiprecision regression functions of Xnumbers have been tested against the NIST linear regression data sets. The table below gives the LRE results from xRegLinCoef and xRegPolyCoef functions

NIST test for linear regression (row data sets)					
StRD Datasets	Level of difficulty	Model of class	Coeff.	Function	LRE
Norris	low	Linear	2	xRegLinCoef	15.0
Pontius	low	Quadratic	3	xRegPolyCoef	15.0
NoInt1	Average	Linear	1	xRegLinCoef	15.0
Filip	high	Polynomial	11	xRegPolyCoef	15.0
Longley	high	Multilinear	7	xRegLinCoef	15.0
Wampler1	high	Polynomial	6	xRegPolyCoef	15.0
Wampler2	high	Polynomial	6	xRegPolyCoef	15.0
Wampler3	high	Polynomial	6	xRegPolyCoef	15.0
Wampler4	high	Polynomial	6	xRegPolyCoef	15.0
Wampler5	high	Polynomial	6	xRegPolyCoef	15.0

The table below gives the LRE results on the NIST test univariate data sets obtained from xStats function

NIST test for univariate data sets						
Dataset	Category	Difficulty	Size	Mean	Stand. Dev.	Autocor. Coef.
PiDigits	Univariate	1	5000	15	15	15
Lottery	Univariate	1	218	15	15	15
Lew	Univariate	1	200	15	15	15
Maveo	Univariate	1	50	15	15	15
Michelso	Univariate	1	100	15	15	15
NumAcc1	Univariate	1	3	15	15	15
NumAcc2	Univariate	2	1001	15	15	15
NumAcc3	Univariate	2	1001	15	15	15
NumAcc4	Univariate	3	1001	15	15	15

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

xexpa(x, [a], [Digit_Max])

Returns the exponential of x in any in base $xexpa(x, a) = a^x$
 The arguments "a" and "x" may be either normal or extended numbers, with $a > 0$.
 If the base "a" is omitted the function assumes $a = 10$.

$$2^{1.234} = xexpa(1.234, 2) = 2.3521825005819296401155858555$$

$$10^{-0.54} = xexpa(-0.54) = 0.288403150312660594239196924659$$

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 constant γ

=xeu([Digits_Max])

Returns the Euler-Mascheroni constant γ
(The same constant returned 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.

`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.

`xcos(1.5)` = 7.07372016677029100881898514342E-2

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

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

`COS(89.99999995)` = `COS(1.570796326)` = 7.94896654250123E-10

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

`xcos(1.570796326)` = 7.94896619231321E-10

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

angle α	α (deg)	<code>xcos(α)</code>	<code>COS(α)</code> 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	89.9999999971950	4.89661923132169E-12	4.8965888522954E-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 `xcos` function keeps its accuracy.

Tan

xtan(a, [Digit_Max])

Returns the tangent of a $\text{xtan}(a) = \tan(a)$
 The argument a, in radians, may be either a normal or an extended number.

Arcsine

xasin(a, [Digit_Max])

Returns the arcsine of a $\text{xasin}(a) = \arcsin(a)$
 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

xacos(a, [Digit_Max])

Returns the arccosine of a $\text{xacos}(a) = \arccos(a)$
 The arccosine is defined between 0 and π
 The argument a, where $|a| \leq 1$, may be either a normal or an extended number.

Arctan

xatan(a, [Digit_Max])

Returns the arctan of a $\text{xatan}(a) = \arctan(a)$
 The arctan(a) is defined between

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

Constant π

These functions return the following multiples of π

xpi([Digit_Max])	xpi = π
xpi2([Digit_Max])	xpi2 = $\pi/2$
xpi4([Digit_Max])	xpi4 = $\pi/4$
x2pi([Digit_Max])	x2pi = 2π

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

Example. Compute the Hermite-Ramanujan constant with 36 significant digits

$$e^{\pi\sqrt{163}}$$

`xexp(xmult(xpi(36),xsqr(163,36),36),36) = 262537412640768743.99999999999250005`

Complement of right angle

xanglec(a, [Digit_Max])

Returns the complement of angle a to the right angle

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

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

Example:

`xanglec(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:

`xanglec(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.

Xnumbers provides several macros based on the following polynomial rootfinder algorithms.

RootFinder JT	Jenkins and Traub algorithm (translated in VB from the original rpoly FORTRAN 77 subroutine)
RootFinder GN	Generalized Newton-Raphson method
RootFinder ADK	Aberth, Durand, Kerner algorithm
Rootfinder RF	Ruffini's method for real integer roots.
Rootfinder LB	Lin-Bairstow algorithm
Rootfinder SK	Siljak algorithm
Rootfinder LA	Laguerre algorithm

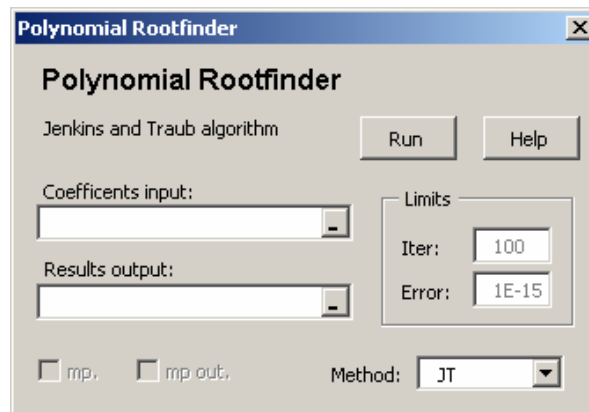
These macros are able to find, in a few seconds, all the roots - real or complex - of a dense polynomial up to 15th - 20th degree, in double or multi-precision. It is remarkable that sometimes the results has shown in an exact way, even if the computation is intrinsically approximate.

The characteristics of each rootfinder are reassumed in the following table

Macro	Roots	Coefficients	Arithmetic
RootfinderJT	Complex	Real	Standard
RootfinderGN	Complex	Real	Multi-precision
RootfinderDK	Complex	Complex	Multi-precision
RootfinderRF	Real, Integer	Real, Integer	Multi-precision
RootfinderLB	Complex	Real	Standard
RootfinderSK	Complex	Complex	Multi-precision
RootfinderLA	Complex	Real	Standard

Input parameters

The polynomial rootfinder interface is simple and straight.



Method:

JT	Jenkins-Traub	Standard	Real coefficients
GN	Gen. Newton-Raphson	Multiprecision	Real coefficients
ADK	Aberth-Durand-Kerner	Multiprecision	Complex coefficients
RF	Ruffini	Standard	Real Integer coefficients
LB	Lin-Bairstow	Standard	Real coefficients
SK	Siljak	Multiprecision	Complex coefficients
LA	Laguerre	Standard	Real coefficients

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

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

RootfinderDK and Siljak can also accept complex coefficients. In that case the input is an (n x 2) array. Examples of possible input 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					
2	-249	-74	2	878	polynomial				
3	88	21	3	-45	x^16-6817x^8+1679616				
4	-15	-2	4	-15					
5	1	0	5	1					

Remarks.

The formula string 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 precision.

Output

The rootfinder macros output their results in the following simplified layout

The roots 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

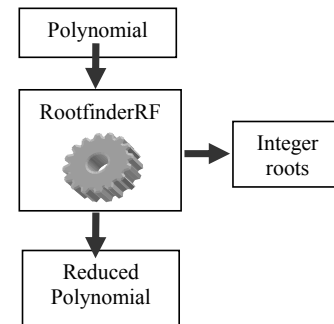
degree	coeff.	Real	Imm	Rel. Err.
0	-125	2	1	3.78E-08
1	225	2	-1	1.13E-07
2	-170	2	1	1.86E-07
3	66	2	-1	5.03E-08
4	-13	5	0	2.00E-18
5	1			
				0.046875
				elab. time (sec)

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

Integer Rootfinder output

The macro RootfinderRF 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

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

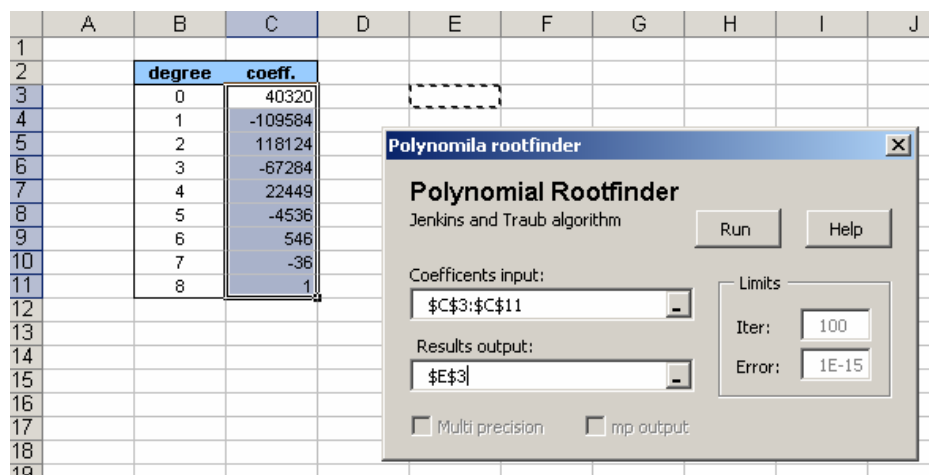
can be factorized as

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

How to use rootfinder macros

Using polynomial rootfinder macros is simple. Select the polynomial coefficients 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".

Example. Select the range C3:C11 and start the RootfinderJT . The coefficients input-box is filled with C3:C11 and the result-output box is filled with the cell E3.



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
	degree	coeff.		Real	Imm	Rel. Err.
	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 16th degree polynomial

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

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

The screenshot shows the Xnumbers interface with a polynomial rootfinder dialog box open. The dialog box is titled "Polynomial Rootfinder" and contains the following fields and buttons:

- Polynomial Rootfinder** (Title)
- Durand-Kerner-Aberth algorithm** (Text)
- Run** and **Help** buttons.
- Coefficients input:** A text box containing $x^{16}-6817x^8+1679616$.
- Results output:** A text box containing x^5 .
- Limits:**
 - Iter:** 50
 - Error:** 1E-15
- ☐ **Multi precision** and ☐ **mp output** checkboxes.

The background shows a table with the following data:

	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						

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:

$$er_i = |x_i - \tilde{x}| / |x_i| \quad \text{for } |x_i| > 0$$

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 high precision of the correspondent root. On the contrary, larger values (for examples 1E-3, 1E-5) indicates a "difficult" roots that require an extra investigation.

For example assume to find the root of the following 6th 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 RootfinderDK and RootfinderSK can solve also complex polynomials. Example: find the roots of the following polynomial with complex coefficients

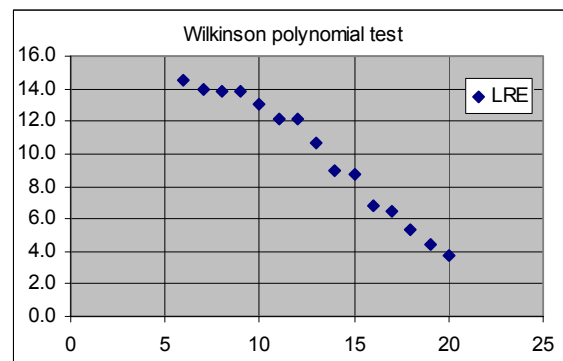
$$P(z) = (-12 + 4i) + 4z + (15 - 5i)z^2 - 5z^3 + (-3 + i)z^4 + z^5$$

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 $z = \pm 1$, $z = \pm 2$, $z = 3 - j$. Observe that the results are shown in exact mode even if the computation is intrinsically approximated.

A polynomial of n degree, having as roots the first $1, 2, \dots, n$ integers, belongs to the Wilkinson class that, as known, is ill-conditioned. This dense polynomial is usually assumed 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 20th degree, we have good 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 16th degree Legendre's polynomial

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

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			

(remember that the higher degree coefficients are at bottom)

In the last column there are the errors estimation 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 values mean a good accuracy, larger values often (but not always) indicate a poor approximation

Integer roots

In applied science it's rarely to come across in polynomials having exact integer roots.

Nevertheless, they are frequent in math, didactical examples and algorithm testing .

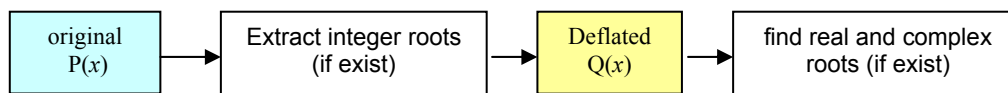
Xnumbers has a dedicated special macro for finding the integer real roots of a polynomial. It uses the Ruffini's method with the QD algorithm for roots isolation.

This method is generally less efficient then JT or DK but it can gain in accuracy.

The roots found by this method have no round-off errors so the deflated polynomial is exact.

Therefore, in that case, the root-finding-deflating process 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 by solving directly the given polynomials.



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 8th 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

Select the range B2:B10 and start the macro for extracting the integer roots and deflated polynomial

The screenshot shows an Excel spreadsheet with columns A through K. Column A contains the degree (a0 to a8) and column B contains the coefficients (8678880 to 10). Column D shows the integer roots (5, 6, 7, 8, 9) and column E shows the polynomial remainder (-574, 457, -118, 10, 0.03125). The 'Polynomial Integer Rootfinder' dialog box is open, showing the Ruffini's method, coefficients input range (\$B\$2:\$B\$10), results output range (\$D\$2), and limits (Iter: 1000). The 'Multi precision' checkbox is unchecked.

Xnumbers Tutorial

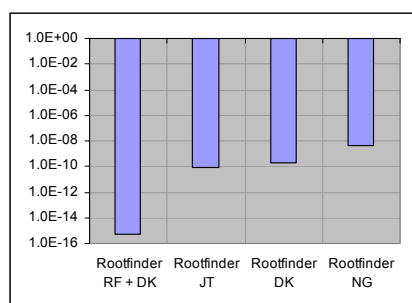
The original polynomial is now cracked into the following factors

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

Now let's find the roots of the following 3rd 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 they are).



Global roots accuracy versus the solving methods:

Rootfinder RF + DK
Rootfinder JT
Rootfinder DK
Rootfinder NG

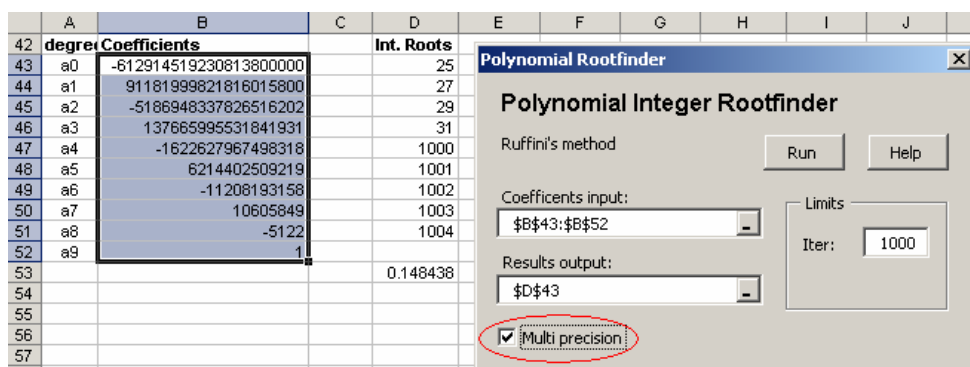
The multiprecision should be used when the coefficients exceed 15 digits (remember that the coefficients must be exact in order to extract the exact integer roots)
Let's see the following 18th degree polynomial having the roots

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



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.

Multiple roots

The macro "Factors" performs the decomposition of a polynomial with multiple roots into smaller factors having all single roots. It uses the GCD method with the Euclid' algorithm

Example, the polynomial:

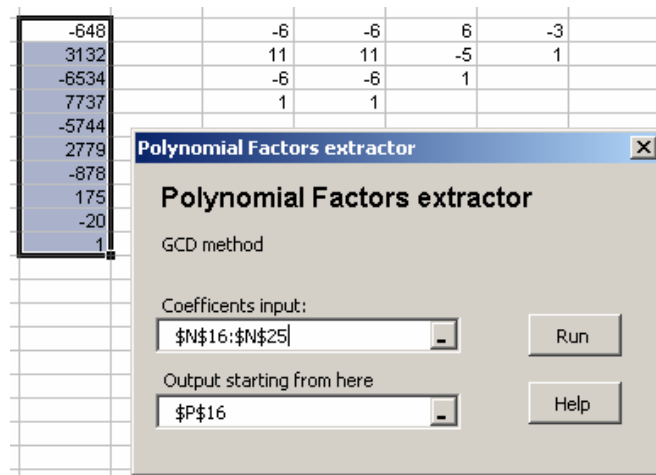
$$-648 + 3132x - 6534x^2 + 7737x^3 - 5744x^4 + 2779x^5 - 878x^6 + 175x^7 - 20x^8 + x^9$$

Has the roots $x = 1$ ($m = 2$), $x = 2$ ($m = 3$) and $x = 3$ ($m = 4$)

It can be decomposed into the product of the following factors

$$(-3 + x)(6 - 5x + x^2)(-6 + 11x - 6x^2 + x^3)^2$$

Using this macros is simple. Select the polynomial coefficients and start the macro. All the input and output cells are automatically filled. Press Run



Each factors contains only single roots and thus can be solved with high precision by any rootfinder macro¹.

¹ For further details see "Non Linear Equations", Foxes Team, 2006

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 precision. 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	degree	coeff		re	im
2	a0	4	z =	4	-2
3	a1	2			
4	a2	-5		re	im
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	degree	coeff		
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				

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	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) =	-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.

$$P(z) = a_0 + a_1z + a_2z^2 + \dots a_nz^n$$

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

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 precision. 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	degree	coeff	z =	3			
2	a0	3	P(z) =	45	=DPOLYN(D1;\$B\$2:\$B\$5;0)		
3	a1	2	P'(z) =	35	=DPOLYN(D1;\$B\$2:\$B\$5;1)		
4	a2	1	P''(z) =	20	=DPOLYN(D1;\$B\$2:\$B\$5;2)		
5	a3	1	P'''(z) =	6	=DPOLYN(D1;\$B\$2:\$B\$5;3)		

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

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

	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		re	im
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	Polynomial root with Newton's method		
2	$x^7 - 5x^6 + 64x^3 - 8000$		
3	=xsub(A5;xdiv(B5;C5))	=POLYN(A5;\$A\$2;30)	=DPOLYN(A5;\$A\$2;1;30)
4			
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	20425.000000000000000000000924
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		24-5x+3x^2+x^3	24	-5	3	1	
13							
14			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	1000+200x+x^6			
14				
15				

PolyWrite(A10:C11)

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

PolyWrite(A16:E16;"t")

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.

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 monovaryable 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 monovaryable 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 monovaryable polynomial strings.

Hermite's and Cebychev's polynomials

By the basic operations we can build any other polynomial.

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

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

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

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

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

Xnumbers Tutorial

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

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

$$U_n(x) = k_{n-2} \cdot (2n-1)x$$

$$a_n = k_{n-1} \cdot (n-1)$$

$$V_n(x) = U_n(x) \cdot L_{n-1}(x) - a_n \cdot L_{n-2}(x)$$

$$b_n = n \cdot k_{n-1} \cdot k_{n-2}$$

$$c_n = \text{GCD}(b_n, \text{coef}(V_n))$$

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 $(k_n, L_n(x))$

$$k_n = \frac{b_n}{c_n}$$

$$L_n(x) = \frac{1}{c_n} V_n(x)$$

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	Legendre's Polynomials							
2				=xMCD(polyterms(D6);E6)			=E6/F6	
3	n	U(x)	a	V(x)	b	c	k	L(x)
4	0						1	1
5	1						1	x
6	2	3x	1	-1+3x^2	2	1	2	-1+3x^2
7								
8								
9								
10								

Formulas shown in the image:

- $=G5*A5$ (for cell C6)
- $=A6*G5*G4$ (for cell D6)
- $=PolyMult(G4*(2*A6-1),"x")$ (for cell D6)
- $=polysub(PolyMult(B6;H5);PolyMult(C6;H4))$ (for cell D6)
- $=polydiv(D6;F6)$ (for cell H6)

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] $\Rightarrow 1$

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	Legendre's Polynomials							
2								
3	n	U(x)	a	V(x)	b	c	k	L(x)
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

Xnumbers Tutorial

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

n	k	L(x)
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 to 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	degree	Coefficients				degree	Coefficients
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							

{=PolyShift(B3:B9;D4)}

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. His 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 **B_x** 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

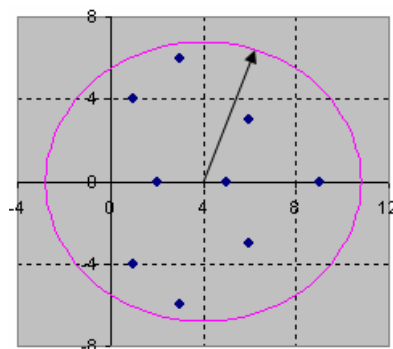
	A	B	C	D
1	degree	coefficients	radius	center
2	a0	-3098250	6.7882251	4
3	a1	4116825		
4	a2	-2427570	=PolyRadius(B2:B11)	
5	a3	916272		
6	a4	-244674	=PolyCenter(B2:B11)	
7	a5	46934		
8	a6	-6430		
9	a7	608		
10	a8	-36		
11	a9	1		

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, picking-up a random polynomial, the most part of the roots are found inside the circle but it is also possible to find same roots outside the circle with 1% of probability.

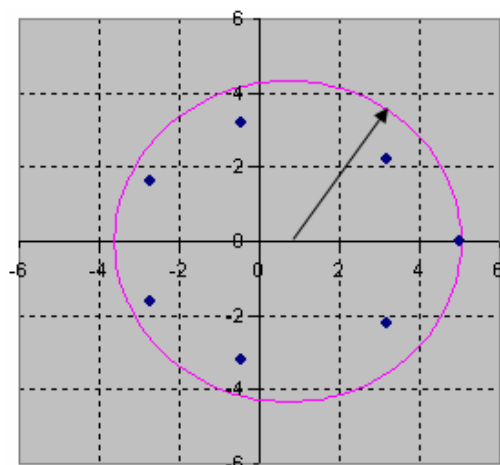
Example: compute the root circle of the polynomial:

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

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 an (n x 1) 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	Xre	Yim	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	Xre	Yim	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	

Complex roots for complex polynomial

If the complex roots are not symmetrical, the polynomial has both real and imaginary part.

	A	B	C
1	Xre	Yim	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 it needs.

	A	B	C
1	Xre	Yim	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 returns the vector of polynomial coefficients if you select more than two vertical cells. It is useful for higher degree polynomial

	A	B	C	D	E
1	x real	x imm		degree	coefficients
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 9th degree polynomial having the 9 roots in the range A2:B10.

Select the range E2:E11 and insert the function PolyBuild with the CTRL+SHIFT+ENTER sequence.

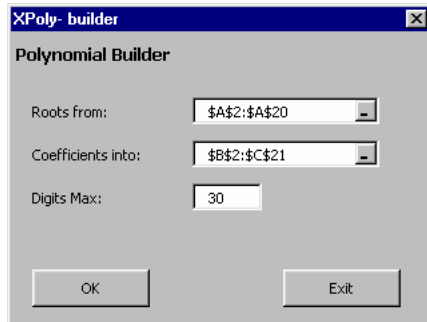
	A	B	C	D	E
1	x real	x imm	degree	coeff. re.	coeff im.
2	9	0	a0	540	-405
3	5	0	a1	-393	351
4	2	1	a2	127	-79
5	3	-6	a3	-19	5
6			a4	1	0

If complex roots are not conjugate, the polynomial has complex coefficients. This function returns also the imaginary part of the coefficients. Simply select the range D2:E6.

Polynomial building with multi-precision

PolyBuildCfx()

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



For using this macro select the range containing 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).

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 up to 10th 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	Degree	Coefficients		REAL	IMM
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					
15					

=PolySolveJT(B2:B12)

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: Given 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 may lose accuracy. See the example below

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:

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

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

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 System of 2nd degree

=SYSPOLY2(Poly1, Poly2)

Solves a system of two 2nd 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 2nd 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 2nd 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

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 value - real or complex - of a bivariate polynomial $P(x, y)$.

The parameter "Polynomial" is an expression strings. Valid examples are:

$13+x+y^2-y+x^2+2x*y$, x^2+y^2-10 , $8x*y+y^2+2x-2$, $10+4x^6+x^2*y^2$

Note: the product symbol "*" can be omitted except for the $x*y$ mixed terms

The third optional parameter DgtMax sets the multiprecision. If missing, the computation is performed in faster double precision.

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

$$x = (2.5 + j \ 1.11803398874989)$$

$$y = (-2.5 + j \ 1.11803398874989)$$

Verify that it is a good approximation of the polynomial root

	A	B	C
1			
2	P(x, y) = $x^2+2x*y+y^2+x-y$		
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..

$$\frac{N(x)}{D(x)} = Q(x) + \frac{R(x)}{D(x)} = Q(x) + \sum F_i$$

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 $D(x)$ is determined from the poles of the fractions itself. In fact, p is 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. A solution can also be arranged in Excel 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 finding the fraction coefficients A_i . One of the most straight and elegant is due to Heaviside that, for a real single root, simply states:

$$A_i = \frac{N(p_i)}{D'(p_i)}$$

where $D'(x)$ is the derivative of $D(x)$

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							=polyn(D7,\$B\$4:\$B\$7)		=G7/H7	
11							=dpolyn(D7,\$C\$4:\$C\$8,1)			
12										

Therefore, the searched 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_1 and c_1 , 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 with the unknown 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	B1	=	123/650
1/17	1/17	1/34	1/34		C1		9/34
1/5	1/10	2/45	1/45		B2		3/10
3/5	1/5	3/58	1/58		C2		63/290

Solving this linear system by any method that we like, for example by SYSLIN, 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 always valid for any compatible value of x

A possible arrangement for solving this problem in Excel 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		=D4^2+E4^2		
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:
 F4: =D4^2+E4^2
 F7: =-2*D4
 G4: =polysolve(C4:C8)

Then we compute the polynomials N, D, D1, D2 for each values of x by the function polyn. We get the 4x5 table visible 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:
 M7: =polyn(J7,\$B\$4:\$B\$7)

From this table we get the complete system matrix in the following way.

	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:
 P7: =J7/M7
 Q7: =1/M7
 R7: =J7/N7
 S7: =1/N7
 T7: =K7/L7

The 4 x 4 linear system can be solved by any method that you want. For example by matrix inversion as shown in the example.

	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:
 W7: =MMULT(MINVERSE(P4:S7),T4:T7)

Orthogonal Polynomials

Orthogonal polynomials are a class of polynomials following the rule:

$$\int_a^b w(x) p_m(x) p_n(x) dx = \delta_{mn} \cdot c_n$$

Where m and n are the degrees of the polynomials, w(x) is the weighting function, and c(n) is the weight. δ_{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]	$(1-x^2)^{-1/2}$	$\begin{cases} \pi & \text{for } n = 0 \\ \pi/2 & \text{for } n \neq 0 \end{cases}$
Chebyshev polynomial of the second kind	[-1, 1]	$(1-x^2)^{1/2}$	$\pi/2$
Gegenbauer polynomial	[-1, 1]	$(1-x^2)^{\alpha-1/2}$	$\frac{2^{1-2\alpha} \pi \cdot \Gamma(n+2\alpha)}{n!(n+\alpha) \cdot \Gamma^2(\alpha)}$ for $\alpha \neq 0$ $2\pi/n^2$ for $\alpha = 0$
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

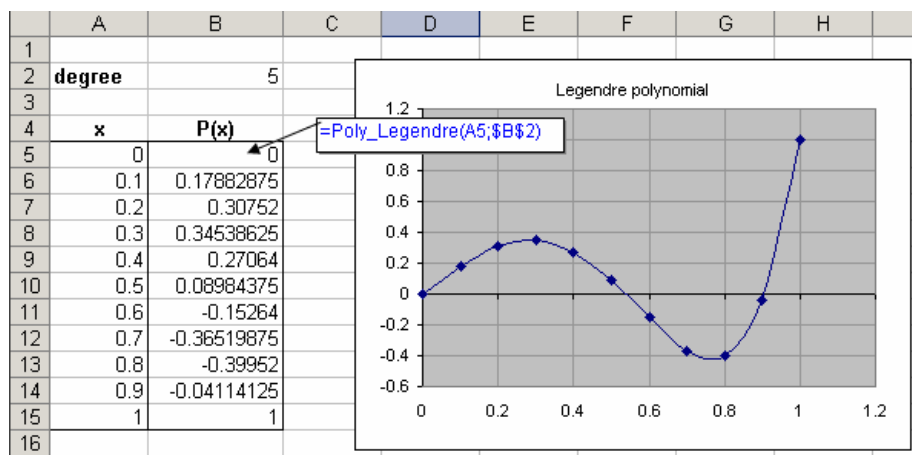
$$h_n = \frac{2^{\alpha+\beta+1}}{2n+\alpha+\beta+1} \frac{\Gamma(n+\alpha+1) \cdot \Gamma(n+\beta+1)}{n! \Gamma(n+\alpha+\beta+1)}$$

Orthogonal Polynomials evaluation

This set of functions¹ 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 6th degree, for $0 \leq x \leq 1$, with step $h = 0.1$

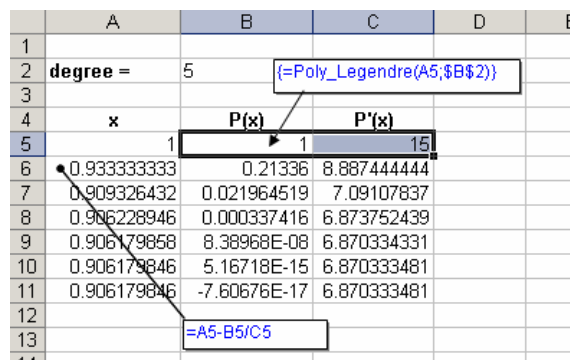


As we can see we have insert **Poly_Legendre** as a standard function, because in this exercise we do not need the derivative information

Example. Find the greatest zero of the 5th degree Legendre polynomial

We can use the Newton-Raphson method, starting from $x = 1$, as shown in the worksheet arrangement.

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



¹ Many thanks to Luis Isaac Ramos Garcia for his great contribution in developing this software

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
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 $c(n)$ of each orthogonal polynomial $p(x, n)$

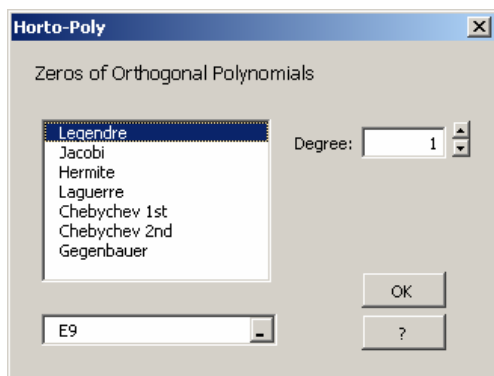
$$c_n = \int_a^b w(x) [p_n(x)]^2 dx$$

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 popular 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

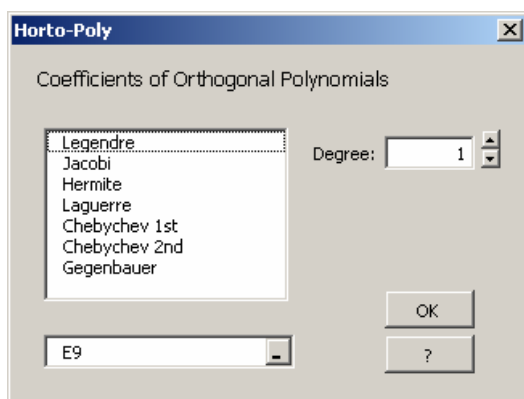
	A	B	C
1	Zeros of Laguerre polynomials		
2	m =	0	
3	Degree =	6	
4	i	root	poly
5	1	15.98287398	2.11164E-13
6	2	9.837467418	1.51048E-14
7	3	5.775143569	2.00361E-15
8	4	2.992736326	3.10805E-17
9	5	1.188932102	-3.25261E-18
10	6	0.222846604	6.95696E-18
11			

This is an example of output for a
Laguerre polynomial of 6th degree
(m = 0)

Note: the format is added for clarity.
The macro does not do this

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
Coeff. of Laguerre polynomials		
weight =	1	
Degree =	4	
kd =	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 ortho-polynomial 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 and 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											

For entering complex functions you must select two cells, insert the complex 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 characterized by long non-integer 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	ρ	θ (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

$$x = \rho \cos(\theta)$$

$$y = \rho \sin(\theta)$$

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^{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^{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

$$\rho = \sqrt{x^2 + y^2} \quad , \quad \theta = \text{atan}\left(\frac{y}{x}\right)$$

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		fx {=cplxroot(B3:C3;3)}			
	A	B	C	D	E	F
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:

$$\rho = \sqrt{x^2 + y^2} \quad , \quad \theta = \operatorname{atan}\left(\frac{y}{x}\right)$$

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

cplxsinh(z)

Returns the hyperbolic cosine of a complex number
Parameter “z” can be a real or complex number (two adjacent cells)

Complex Hyperbolic Tan

cplxsinh(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

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$$

Complex Complementary Error Function

cplxerfc(z)

Returns the complementary error function for a complex number

$$\operatorname{erfc}(z) = 1 - \operatorname{erf}(z)$$

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.

For $|s| > 1$ the function is defined as:

$$\zeta(n) = \sum_{k=1}^{\infty} \frac{1}{k^n}.$$

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

$$z^2 + (9 - 2i)z + 4 + i = 0$$

	A	B	C	D	E	F
1						
2					{=cplxEquation2(B4:C4,B5:C5,B6:C6,20)}	
3						
4	a	1	0		real	im.
5	b	9	-2		-8.5918392090132821071947	2.2219443982515907319261
6	c	4	1		-0.4081607909867178928053	-0.2219443982515907319261
7						

Of course the function can also works with real coefficients equations

$$z^2 + 2z + 3 = 0$$

	A	B	C	D	E	F
7						
8					{=cplxEquation2(A11:B11,C1,20)}	
9						
10	a	b	c		real	im.
11	1	2	3		-1	1.4142135623730950488016
12					-1	-1.4142135623730950488016
13						

Number Theory

Maximum Common Divisor

xMCD(a1, [a2])

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])

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		x		
2		831402		
3		1339481		
4		291720		
5		1650649		
6		255255		
7		1205776		
8		2387242		
9				
10	GCD =	2431		
11	LCM =	9967402918352880		
12	Π =	3.94008780758332018835283346146E+41		

Note that LCM may easily overcome the standard precision limit even if its arguments are all in standard precision.

Rational Fraction approximation

xFract(x, [Digit_Max])

Fract(x, [ErrMax])

Returns the fractional approximation of a non-integer number x, the functions returns a vector of two integer 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 uses the continued fraction expansion¹

$$N_0 = 0, N_1 = 1$$

$$D_0 = 1, D_1 = 0$$

$$N_{i+1} = a_i \cdot N_i + N_{i-1}$$

$$D_{i+1} = a_i \cdot D_i + D_{i-1}$$

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 the number 0.126. The function returns the solution:
N = 63 , D = 500

$$0.126 \approx 63 / 500$$

	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

¹ from *The art of Computer Programming*, D.E.Knuth, Vol.2, Addison-Wesley, 1969

Continued Fraction

xFractCont(x, [Digit_Max])

FractCont(x)

These functions perform the continued fraction expansion of a decimal number x

Usually the continued fraction is written in the compact form $x = [a_0, a_1, a_2, a_3, \dots]$

Each step gives the rational fraction $f_i = s_i / t_i$, where the fractions f_i converge to the number x

$$x = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \dots}}}$$

Both the above functions return an array containing the integer values [ai, si, ti].

The algorithm stops when the fraction approximates x with the maximum precision allowed, that is 10^{-15} and $10^{-(\text{DgtMax})}$ respectively for the standard and the multiprecision function.

Example. Compute the continued fraction expansion of the number $e^2 = 7.38905\dots$ with standard precision (error < $1\text{E-}15$)
As we can see, the continued fraction of the transcendental real number e^2 can be written as:
 $e^2 = [7, 2, 1, 1, 3, 18, 5, 1, 1, 6, 30, 8, \dots]$

while the correspondent approximating fractions are

$$e^2 \cong 15/2 \text{ (1.5\%)},$$

$$e^2 \cong 22/3 \text{ (0.75\%)},$$

$$e^2 \cong 37/5 \text{ (0.15\%)},$$

$$e^2 \cong 133/18 \text{ (0.002\%)},$$

....

	A	B	C	D
1	N	a	num.	denom.
2	7.389056099	7	7	1
3		2	15	2
4		1	22	3
5	{=FractCont(A2)}	1	37	5
6		3	133	18
7		18	2431	329
8		5	12288	1663
9		1	14719	1992
10		1	27007	3655
11		6	176761	23922
12		30	5329837	721315
13		8	42815457	5794442
14		1	48145294	6515757
15		1	90960751	12310199
16		#N/D	#N/D	#N/D

Sometime the continued fraction expansion requires the multiprecision. Expand, for example, the Ramanujan number $e^{\pi\sqrt{58}} \cong 24591257751.9999998222132414696\dots$ with 30 significant digits

	A	B	C
1	Ramanujan number		
2	24591257751.9999998222132414696	{=xFractCont(A2)}	
3			
4	a	num.	denom.
5	24591257751	24591257751	1
6	1	24591257752	1
7	5624714	138318816346540679	5624715
8	1	138318840937798431	5624716
9	6	968231861973331265	39373011
10	34	33058202148031061441	1344307090
11	1	34026434010004392706	1383680101
12	2	101111070168039846853	4111667292
13	#N/D	#N/D	#N/D

As we can see, the last fraction: $101111070168039846853 / 4111667292$ approximates $e^{\pi\sqrt{58}}$ with a relative error of about $1\text{E-}30$

Continued Fraction of Square Root

FractContSqr(n)

This function returns the continued fraction expansion of the square root of an integer number

$$\sqrt{n} = [a_0, a_1, a_2, a_3 \dots 2a_0]$$

$$\sqrt{n} = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \dots}}}$$

As known, the expansion is infinite and periodic.

The algorithm stops when it come across into a term double of the first term a_0

This function returns the (m+1) vector $[a_0, a_1, a_2, \dots, a_m]$ where $a_m = 2 \cdot a_0$

Use the ctrl+shift+enter sequence.

Example. Calculate the continued fraction of $\sqrt{77}$

	A	B	C	D	E	F	G	H	I	J
1										
2										
3	77	8	1	3	2	3	1	16	0	0
4										

Therefore the infinite, periodic continued fraction is

$$\sqrt{77} = [8, 1, 3, 2, 3, 1, 16, 1, 3, 2, 3, 1, 16, 1, 3, 2, 3, 1, 16, \dots]$$

Note that the function returns only the first terms + the period.

Check Prime

Prime(n)

Prime(n) = "P" if n is prime, or the lowest factor if n is not prime
Returns "?" if the function is not able to check it.

Example

```
prime(134560093) ="P"
```

```
prime(134560079)= 89
```

Next Prime

NextPrime(n)

This function¹ returns the prime number greater than n or "?" if the function is not able to calculate it

```
nextprime(9343560093) = 9343560103
```

Modular Addition

xaddmod(a, b, m)

Performs the modular addition¹

$$(a + b) \pmod{m}$$

where a, b integer and m positive integer

Modular Subtraction

xsubmod(a, b, m)

Performs the modular subtraction

$$(a - b) \pmod{m}$$

where a, b integer and m positive integer

Modular Multiplication

xmultmod(a, b, m)

Performs the modular multiplication

$$(a \cdot b) \pmod{m}$$

where a, b integer and m positive integer

Modular Division

xdivmod(a, b, m)

Performs the modular division

$$(a / b) \pmod{m}$$

where a, b integer and m positive integer

Remember that the modular division is always possible if, and only, the module m is prime. Otherwise the division could be impossible. In that case the function returns "?"

Modular Power

xpowmod(a, p, m)

Performs the modular integer power of a^p

That is defined as the remainder of the integer division of a^p by m

$$r = a^p - m \cdot \left\lfloor \frac{a^p}{m} \right\rfloor$$

¹ Modular function xaddmod, xsubmod, xmultmod, xdivmod appear thanks to the courtesy of John Jones

Xnumbers Tutorial

Example: compute

$$3^{24} \pmod{9005}$$

```
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.

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
```

Examples: Miscellanea of modular function

	A	B	C
1			
2	m	1000453	(prime)
3	a	100456789023075	
4	b	100600000080025600	
5			
6			
7	a+b	370374	xaddmod(B3,B4,B2)
8	a*b	272322	xmultmod(B3,B4,B2)
9	a-b	35711	xsubmod(B3,B4,B2)
10	a/b	551782	xdivmod(B3,B4,B2)
11			

Observe that m = 1000453 is prime.

Perfect Square

xlsSquare(n)

Checks if a number n is a perfect square

```
xlsSquare(1000018092081830116) = TRUE
```

Because: $1000018092081830116 = 1000009046^2$

```
xlsSquare(2000018092081830116) = FALSE
```

Check odd/even

xlsOdd (n)

Checks if a number n is odd (TRUE) or even (FALSE)

Check Integer

xlsInteger(x)

Checks if a number x is integer

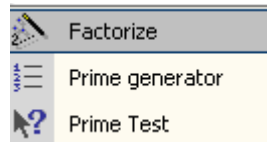
Macro - Factorize

Factorize()

This macro factorize a number of active cell returning the list of prime number and their exponents

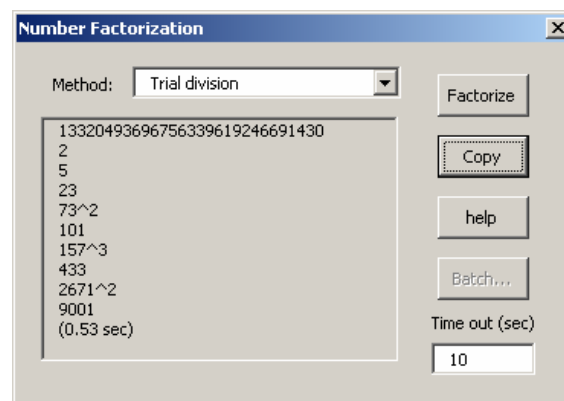
$$n = p_1^{e_1} p_2^{e_2} p_3^{e_3} \dots p_k^{e_k}$$

Select the cell contains the number you want to factorize and the run the macro Factorize from the menu or **Prime... > Factorize**



Example. Assume to have in the cell A2 the following extended number

13320493696756339619246691430



Click "copy" if you want to copy the list in the worksheet, starting from the cell just below cell A2.

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 6 digts max. For higher factor the elaboration time becomes extremely long and it comes usefull 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 factos are prime so the factorization stops itself.

For testing the primality see the probabilistic **Prime test**

Large numbers having factors more than 20 digits can be only manipulate with Xnumbers but no more factorized. This task requires sophisticated algorithms joined with extreme fast routines, usually written in C++ or Assembler.

When the numbers are very large, no efficient integer factorization algorithm is published; a recent effort which factored a 200 digit number (RSA-200) took eighteen months and used over half a century of computer time. The supposed difficulty of this problem is at the heart of certain algorithms in cryptography such as RSA.

Not all numbers of a given length are equally hard to factor. The hardest instances of these problems are those where the factors are two randomly-chosen prime numbers of about the same size, but not too close

Factoring software is available either in commercial math packages or in standalone freeware programs.

One of the most interesting program released in the public domain, supporting Quadratic Sieve (QS) and Number Field Sieve (GNFS), that we have used for many years is Msieve.

Batch Factorization with Msieve

Msieve is a very power freeware program, created by Jason Papadopoulos, for factoring large integers.

You can download the latest version of Msieve.exe from www.boo.net/~jasonp

No particular installation is required. Simply copy it in any folder that you like, for example, /msieve.

What Msieve Does

Factoring is the study (half math, half engineering, half art form, and half... genial tricks) of taking big numbers and expressing them as the product of smaller numbers. As the number to be factored becomes larger, the difficulty involved in completing its factorization explodes and the elaboration time increase sharply. The multiprecision library contained in Xnumber, written in VBA, is no more sufficient for performing the factorization of number larger of 18-20 digits.

Msieve can with high probability find the complete factorization of any input number up to about 125 digits in size. The actual number of digits supported is much higher (up to 164 digits), but problems larger than 125 digits are likely to fail.

Trial division is used on all inputs; if the result is less than 25 digits in size, tiny custom routines do the factoring. For larger numbers, the code switches to more powerful methods. Prior to version 1.04, those methods were limited to the quadratic sieve (QS). From that point on, however, an implementation of Pollard-Brent algorithm and the number field sieve NFS are also available.

A description of QS and NFS can be found in the Msieve Library itself with also a good amount of Quadratic Sieve references.

The oldest users will be pleasant to know that Msieve.exe is a consolle program. It can be started at the prompt command (the old DOS enviroment) using the following general syntax

```
>> msieve [options] [number]
```

Msieve supports many useful options. For a complete list, run the command

```
>> msieve -h
```

For factoring one number give the following command

```
>> msieve -q 8004000546054003543176004301
```

```
8004000546054003543176004301
p10: 1000400017
prp10: 2000000011
prp10: 4000400023
```

In this case the program only outputs all the factors found. The code "p10" indicates a prime factor of 10 digits; "prp10" indicates a probable prime factor of 10 digits.

Xnumbers Tutorial

With the option -v (verbose), the program will also output the factoring trace with many useful information (for factoring experts, of course).

```
>> msieve -v 8004000546054003543176004301
```

One useful option is also the time limit setting -d

```
>> msieve -v -d 10 number
```

This elaboration will stop after 10 minute, ended or not, and the intermediate results will be saved in a working file, ready for a successive restart.
Onother way to stop a running elaboration is sending the key sequence CTRL+C to the active command window.

The younger users will be happy to know that this elaboration can be performed directly from Excel by the macro Numbers/Factors...

Note. The first time that you run Msieve from this macro, you have to provide the folder where Msieve.exe is located. Follow the simple configuration instructions.

After that, the macro can start Msieve in batch input mode, returning output in Excel at the end of the elaboration.

Using is very simple. Select the cell containing the number to factorize

	G	H	I	J	K	L	M	N
16								
17		163124803832237552410967357287903138290607634257976801034173						
18								

Start the macro Numbers/Factors..., choose the Msieve and click on "Factorize".



If we like we can modify the job timeout (default 15 min).

The "Factorization Job Manager" lists and visualizes all the jobs submitted to Msieve.

ID	Status	Date	Time	Progress
F-070316225024-A38	C	16/03/2007	22.50.24	27
F-070316225324-H23	C	16/03/2007	22.53.24	24
F-070316225520-H31	C	16/03/2007	22.55.20	35
F-070316225730-H7	C	16/03/2007	22.57.30	0
F-070316225812-H16	C	16/03/2007	22.58.12	0
F-070316225908-H17	C	16/03/2007	22.59.08	31
F-070316230027-H18	C	16/03/2007	23.00.27	33

Msieve v. 1.17
factoring
163124803832237552410967357287903138290607634257976801034173
(60 digits)
prp11 factor: 51030885799
prp11 factor: 51030885799
prp17 factor: 11516787708996547
prp22 factor: 5439042183600204290159

- Lists Refresh
- Imports the results in Excel
- Kills a running job
- Deletes a job
- Verbose mode on/off
- This Help

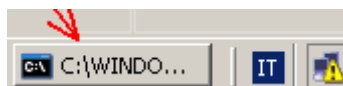
The jobs list is automatically refreshed every second. If the refreshing activity stops for some reason, you can manually refresh it.

Xnumbers Tutorial

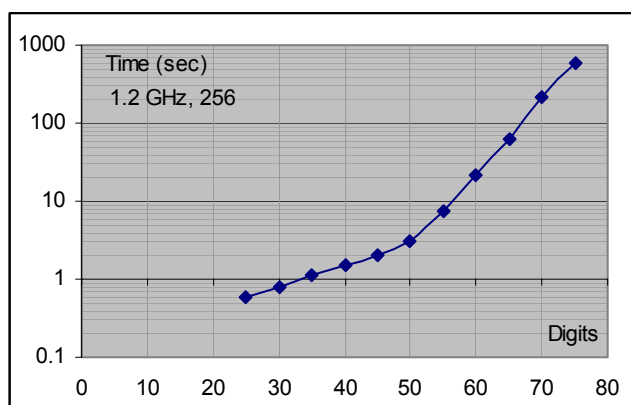
The factors obtained can be imported into Excel by the apposite button. The macro outputs each factor in the format [exponent, factor]

	A	B	C	D	E	F	G
20							
21	163124803832237552410967357287903138290607634257976801034173						
22							
23	1	11516787708996547					
24	1	5439042183600204290159					
25	2	51030885799					
26							

An active factorization job runs in a minimize window. If you like, you can open it at normal size and observe what Msieve is doing



The precedent submitted jobs remain memorized until they are delete. They can be recalled and viewed by the button **Batch...** of the Number Factorization panel
The factorization time is usually very fast for numbers up to 50 digits and remains reasonably fast for numbers in the range 50 - 75 digits, as shows the following statistical graph



Factorization function

Factor(n)

This function performs the decomposition in prime factors of a given integer number
Returns an array of two columns: the first column contains the prime factors and the second column contains the exponents

This function is useful for factorizing on-line integer numbers $N < 1E14$ directly in the worksheet.

It uses the so called "brute force attack" and the Fermat-Lehman method.

	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

$$2277785128000 = 2^6 5^3 23^2 73^2 101$$

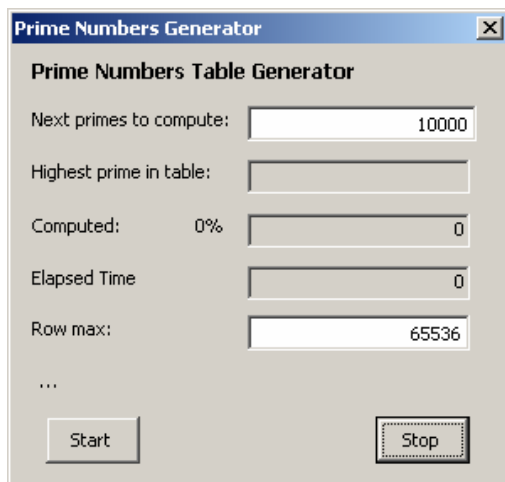
The #N/D symbol indicates the end of factors list. To make sure to get all factors you have to extend the selection until you see this symbol

Using this function is very simple. Select a range of 2 columns and several rows (for example 7 rows). The insert the function Factor using the ctrl+shift+enter sequence.

Macro - Prime Numbers Generator

PrimeGenerator

This macro is useful to generate your own table of prime numbers. The table begins from the cell A1 of the active worksheet.



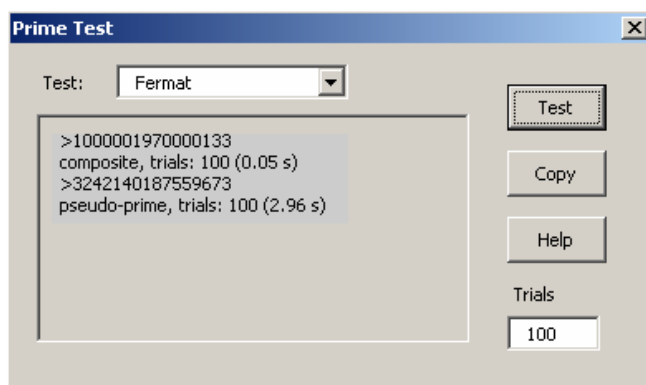
The macro computes 10.000 (default) prime numbers for each time. The macro can be stop and restart as you like

It always restarts from the last prime number written in the worksheet.

Prime Test

This macro perform the probabilistic prime test with the Fermat or Miller-Rabin method. These tests are especially adapt for long number. Using is very simple. Select the number that you want to test and start the macro from the menu **Macros / Numbers / Prime test**

Select the method that you want 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).

The numbers satisfying the Fermat or Miller-Rabin are called **"pseudo-prime"**

The probability is correlated to the trials number T with the following approximate formula

Fermat test	$p = 1 - 2^{-T}$	For T = 100 the probability is about $1 - 7.8 \cdot 10^{-31}$
Miller-Rabin test	$p = 1 - 4^{-T}$	For T = 50 the probability is about $1 - 7.8 \cdot 10^{-31}$

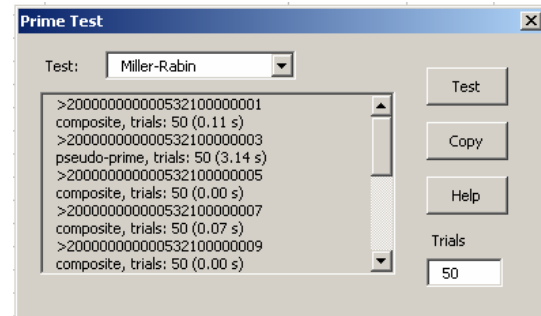
You can also select a list of cells containing several numbers to test. This is useful for finding prime in a set of consecutive integers.

Example: find the next prime number after 200000000000532100000000

A prime number must be odd, so let's begin to prepare a sequence of 20 or more odd numbers starting from 200000000000532100000001

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.

Miller-Rabin test	
200000000000532100000001	composite, trials: 50 (0.13 s)
200000000000532100000003	pseudo-prime, trials: 50 (3.16 s)
200000000000532100000005	composite, trials: 50 (0.00 s)
200000000000532100000007	composite, trials: 50 (0.06 s)
200000000000532100000009	composite, trials: 50 (0.00 s)
200000000000532100000011	composite, trials: 50 (0.00 s)
200000000000532100000013	composite, trials: 50 (0.07 s)
200000000000532100000015	composite, trials: 50 (0.01 s)
200000000000532100000017	composite, trials: 50 (0.00 s)
200000000000532100000019	composite, trials: 50 (0.10 s)
200000000000532100000021	composite, trials: 50 (0.09 s)



In this case we have found a probable prime 200000000000532100000003

You may prove by yourself 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 (2 x 2) array 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 only want a particular solution $[x_0, y_0]$ 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		a	b	c
3		2	3	6
4		{=DiophEqu(B4;C4;D4)}		
5				
6		x	y	
7		-6	6	
8		3	-2	

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 the following.

Long numbers. This function works also with extended numbers.

Example. Find a solution of the equation $ax+by = c$ having the following coefficients

a	b	c
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.

You can enjoy yourself to prove that the result returned from the function DiophEqu is correct

	A	B	C
1			
2	a	18760000596690052	
3	b	13650847757772	
4	c	64	
5		{=DiophEqu(B2,B3,B4)}	
6		x	y
7		-6662999124128	9156784235119760
8		3412711939443	-4690000149172513

Brouncker-Pell Equation

= PellEqu(d, [n])

This function solves the so called Brouncker-Pell quadratic equation on the integer domain

$$x^2 - d \cdot y^2 = 1 \quad , \quad x, y, d \in \mathbb{N}$$

The function returns the n^{th} integer solution (default $n = 1$) using the continued fraction method. As know, this equation has non trivial solutions if d is not a perfect square. The function works for low-moderate integer d values but, even with this constraints, sometimes this smallest solution is quite small, and sometimes it is huge. Let's see.

Compute the first 3 integer solutions of the equation $x^2 - 31 \cdot y^2 = 1$

	A	B	C	D	E
1	Brouncker/Pell equation				
2	$x^2 - d \cdot y^2 = 1$			{=PellEqu(A4,C4)}	
3	d		i	x	y
4	31		1	1520	273
5			2	4620799	829920
6			3	14047227440	2522956527

As we can see the solutions grow sharply also for very small d .

Sometime the first solution is huge also with moderate d . In that case is better to arrange the solution in vertical cells, as shown in the following examples

	A	B	C	D	E
9					
10	d =	989	x =	550271588560695	
11			y =	17497618534396	
12					
13		{=PellEqu(B15)}		{=PellEqu(B10)}	
14					
15	d =	991	x =	379516400906811930638014896080	
16			y =	12055735790331359447442538767	
17				{=xCalc(D15^2-B15*D16^2, 60)}	
18					
19			check =>	1	

Note that the "smallest" solution of the equation $x^2 - 991 \cdot y^2 = 1$ has 30 digits !

When the solution exceed 250 digits or the time of 200 iterations the functions returns "?".

Euler's Totient function

= Totient(n)

Returns the number of integer not exceeding and relatively prime to n

This function is intended only for low-moderate numbers.

n	$\phi(n)$
210	48
7429	6336
7433	7432
323323	207360
323333	323332

Remember that two numbers n , m are relative prime if, an only if, $\text{GCD}(n, m) = 1$

With this function is easy to set a table of the totient $\phi(n)$ values for $1 \leq n \leq 100$

$\phi(n)$	1	2	3	4	5	6	7	8	9	10
0	1	1	2	2	4	2	6	4	6	4
10	10	4	12	6	8	8	16	6	18	8
20	12	10	22	8	20	12	18	12	28	8
30	30	16	20	16	24	12	36	18	24	16
40	40	12	42	20	24	22	46	16	42	20
50	32	24	52	18	40	24	36	28	58	16
60	60	30	36	32	48	20	66	32	44	24
70	70	24	72	36	40	36	60	24	78	32
80	54	40	82	24	64	42	56	40	88	24
90	72	44	60	46	72	32	96	42	60	40

Integer relation

A set of real numbers $[x_1, x_2, \dots, x_n]$, is said to possess an integer relation if there exist a set of integers $[a_1, a_2, \dots, a_n]$, not all zero, such that

$$a_1 x_1 + a_2 x_2 \dots + a_n x_n = 0$$

The integer relation algorithm can be used to solve subset sum problems, as well as to determine if a given numerical constant is equal to a root of a univariate polynomial with integer coefficients; in this case we say the constant to be "rational" such as "12/51" or "algebraic", such as $\sqrt{2}$. On the contrary, if such polynomial does not exist, we say the constant "transcendental", such as π .

Therefore this algorithm can be used for algebraic-transcendental screening of real numbers. Because we always have an approximated value of the constant, the found integer relation will not exactly zero but very close to it only if we use a sufficient number of digits. For example, we find the following 5th degree relation

$$[a_1, a_2, a_3, a_4, a_5] \cdot [1, x, x^2, x^3, x^4]^T = a_1 + a_2 x + a_3 x^2 + a_4 x^3 + a_5 x^4 \cong 0$$

where $[a_1, a_2, a_3, a_4, a_5] = [-1, 0, -2, 0, 1]$, and $x = 1.553773974\dots$

Of course, a numerical discovery of a relation by this algorithm does not in general constitute a proof of this relation; one of the reasons being that the computer operates in finite precision. In many cases, however, the relations we first discovered numerically subsequently received rigorous mathematical proofs. Moreover, many complicated relations probably would never, have been dreamed of without the assistance of the computer.

Algorithms for finding integer relations include HJLS algorithm, LLL algorithm, PSLQ algorithm. All these algorithms requires high-precision arithmetic, but the PSLQ, developed by Ferguson and Forcade (1979) and improved by Ferguson and Bailey (1992), is proved the most efficient and stable integer relation algorithm. The Xnumbers VB macro has been developed from the PSLQ described by Bailey, D. and Plouffe at <http://www.cecm.sfu.ca/organics/papers/bailey>

Norm and Bounding.

The norm of the relation is defined the Euclidean norm of the vector $|x| = [x_1, x_2, \dots, x_n]$

Among the relations satisfying the constraint $|\sum x_i a_i| < \varepsilon$, we are clearly interested in those relations having small norm, and, if possible, with the smallest nth degree.

The constraint of small norm has been introduced for filtering between relations.

In fact, if we can easily observe that, if the vector x becomes exagerately large, then an integer relation will be surely find for any real number.

$$|314159265358979323 - 10000000000000000 \pi| < 10^{-16},$$

Of course, the norm of this relation is very high (about $3.2 \cdot 10^{17}$) and thus we have to reject it. PSLQ computes the norm of any examined relation and returns the bound of the coefficients. Even if a relation is not found, the resulting bound means that cannot possibly be the root of a polynomial of degree n , with coefficients of size less than the established bound. Even negative results of this sort are often of interest.

Examples

given a real value x we build the vector $[1, x, x^2, x^3, \dots, x^n]$ with n in increasing order $n = 3, 4, 5, \dots$, we are searching for possible integer relations having small error $\varepsilon < 1E-14$ and small norm $|x|$

Epprox. value: 3.189207115002720, Exact value: $\sqrt[4]{2} + 2$

Relation: [14, -32, 24, -8, 1], Polynomial: $14 - 32x + 24x^2 - 8x^3 + x^4$

Epprox. value: 2.67413461226797, Exact value: $\sqrt[3]{2} + \sqrt{2}$

Relation: [4, 24, -12, 4, 6, 0, -1], Polynomial: $4 + 24x - 12x^2 - 4x^3 + 6x^4 - x^6$

Exprox. value: 2.991971857463750, Exact value: $\sqrt[3]{2} + \sqrt{3}$

Relation: [-23, -36, 27, -4, -9, 0, 1], Polynomial: $-23 - 36x + 27x^2 - 4x^3 - 9x^4 + x^6$

Exprox. value: 0.809016994374947, Exact value: $(\sqrt{5} + 1)/2$

Relation: [1, 2, -4], Polynomial: $1 + 2x - 4x^2$

Exprox. value: 0.923879532511287, Exact value: $\sqrt{(\sqrt{2} + 2)/4}$

Relation: [-1, 0, 8, 0, -8], Polynomial: $-1 + 8x^2 - 8x^4$

Exprox. value: 0.900968867902419, Exact value: $\cos(\pi/7)$

Relation: [1, -4, -4, 8], Polynomial: $1 - 4x - 4x^2 + 8x^3$

Exprox. value: 1.839286755214160, Exact value: $\sqrt[3]{\frac{19}{27} - \frac{\sqrt{33}}{9}} + \sqrt[3]{\frac{19}{27} + \frac{\sqrt{33}}{9}} + \frac{1}{3}$

Relation: [1, 1, 1, -1], Polynomial: $1 + x + x^2 - x^3$

Exprox. value: 0.012345678901235, Exact value: $1/81$

Relation: [1, -162, 6561], Polynomial: $1 - 162x + 6561x^2 = (1 - 81x)^2$

A very simple trick for testing a relation consists in increasing the degree to $n+1$; if the algorithm still returns the same relation then, with high probabability, the relation is correct. But pay attention, because, sometime a numeric value can have two different relations with different degree. This means that the given value is the root of two different polynomials. Let's see the following example

Exprox. value: 7.7099759466767, Exact value: $\sqrt[3]{5} + 6$

Relation: [221, -108, 18, -1], Polynomial: $221 - 108x + 18x^2 - x^3$

But the same value also returns the following relation of degree 5

Relation: [221, 113, -90, 17, -1], Polynomial: $221 + 113x - 90x^2 + 17x^3 - x^4$

We can verify that both relations are correct, in this case.

Higher degree relations

All the previous examples can be found using PSLQ in standard 64 bit precision. However, when we search for higher the Polynomial degree ($n > 6$), the multiprecision is required

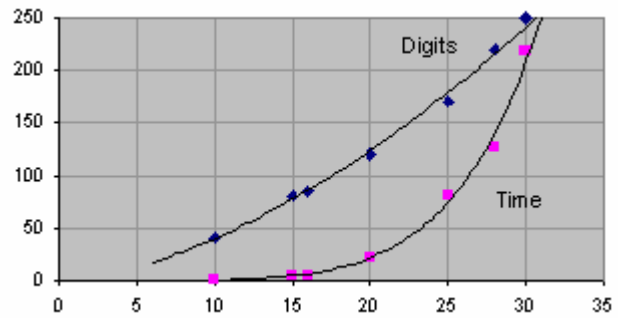
Exprox. value: 3.00161461434129452355339436348, Exact value: $\sqrt[3]{4} + \sqrt{2}$

Relation: [-8, 48, -20, 56, -6, 8, 5, 0, -1], Polynomial: $-1 + 104x^2 - 18x^4 + 8x^6 - x^8$

Time: 0.4 sec

We have seen that for 8th degree polynomial we have needed about 30 digits, in this case. Generally speaking, the required precision increases with the degree and the norm of the coefficients of the polynomial, as reported in the following table

Degree	Digits	Time (sec)
10	40	0.5
15	80	4.5
16	85	5.4
20	120	23.0
25	170	81.7
28	220	126.1
30	250	217.7



Absolute limits. The table can be accessed in two ways: given a polynomial degree we get the minimum precision of the number (number of significant digits); on the contrary, given a precision of a number (digits) we get the maximum obtainable polynomial degree.

For example, if we are searching for a polynomial of 15 degree, we must have a number with 80 digits, at least. If we have calculate a number with 100 digits we can search for polynomials having degree $n < 18$.

In the above table and graph we have also reported the time¹ in second of the PSLQ (two-level) algorithm. As we can see, the time increases sharply with the degree. The time stays less the few seconds for $n < 16$, but requires about 3-4 minutes for $n = 30$ and 250 digits (the maximum possible for Xnumbers)

Some other high results are shown in the following list. In these trials, the problem is to recover the polynomial of degree n satisfied by α , where $\alpha = a^{1/r} \pm b^{1/s}$. In other words, the input vector $x = (1, \alpha, \alpha^2, \alpha^2 \dots \alpha^{n-1})$, where $n = (r s + 1)$. Here the working precision was set to the level required by PSLQ and the input number α was computed with the needed precision, as determined by the above table.

Degree: 12. Exact value: $\alpha = 3^{1/6} + 2^{1/2} \cong 2.61515051\dots$ (50 digits)

Relation: [-25, 0, 552, 0, -60, 0, 166, 0, -60, 0, 12, 0, -1],

Time: 2.8 sec

Degree: 12. Exact value: $\alpha = 3^{1/4} + 2^{1/3} \cong 2.57599506\dots$ (50 digits)

Relation: [-11, -216, -360, -32, 27, -288, 24, 0, -9, -8, 0, 0, 1],

Time: 3 sec

Degree: 15. Exact value: $\alpha = 3^{1/5} + 2^{1/3} \cong 2.50565198951\dots$ (80 digits)

Relation:, [-59, -360, -540, 80, -1620, 27, -80, -540, 0, 40, -9, 0, -10, 0, 0, 1]

Time: 5 sec

Degree: 16. Exact value: $\alpha = 3^{1/4} - 2^{1/4} \cong 0.126866897949771\dots$ (85 digits)

Relation: [1, 0, 0, 0, -3860, 0, 0, 0, -666, 0, 0, 0, -20, 0, 0, 0, 1],

Time: 5.6 sec

Degree: 20. Exact value: $\alpha = 3^{1/5} - 2^{1/4} \cong 0.0565238246127\dots$ (120 digits)

Relation: [49, -1080, 3960, -3360, 80, -108, -6120, -7440, -80, 0, 54, -1560, 40, 0, 0, -12, -10, 0, 0, 0, 1],

Time: 16.8 sec

Degree: 20. Exact value: $\alpha = 3^{1/5} + 2^{1/4} \cong 2.434938054618\dots$ (120 digits)

Relation: [49, -1080, 3960, -3360, 80, -108, -6120, -7440, -80, 0, 54, -1560, 40, 0, 0, -12, -10, 0, 0, 0, 1],

Time: 23 sec

¹ Computation on a PC with AMD Athlon, 2GHz, 512 MB Ram

It is interesting to note that the two numbers $3^{1/5} \pm 2^{1/4}$ gives the same relations. In fact they are both roots of the same 20th degree polynomial.

Degree: 25. Exact value: $\alpha = 3^{1/5} + 2^{1/5} \cong 2.39442929461255233...$ (170 digits)

Relation: [-3125, 0, 0, 0, 0, 21875, 0, 0, 0, 0, -57500, 0, 0, 0, 0, -3500, 0, 0, 0, 0, -25, 0, 0, 0, 0, 1],

Time: 81.7 sec

Degree: 25. Exact value: $\alpha = 3^{1/5} - 2^{1/5} \cong 0.0970325846184823...$ (170 digits)

Relation: [1, 0, 0, 0, 0, -116255, 0, 0, 0, 0, -11240, 0, 0, 0, 0, -3760, 0, 0, 0, 0, 5, 0, 0, 0, 0, -1],

Time: 68.7 sec

Degree: 28. Exact value: $\alpha = 3^{1/7} - 2^{1/4} \cong -0.01927630224403418...$ (220 digits)

Relation: [-47, -2688, -13104, -7560, 448, -79296, 136584, -108, -672, -184128, -25956, 0, 560, -74592, 54, 0, -280, -4872, 0, 0, 84, -12, 0, 0, -14, 0, 0, 0, 1],

Time: 126 sec

Degree: 30. Exact value: $\alpha = 3^{1/6} - 2^{1/5} \cong 0.0522386001789677198...$ (250 digits)

Relation: [179, -4860, 30780, -65520, 30240, -192, -405, -98820, -538380, -149520, -240, 0, 270, -84240, 90360, -160, 0, 0, -90, -7380, -60, 0, 0, 0, 15, -12, 0, 0, 0, 0, -1],

Time: 218 sec

Macro Integer Relation Finder

The macro Integer Relation Finder, from the menu "Numbers", searches for two type of integer relations:

Polynomial relation

Input: a real number x , and the n -th degree

$$a_0 + a_1x + a_2x^2 + a_3x^3 + \dots a_nx^n$$

General relation

Input: a vector \mathbf{x} of n -th dimension

$$a_1x_1 + a_2x_2 + a_3x_3 + \dots a_nx_n$$

Using this macro is very simple: select the cell containing the number x or the range of the column-vector \mathbf{x} and start the macro. The macro assumes a simple layout in order to speed-up the input operation. The input layout is different for the two type of relation

Polynomial relation input layout

	A	B	C	D	E	F
1						
2	x =>	0.817120592832139658891211756327				
3	n =>	5				
4						

The cell B2 contains the multiprecision number x of 30 digits and just below, the cell B3 contains the degree n . Select the cells B2 and start the macro

Multiple searching for one x and several degree

	A	B	C	D	E	
1						
2	x =>	0.817120592832139658891211756327				
3	n =>	5	6	7		
4						

The cell B2 contains the multiprecision number x of 30 digits and just the row below, the range B3:D3 contains the degree $n = 5, 6, 7$. Select the cells B2 and start the macro. For the same number x , the macro will search for integer polynomial with degree 5, 6 and 7 respectively

Multiple searching for several x and several degree

	A	B	C	D	E	
1						
2	x1 =>	2.99197185746375045829				
3	x2 =>	0.12686689794977139410171893				
4	x3 =>	0.817120592832139658891211756327				
5	n =>	5	6	7	8	
6						

The range B2:B4 contains the multiprecision numbers x_1, x_2 and x_3 and just the row below, the range B5:E5, contains the degree $n = 5, 6, 7$ and 8. Select the range B2:B4 and start the macro. For the each number x , the macro will search for 4 integer polynomial with degree 5, 6, 7 and 8, respectively

General relation input layout

	A	B	
1	k	\mathbf{x}	
2	1	35.81255117	
3	2	3.141592654	
4	3	2.718281828	
5	4	0.318309886	
6	5	0.367879441	
7			

The range B2:B6 contains the vector \mathbf{x} . Select the range B2:B6 and start the macro. The dimension of the relation will be 5, just the same of the vector dimension

Multiple searching for several vectors is possible. Simple select as many adjacent vectors as we need. Note that each vector can have different dimension. In this case, select the rectangular range containing all the vectors.

	A	B	C	D
1	k	x1	x2	x3
2	1	35.81255117	8.554440156	27.25811101
3	2	3.141592654	3.141592654	3.141592654
4	3	2.718281828	2.718281828	2.718281828
5	4	0.318309886	0.318309886	0.318309886
6	5	0.367879441	0.367879441	0.367879441
7				

	A	B	C	D
1	k	x1	x2	x3
2	1	35.812551	8.554440156	27.25811101
3	2	3.1415927	3.141592654	3.141592654
4	3	2.7182818	2.718281828	2.718281828
5	4	0.3183099	0.318309886	0.318309886
6	5		0.367879441	0.367879441
7	6			0.367879441
8				

Output results

The macro writes the result in the worksheet beside the input data using a simple standard layout almost similar for polynomial and general relation.

Let's see with an example

Example 1. We want to search for a smallest integer polynomial having the approximate root:
 $x \approx 2.99197185746375045829465694878...$ (30 digits)

We restrict our searching to the polynomials from 3 to 8 degree

	A	B	C	D	E	F
1	x =	2.99197185746375045829465694878				
2	n =	3	4	5	6	7
3						
4	12.19.30 Integer Relation searching for:					
5	Value	2.99197185746375045829465694878				
6						
7	Digits	30	30	30	30	30
8	Degree	3	4	5	6	7
9	RC	1	1	1	0	0
10	Residue	-4.1E-30	0	0	4.16E-30	4.63E-30
11	Bound	8.32E+24	1.92E+25	52948.1	0.30	22.9
12	Iter	286	638	201	103	116
13	Time	0.19	0.30	0.15	0.13	0.21
14						
15	k	coef	coef	coef	coef	coef
16	0	90109	-2.7E+25	3600	23	23
17	1	1.19E+25	-4.3E+24	46156	36	13
18	2	1.5E+24	7.86E+24	-47672	-27	-63
19	3	-1.8E+24	3E+24	-53088	4	31
20	4		-1.4E+24	-4179	9	5
21	5			8516	0	-9
22	6				-1	-1
23	7					1
24						

Insert the input data x and the degree 3, 4, 5, 6 and 7 as in the sheet at the left. Select the cell B1 and start the macro

The input fields are correctly filled. Press "start". In few seconds the macro finds the polynomials adding same useful information. Fortunately it have found 2 "good" polynomials, in this case

The coefficients of the polynomials are listed in the last section of the report, but just above, the macro write the information returned by the PSLQ routine

Digits: the precision of the calculus

Degree: degree of the polynomial; degree +1 is the dimension of the relation

RC: return code: 0 =OK, 1 =precision exhausted, 2 = iteration overflow, 3= overflow error
 This code states the "goodness" of the relation found

Residue: relative residue of the relation, computed as ε / a , where

$$\varepsilon = \left| \sum a_i x_i \right|, \quad a = \max(|a_i x_i|)$$

Bound: this number M states that there can exist no relation vector whose Euclidean norm is less than M. This is a very important result provided by PSLQ algorithm. Together with RC helps us to analyze the goodness of the relation found.

Iter: number of iterations needed for reaching the result

Time: elaboration time in seconds

Example 2. We have 3 real numbers x_1, x_2, x_3 computed with 15 precision digits.

$$x_1 = 35.812551166675, \quad x_2 = 8.554440155849, \quad x_3 = 4.186428394404$$

We want to investigate if they could be express by the following symbolic formula

$$x = \frac{1}{a_1} \left(a_2 \pi + a_3 e + \frac{a_4}{\pi} + \frac{a_5}{e} \right) \Leftrightarrow -a_1 x + a_2 \pi + a_3 e + \frac{a_4}{\pi} + \frac{a_5}{e} = 0$$

where $[a_1, a_2, a_3, a_4, a_5]$ all integer.

For finding this integer relation of dimension $n = 5$ (that it is compatible with the precision of the numbers that we have) we compute the following vectors $[x, \pi, \pi^{-1}, e, e^{-1}]$ changing the x value with x_1, x_2, x_3 respectively

	A	B	C	D
1		Vector 1	Vector 2	Vector 3
2		35.812551166675	8.554440155849	4.186428394404
3		3.141592653590	3.141592653590	3.141592653590
4		2.718281828459	2.718281828459	2.718281828459
5		0.318309886184	0.318309886184	0.318309886184
6		0.367879441171	0.367879441171	0.367879441171
7				
8	Digits	15	15	15
9	Dim	5	5	5
10	RC	0	0	0
11	Residue	2.79232E-16	6.43789E-15	-3.54992E-15
12	Bound	16.42520273	13.65500087	345620339.9
13	Iter	17	22	266
14	Time	0.05859375	0.0703125	0.1875
15				
16	k	coef	coef	coef
17	1	1	1	19791281914
18	2	-10	-3	3609428878
19	3	-6	-4	-40623080643
20	4	12	8	49810444343
21	5	22	25	1021218375
22				

Arrange the 3 vectors as in the sheet at the left. Select the range B2:D6 and start the macro.

The input fields are correctly filled and the option button "General" is selected. Press "start".

In very few seconds the macro successfully finds integer relations for the first two numbers, while, on the contrary, the third numbers has not a small simple relation. Note how large are the coefficients of the third column. Note also that its bound $M = 3.4E+9$ its very high; it assures that no integer relation exists with norm $\|a\| < 3.4E+9$

Linear Algebra Functions

Matrix Addition

xMatAdd(mat1, mat2, [DgtMax])

Performs the addition of two matrices. 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. 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. 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 a (n x n) square matrix

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.

It returns "?" for singular matrix.

Matrix Modulus

xMatAbs(A, [DgtMax])

Returns the absolute value of a matrix or vector.

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

$$c = V_1 \bullet V_2 = \sum_{i=1}^n V_{1i} \cdot V_{2i}$$

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 Transform

= xMatBAB(A, B, [DgtMax])

Returns the matrix product:

$$C = B^{-1} A B$$

This operation is also called the "*Similarity Transform*" 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 Transform is also called the "*orthogonal transform*". A and B must be square matrices.

Matrix Power

= xMatPow(A, n, [DgtMax])

Returns the integer power of a square matrix.

$$B = A^n = \overbrace{A \cdot A \cdot A \dots A}^{n \text{ time}}$$

Example

E2		fx {=xMatPow(A2:C4,15)}					
	A	B	C	D	E	F	G
1		A				A ¹⁵	
2	6	9	0		16653614364314747424	28397784411897880608	-19149126329521274688
3	9	10	-9		28397784411897880608	48423978210235080160	-32653145818458163872
4	0	-9	8		-19149126329521274688	-32653145818458163872	22018597254389769056
5							

Matrix LU decomposition

= xMatLU(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 refers to a permutation of **A**

If the square matrix has dimensions (n x n), this function returns an (n x 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: **A = P L U**, where **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**

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1															
2															
3															
4															
5															
6															
7															
8															

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

= xMatLL(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 matrix

Example.

	A	B	C	D	E	F	G	H	I	J	K
1								matrix L			
2		1	0	2	1			1	0	0	0
3		0	1	-1	-2			0	1	0	0
4		2	-1	21	8			2	-1	4	0
5		1	-2	8	7			1	-2	1	1
6											
7											

{=xMatLL(B2:E5)}

The diagonal elements of the **L** matrix are all positive. So the matrix **A** is positive definite 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									

<Attention !>

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 V_1 and V_2 are parallels, the vector product is the null vector.

Solve Linear Equation System

xSYSLIN(A, B, [DgtMax])

Solves a linear system 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. It 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 have an average accuracy of about 1E-8, while in multiprecision we have an accuracy better than 1E-28

	A	B	C	D	E	F	G	H	I
1	A								b
2	462	792	1287	2002	3003	4368	12376		24290
3	924	1716	3003	5005	8008	12376	31824		62856
4	1716	3432	6435	11440	19448	31824	75582		149877
5	3003	6435	12870	24310	43758	75582	167960		333918
6	5005	11440	24310	48620	92378	167960	352716		702429
7	8008	19448	43758	92378	184756	352716	705432		1406496
8	12376	31824	75582	167960	352716	705432	1352078		2697968
9									
10	Solution x								Error
11	1.00000000000000000000000000000004								4E-29
12	0.99999999999999999999999999999907								9.3E-29
13	1.00000000000000000000000000000009								9E-29
14	0.99999999999999999999999999999953								4.7E-29
15	1.00000000000000000000000000000001								1E-29
16	1								0
17	0.99999999999999999999999999999998								2E-30
18									

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 ($n \geq 3$), in vertical consecutive cells. ($n = 3$ 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_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	0.857553216	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, C2, D3 contain the formulas

[B3]=C2

[C2]=COS(B2)

[D3]=ABS(B2-C2)

Selecting the range B3:D3 and dragging down we can easily perform the iterative process as we like.

We observe that the convergence is evident but quite slow.

	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 12th value has an error of about $3.5E-3$. But if we perform 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 Δ^2 extrapolation every 3 iterations.

We have inserted in the cell B5

=ExtDelta2(B2:B4)

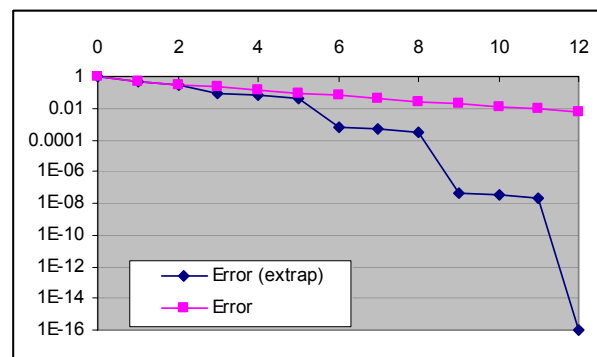
In the cell B8

=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 the acceleration effect



The Aitken's extrapolation formula works very well with the Gauss-Seidel iterative method, and for accelerating the convergence of many iterative processes.

Macro for Multiprecision Matrix Operations

This application collects a set of useful macros performing multiprecision matrix operations

Determinant	$\det(A)$	Gauss-Jordan algorithm
Addition	$A + B$	
Subtraction	$A - B$	
Multiplication	$A \cdot B$	
Scalar multiplication	$k \cdot A$	
Inverse	A^{-1}	Gauss-Jordan algorithm
Similarity transform	$B^{-1} A B$	
Linear System	$AX = B$	Gauss-Jordan algorithm
Linear System overdetermined.	$Ax = b$	rows > columns
LU decomposition	$A = LU$	Crout's algorithm
Cholesky decomposition	$A = LL^T$	Cholesky's algorithm
Norm	$\ A\ $	
Scalar product	$A^T \cdot B$	
SVD	$U \cdot \Sigma \cdot V^T$	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 (addition, multiplication, etc.) require two matrices. In that case you have also to select the second matrix in the second input-box.

The internal calculus is performed in multiprecision. The result is converted in standard precision (15 significant digits max) for more readability, but, if you like, you may also leave it in full multiprecision format.

Example: Solve the following linear system $Ax = b$.

	A	B	C	D	E	F	G	H	I	J
1		A					b		x	
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

Choose the operation "Linear System" and then move into the right field to select the vector b.

Matrix / vector A (4 × 4)

Matrix / Vector B (4 × 1)

Output

starting from cell:

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 output starting from the output cell I2.

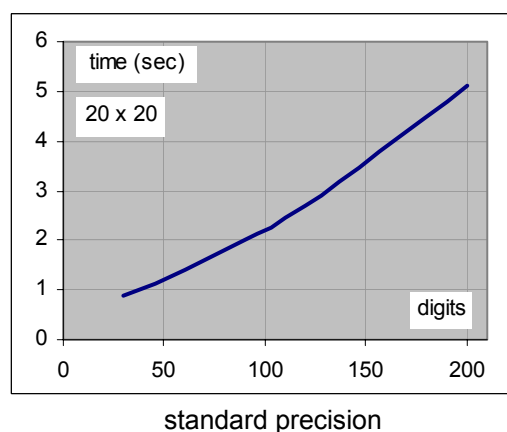
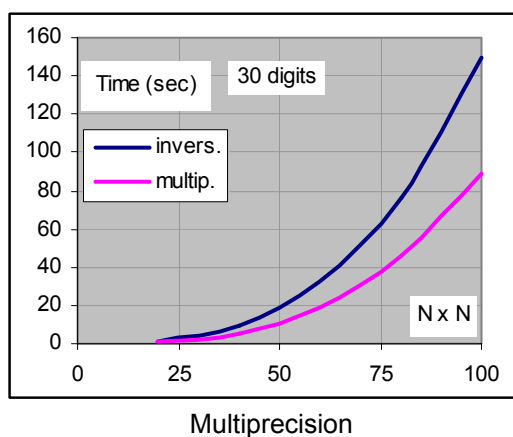
Smart Selector



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 does slow 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 for the multiplication of two dense matrices.



As we can see, the inversion of a (100 x 100) dense 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, [central])

=FFT(samples, [central])

Returns the complex matrix of the DFT transform of N samples.

This function returns an (N x 2) array. The first column contains the real part; the second column the complex part .

The optional parameter "central" (default False) outputs a central transform.

If N is an integer power of 2, that is $N = 2^P$, 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:

$$F(k) = F_r(k) + i \cdot F_i(k) = \sum_{n=0}^{N-1} f(n) \cdot [\cos(2\pi nk / N) - i \cdot \sin(2\pi nk / N)]$$

The components (F_r, F_i) are called the harmonic spectrum of $f(t)$

By the Fourier series, we can approximate a periodic function $f(t)$:

$$f(t) \cong a_0 + \sum_{k=1}^{K-1} a_k \cos(k\omega \cdot t) + b_k \sin(k\omega \cdot t)$$

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

$$\Delta t = \frac{T}{N} = \frac{1}{16} \Rightarrow t_i = i \cdot \Delta t \quad , \quad i = 0, 1, \dots, N-1$$

$$f_i = 3 + \cos(\omega t_i) + 0.5 \cos(3\omega t_i)$$

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	ΔT	ω	
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					

`{=FFT(C5:C20)}`

Note that the FFT returns a (16 x2) 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

$$A_i = \sqrt{(FFT_{re})^2 + (FFT_{im})^2}$$

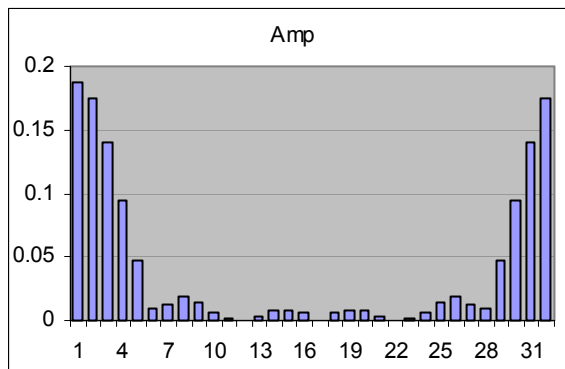
$$\theta_i = \arctan\left(\frac{FFT_{im}}{FFT_{re}}\right)$$

Observe that the first row of the FFT contains the mean of f(t).

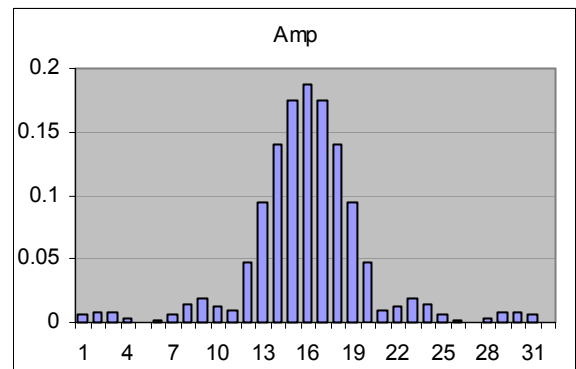
Note also that the rows from 10 to 16 are the mirror-copy of the previous rows.

Rearranging the mirror part we can obtain the central Fourier's spectrum.
For example, below there are two representations of the same DFT, obtained setting the "central" parameter "false" and "true", respectively.

Bilateral DFT



Central DFT



Discrete Fourier Inverse Transform

=DFT_INV(samples, [central])

=FFT_INV(samples, [central])

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).

The optional parameter "central" (default False) outputs a central transform.

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	ΔT	ω			
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 itself

Discrete Fourier Spectrum

=DFSP(samples, [dB], [Angle])

This function returns the harmonic spectrum of a samples set

The parameter "samples" is a vector of N equidistant samples

The optional parameter "dB" (default FALSE) sets the amplitude conversion in decibel

The optional parameter "Angle" (default "RAD") sets the angle unit (RAD, GRAD, DEG)

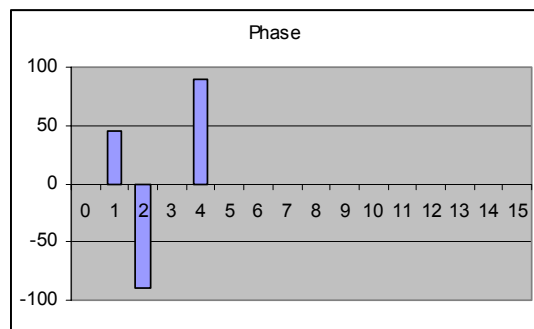
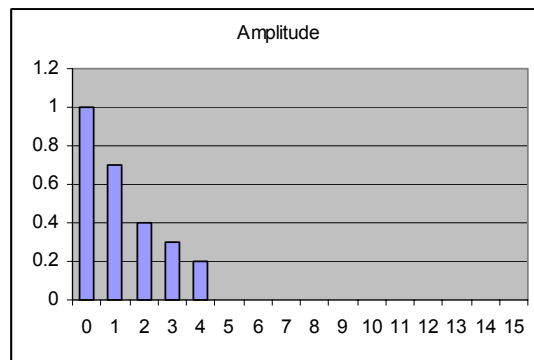
The function returns an (N x 2) array, containing the amplitude and phase.

The spectrum (A_n, θ_n) is computed for real positive frequencies, being

$$f(t) \cong f(0) + \sum A_n \sin(n\omega t + \theta_n)$$

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 builds 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 set (x_i, y_i).

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 work with matrices up to (64 x 64).

Example: Analyze the harmonic component of the following (8 x 8) table

	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 of the FFT2D. It accepts in 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:

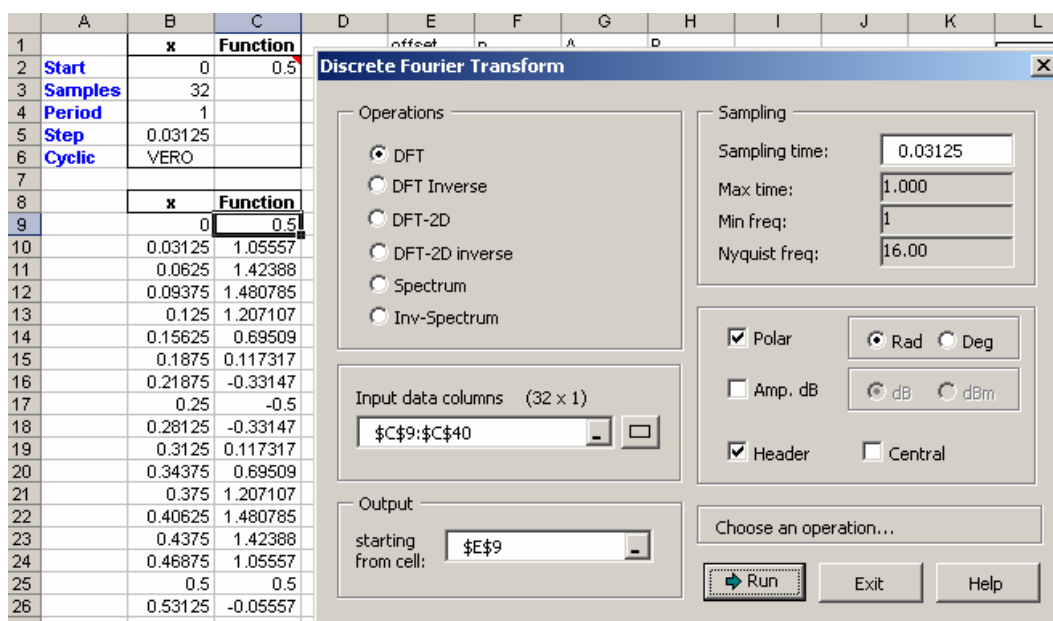
- the DFT of a data set of N samples
- the DFT-Inverse of a data set of N complex samples
- the 2D-DFT of a matrix of N x M samples
- the 2D-DFT-Inverse of a two matrices of N x M samples

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.

Its use is quite simple. Select the vector of samples and then start the macro



The column "x" is not strictly necessary. If present, the macro uses it to calculate the sampling parameters (see the top-right box).

For long input vector, you can select only the first cell C9 and start the macro. The entire input data set (32 x 1) will be automatically selected.

The macro writes the results in the following way

D	E	F	G	H
f	Hre	Him	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)

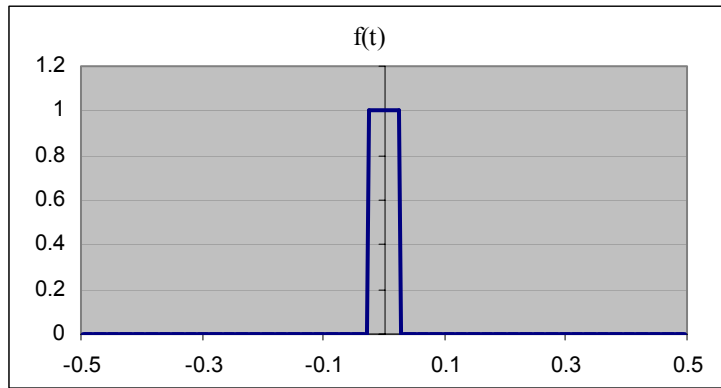
Phase = Phase (if "polar" is checked)

The amplitude can be converted in dB. That is: $\text{Amp}_{\text{dB}} = 20 \log(\text{Amp})$, or also in dBm, the power ratio relative to 1 mW (a sine wave of 1 V_{peak}, with 50 ohms load, is equivalent 10 dBm).

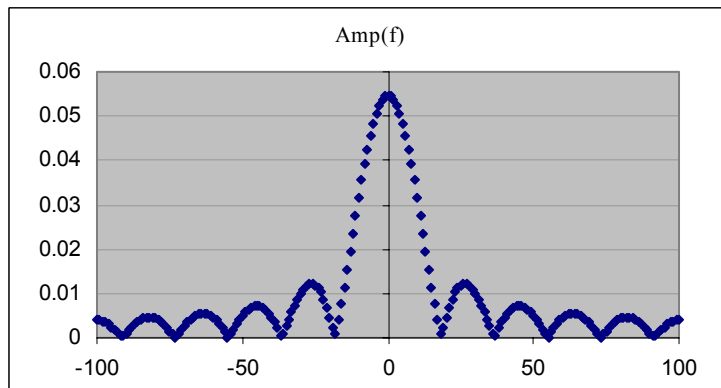
The option check box "central" specifies if the DFT output is bi-lateral or central. See the following examples

Xnumbers Tutorial

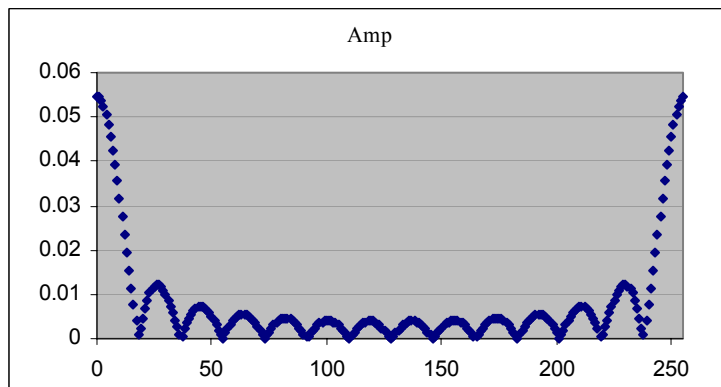
Example of a Fourier
transform of pulse.
Period = 1 sec
Duty cycle = 5.5%
Amplitude = 1
Sampling time = 1/256



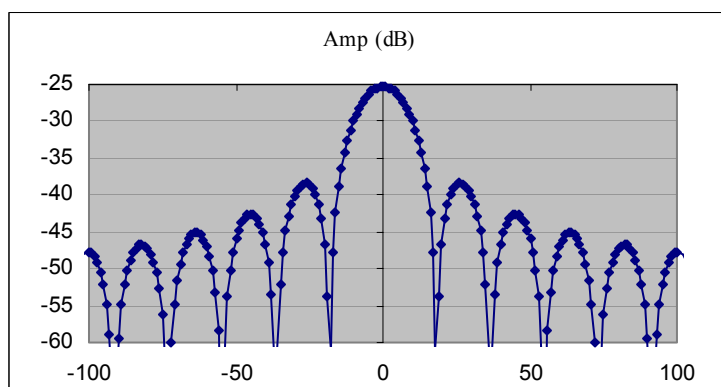
Amplitude of the central
DFT
($-100 < f < 100$)



Amplitude of the bi-lateral
DFT
($0 < f < 255$)



Amplitude in dB of the
central DFT
($-100 < f < 100$)



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 check the “polar” option box and choose consequently the appropriate units: dB and angle

Operation 2D-DFT

Dimension N and M must be integer power of 2

In this case you have to select a matrix of N x M values (do not select the axes-scales) and start the macro as usually.

If you want the DFT in polar form (Amplitude, Phase), you have to check 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
18									

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 and start the macro as usually.

If the DFT is in polar form (Amplitude, Phase), you have to check 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 multivariate functions such as:

$$f(x), f(x_1, x_2) \dots f(x_1, \dots x_m)$$

The samples can be arranged in a list or in a table (only for 2 variables).

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		x	f(x)			x1	x2	Function		
2	Start	0	0		Start	0	1	1		
3	Samples	10			Samples	10	5			
4	Period	1.8			Period	1.8	1.6			
5	Step	0.2			Step	0.2	0.4			
6	Cyclic				Cyclic					
7										
8		x	f(x)			x2	→ 1	1.4	1.8	2.2
9		0	0		x1	0	1	1.96	3.24	4.84
10		0.2	0.04			0.2	1.2	2.16	3.44	5.04
11		0.4	0.16			0.4	1.4	2.36	3.64	5.24
12		0.6	0.36			0.6	1.6	2.56	3.84	5.44
13		0.8	0.64			0.8	1.8	2.76	4.04	5.64
14		1	1			1	2	2.96	4.24	5.84
15		1.2	1.44			1.2	2.2	3.16	4.44	6.04
16		1.4	1.96			1.4	2.4	3.36	4.64	6.24
17		1.6	2.56			1.6	2.6	3.56	4.84	6.44
18		1.8	3.24			1.8	2.8	3.76	5.04	6.64
19										

The tables at the top are the skeletons to generate the samples-list or the samples-table just below. The skeleton contains the following parameters for the sampler.

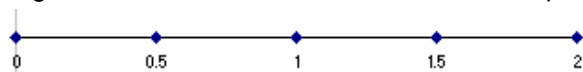
Start	starting point of the variable X_0
Samples	number of samples to generate: N
Period	length of the sampling: P
Step	length between two consecutive point $H = X_1 - X_0$
Cyclic	True or False (default), specifies if the function is periodic with period P.

The difference between a cyclic or no-cyclic function is in the step formula

$$S = P / N \quad \text{for cyclic function}$$

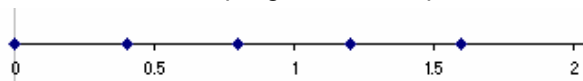
$$S = P / (N-1) \quad \text{for no-cyclic function}$$

For example, the sampling with $N = 5$, $X_0 = 0$ and $P = 2$, needs a step $H = 0.5$



The first point and the last point, 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 is $f(0) = f(2)$. Usually periodic functions require to set Cyclic = "True" for the FT analysis

The skeleton can be drawn by hand or automatically. In this case you have only to give the number of variables that you want.

The check-box "Function seed" tells the macro to created also the cell in which you can insert the function to sample

Xnumbers Tutorial

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

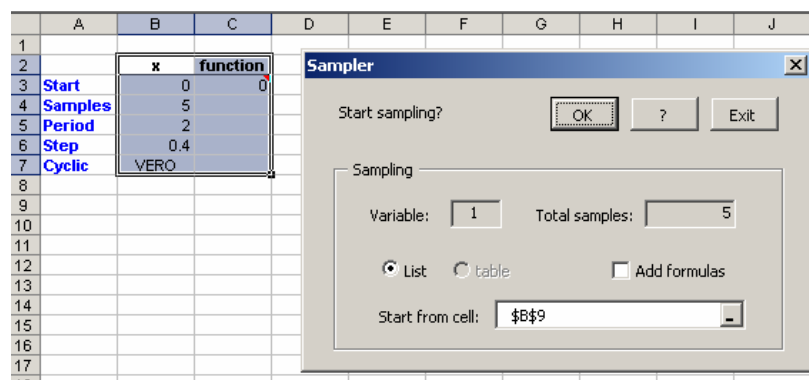
Given parameters	Calculated parameter
Samples, Period (N, P)	Step (H)
Samples, Step (N, H)	Period (P)
Period, Step (P, H)	Samples (N)

Look at the following three examples below for better explanation. The given parameters are in blue while the calculated 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 it contains only the values. Formula can be added only for a monovariate list or for a table.

Integral function

=IntegrData(xi, yi, [IntType])

Computes numerically the integral functions $F(x_i)$ of a given dataset (xi, yi)

$$F(x_i) = \int_0^{x_i} y_i dx$$

The sampling interval must be constant

The IntType parameter (default = 2) sets the integration formulas 1, 2, 3

Case 1 - 2 points integration formula of 1st degree

symmetric	$I_{12} = h(f_1 + f_2)/2 \quad E \approx h^3 f^{(2)}/12$
-----------	--

Case 2 - 4 points integration formulas of 3rd degree

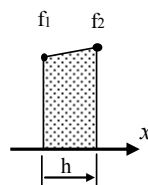
symmetric	$I_{23} = h(-f_1 + 13f_2 + 13f_3 - f_4)/24 \quad E \approx 11/720 \cdot h^5 f^{(4)}$
Left side	$I_{12} = h(9f_1 + 19f_2 - 5f_3 + f_4)/24$
Right side	$I_{34} = h(f_1 - 5f_2 + 19f_3 + 9f_4)/24$

Case 3 - 6 points integration formulas of 5th degree

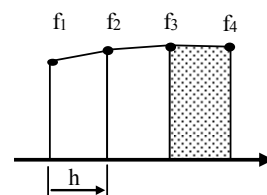
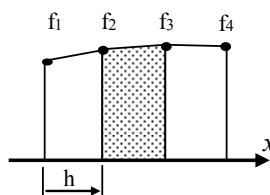
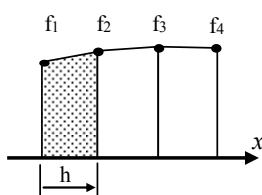
symmetric	$I_{34} = h(11f_1 - 93f_2 + 802(f_3 + f_4) - 93f_5 + 11f_6)/1440 \quad E \approx 191/60480 \cdot h^7 f^{(6)}$
Left side	$I_{12} = h(475f_1 + 1427f_2 - 798f_3 + 482f_4 - 173f_5 + 27f_6)/1440$ $I_{23} = h(-27f_1 + 637f_2 + 1022f_3 - 258f_4 + 77f_5 - 11f_6)/1440$
Right side	$I_{45} = h(-11f_1 + 77f_2 - 258f_3 + 1022f_4 + 637f_5 - 27f_6)/1440$ $I_{56} = h(27f_1 - 173f_2 + 482f_3 - 798f_4 + 1427f_5 + 475f_6)/1440$

Depending on the case, the algorithm uses these formulas at the beginning, at the center and the end of the integration interval.

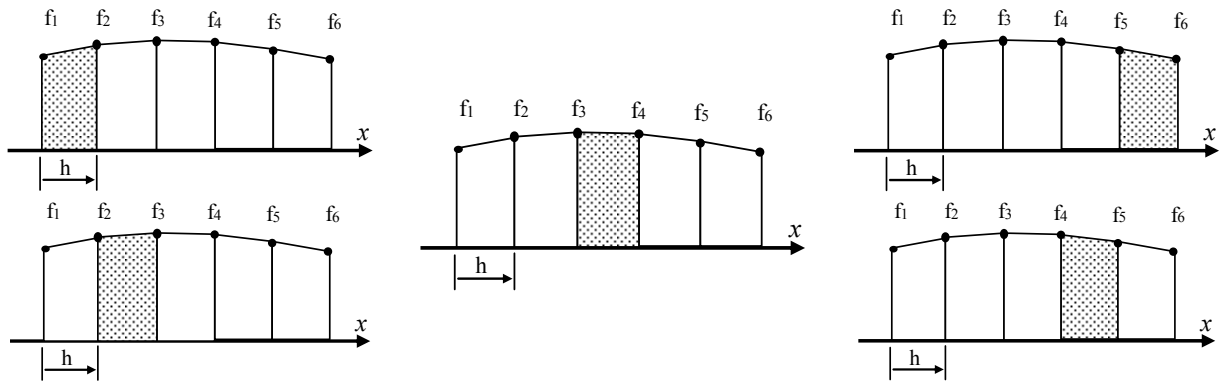
2 points formula



4 points formulas

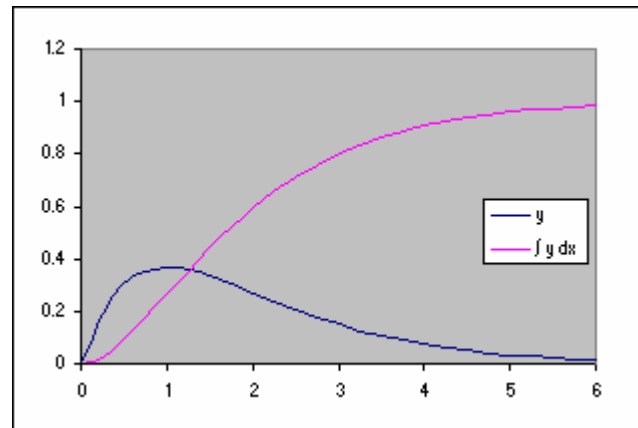


6 points formulas

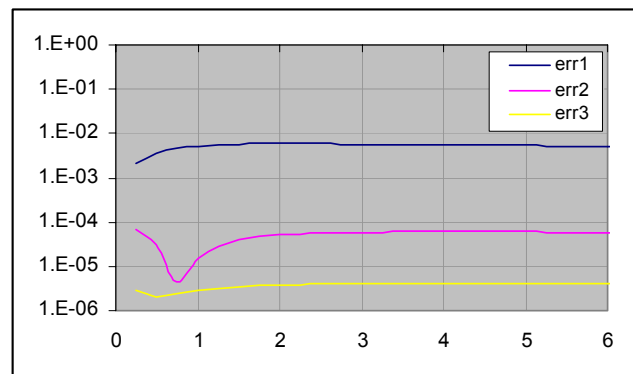


Example. Compute the integral function of the following tabulated function $y(x_i)$

	A	B	C
1	h	0.25	
2			
3			
4	x	y	$\int y dx$
5	0	0	0
6	0.25	0.1947	0.02643
7	0.5	0.30327	0.090172
8	0.75	0.35427	0.173354
9	1	0.36788	0.264256
10	1.25	0.35813	0.355393
11	1.5	0.3347	0.442214
12	1.75	0.3041	0.522168
13	2	0.27067	0.594045
14	2.25	0.23715	0.657507
15	2.5	0.20521	0.712759
16	2.75	0.1758	0.760329



Using the integration formulas 1, 2 and 3, we have, of course, different precision. The following graph shows the error behaviour of the three schemas. The average errors are about 0.05 for the 1st degree formula, 5E-5 for the 3rd degree and 4E-6 for the 5th degree. Clearly, for smooth, regular functions, the highest degree formulas reach the highest precision but we have to consider that they are also more expensive



Usually the 3rd degree formulas are the best compromise between cost and precision

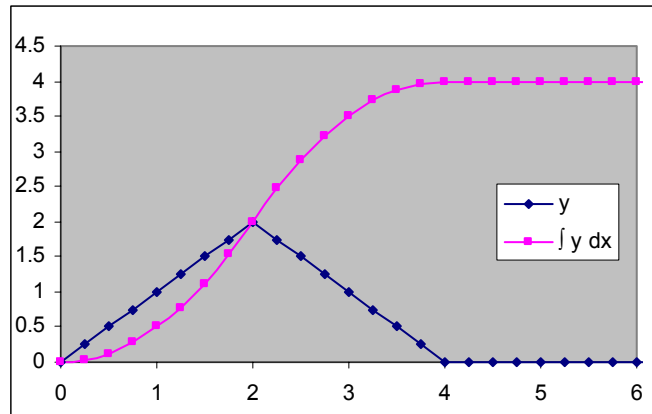
Avoid discontinuities. But, generally, do highest integration degree formulas always give the best precision? Not always. When the function to integrate shows jumps or direction discontinuities, the better choice is the linear formula. See the following example. Given the following functions

$$y(x) = (|x - 4| + x) / 2 - |x - 2|$$

its exact integral function is

$$F(x) = \int_0^x y(x) dx = [(x-4)|x-4| - 2(x-2)|x-2| + x^2 + 8] / 4$$

Sampling the function $y(x)$ and $F(x)$ with $h = 0.25$ for $0 \leq x \leq 6$, we have the following data sets $(x_i, y(x_i))$ and $(x_i, F(x_i))$



As we can see, the function $y(x)$, always linear, has no derivative at the points $x = 2$, and $x = 4$. If we integrate the data set (x_i, y_i) with the 1st degree formula ($\text{IntType} = 1$) the result coincides with the exact solution (error = 0). But if we try with the other higher formulas the average error is, surprisingly, about 0.005

Integral function of a symbolic formula

When we have the function $f(x)$ written in a symbolic formula we can obtain the plot of its integral function in two ways.

- Sampling the given function $f(x_i)$ for $x_i = x_0 + i \cdot h$ with a suitable step h
- Solving the associated ODE equation
- Integrating $f(x)$ from x_0 to x_i for $x_i = x_0 + i \cdot h$ with a suitable step h

The first method is simple and it is already shown in the previous chapter; the second way need some more explanation.

The function $y(x)$ that we want to plot is defined as

$$y(x) = \int_{x_0}^x f(x) dx$$

Taking the derivatives of both sides and remembering Leibniz's rule, we have

$$y'(x) = f(x) \quad , \quad y(x_0) = 0$$

As we can see, the computing of any integral function is equivalent of solving an ODE. Therefore we can use any method that we have used for solving ODE: Runge-Kutta, Predictor-Corrector, Taylor, etc. (see chap. ODE for further details)

The third method, less efficient than the others, may be successfully used when the function $f(x)$ or its derivatives $f'(x)$ has some singularities in the integration range. The following example explain better the concept. Assume to have to plot the integral function of the following function

$$y(x) = \int_0^x \ln(x^2) dx$$

We note that, $\lim_{x \rightarrow 0} \ln(x^2) = -\infty$

Therefore, $x = 0$ is a singular point. From theory we know that the integral exists, thus we can project to tabulate the given integral for several points $x_i = i \cdot h$ with a suitable step h , for example $h = 0.1$

For this scope we may use the function **integr_de** or **Integr** that are more suitable for such singularities

	A	B	C
1	f(x) =	Ln(x^2)	
2		=Integr_de(\$B\$1,\$A\$4,A5)	
3	x	f(x)	f(x)
4	0	0	
5	0.1	-0.66052	-4.60517
6	0.2	-1.04378	-3.218876
7	0.3	-1.32238	-2.407946
8	0.4	-1.53303	-1.832581

Insert the function definition in cell B1

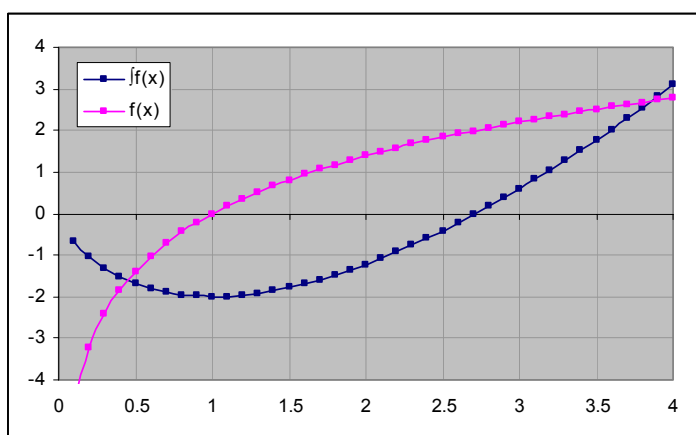
Ln(x^2)

Build a sequence x = 0, 0.1, 0.2...4 in the range A4:A44

Insert the function integr_de in cell B5 and the function = LN(A5^2) in cell C5 and drag down the range B5:C5.

Note that we cannot complete the first cell C4 because of #NUM! error.

Now, plotting the the range A5:C44, we have the following graph



Comparing with the exact solution

$$y(x) = x \ln(x^2) - 2x$$

we observe a precision better then 1E-15.

From the graph we see that the integral function y(x) has a zero between 2.5 and 3

From the table we may take a more precise bracketing of the zero

$$2.7 < x_z < 2.8$$

Zeros of integral function

We may obtain a more close solution of the integral equation using the Newton-Raphson iterative algorithm

$$y(x) = \int_{x_0}^x f(t)dt = 0$$

$$x_{n+1} = x_n - \frac{y(x_n)}{f(x_n)}$$

	A	B	C
1	f(x) =	Ln(x^2)	
2			
3	=A5-C5/B5	=LN(A5^2)	=Integr(\$B\$1,0,A5)
4	x	f(x)	y(x)
5	2.8	2.059238834	0.165868736
6	2.7194514334876	2.000860362	0.002339713
7	2.7182820799038	2.000000185	5.0289E-07
8	2.7182818284591	2	2.33399E-14
9	2.7182818284591	2	4.2671E-16

In the cell B1 insert the definition of the function f(x)

Ln(x^2)

Starting from $x_0 = 2.7$ we calculate the integral from 0 to 2.7 in the cell C5, The derivative is the function f(x) itself, calculated in the cell B5. The new value x_1 is calculated in the cell A6

Iterating the process, the solution happens in the last cells of the column A

As we can see, after few iteration the zero converges to the exact solution $x = e$ with a precision better then 1E-15

Function Integration (Romberg method)

=Integr_ro(Funct, a, b, [Param], [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 label and 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 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 function can also display 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					
				f_x	<code>{=Integr_ro(A2,B2,C2)}</code>	
	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

This result is reached with rank = 4, s = 16 sub-intervals, and an estimate error of about $E = 3.75E-16$

The function accepts also external parameters. Remember only to include the label in the parameter selection.

	A	B	C	D	E
1	f(x)	a	b	integr.	
2	x*cos(w*x)	0	2	1.3355684934371	
3					
4	label	w			
5		0.6			
6					

`=Integr_ro(A2,B2,C2;B4,B5)`

Function Integration (Double Exponential method)

= Integr_de(funcnt, a, b, [Param])

This function¹ computes the numeric integral of a function $f(x)$ by the Double Exponential method. This is specially suitable for improper integrals and infinite, not oscillating integrals.

$$I = \int_a^b f(x)dx \qquad I = \int_a^{+\infty} f(x)dx \qquad I = \int_{-\infty}^{+\infty} f(x)dx$$

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, suitable 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:

$$x(t) = \frac{b+a}{2} + \frac{b-a}{2} \tanh(\sinh(t)) \qquad h(t) = \frac{b-a}{2} \frac{\cosh(t)}{\cosh^2(\sinh(t))}$$

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 for reaching the high accuracy of 1E-14

	A	B	C	D
1	function	a	b	Integral
2	x^0.5 *(1-x)^0.3	0	1	0.474421155
3				
4	=Integr_de(A2;B2;C2)			

This function can also evaluate infinite and/or semi-infinite integral.

¹ 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

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	function	a	b	n	Integral
2	1/x^n	1	inf	1.1	10
3					
4					
5					

=Integr_de(A2;B2;C2;D1:D2)

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	function	a	b	Integral
2	Ln(x^2)	0	1	-2
3				
4				
5				

=Integr_de(A2;B2;C2)

Note that the function $f(x)$ is not defined for $x = 0$

Example. Another difficult improper integral

$$\int_0^1 \ln(x) \cdot \sqrt{x} dx = -4/9$$

	A	B	C	D
8	f(x)	a	b	integr.
9	ln(x)*sqr(x)	0	1	-0.444444444
10				
11				
12				

=Integr_de(A9;B9;C9)

Function Integration (mixed method)

= Integr(Func, 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 compute definite integrals, improper integrals and piece-wise functions integrals. The parameter "func" 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.

"func" 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 method¹ (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.

	A	B	C	D	E	F	G
1	f(x)	a	b	k	integr	=Integr(A2;B2;C2;D1:D2)	
2	x*cos(k*x)	0	0.4	2	0.067647896		
3							
4	f(x)	a	b	w	q	integr	=Integr(A5;B5;C5;D4:E5)
5	(1+w*x) / (1+q*x^2)	0	0.4	0.8	0.25	0.457544261	
6							
7							

¹ 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

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 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 the following integrals to have to compute

$$\int_0^{1/2} \frac{2}{2x^2-1} dx \quad , \quad \int_0^1 \frac{2}{2x^2-1} dx$$

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:

$$\int \frac{2}{2x^2-1} dx = \frac{\sqrt{2}}{2} \log \left(\frac{|\sqrt{2}x-1|}{|\sqrt{2}x+1|} \right) \quad \Rightarrow \quad \int_0^{1/2} \frac{2}{2x^2-1} dx = \sqrt{2} \log(\sqrt{2}-1)$$

	A	B	C	D
1	a =	0		
2	b =	0.5		
3	f(x) =	2/(2*x^2-1)		
4	integral =	-1.24645048028	=Integr(B3;B1;B2)	
5	refer. =	-1.24645048028		
6	error =	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

$$\lim_{x \rightarrow x_p^-} \log \left(\frac{|\sqrt{2}x-1|}{|\sqrt{2}x+1|} \right) = -\infty \quad , \quad \lim_{x \rightarrow x_p^+} \log \left(\frac{|\sqrt{2}x-1|}{|\sqrt{2}x+1|} \right) = -\infty$$

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 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**

	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 equidistant or random. The parameter "Degree", from 1(default) to 10, set the degree of the Newton-Cotes formula written as:

$$\int_{x_0}^{x_0 + nh} f(x)dx = \frac{h}{k} \cdot \sum_{j=0}^n f_j \cdot b_j$$

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 function IntegrDataC is very easy.

Example. Calculate the integral with the Newton-Cotes formulas of degree = 1, 2, 4, 6 for the dataset obtained by sampling the function $\sin(x)/x$ with step 0.2 . In a previous example, using the Romberg method we have approximated this integral obtaining

$$\text{Si}(1.6) \cong 1.38918048587044 \text{ with an accuracy better than } 1\text{E-}9$$

Let's see now how the Newton-Cotes formulas work.

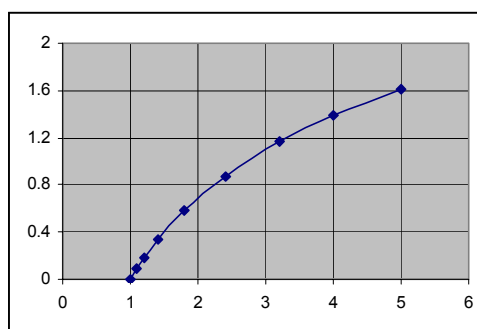
	A	B	C	D	E	F	G
1	n	x	f(x)		degree	Integral	error
2	0	0	1		1	1.387817610	1.36E-03
3	1	0.2	0.993346654		2	1.389182552	2.07E-06
4	2	0.4	0.973545856		4	1.389180464	2.21E-08
5	3	0.6	0.941070789		6	1.389180487	9.87E-10
6	4	0.8	0.896695114				
7	5	1	0.841470985				
8	6	1.2	0.776699238				
9	7	1.4	0.703892664				
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 6th degree Newton-Cotes formula. The IntegrDataC can work with random samples.

Example. Given the data table (xi yi) , 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			
3	1.1	0.09531018		integral	
4	1.2	0.182321557		4.04688	
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

$$5 \cdot \ln(5) - 4 \cong 4.0471896$$

Data integration for random points.

Having a set of not equidistant points (x_i, y_i) , 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	Equidistant Samples
$\{ (x_i, y_i) ; i = 0, 1, \dots, n \}$	\Rightarrow	$\{ (x_i = x_0 + i \cdot h, y_i(x_i)) ; i = 0, 1, \dots, m \}$

For computing the values $f(x_0 + i \cdot h)$ at the equidistant grid points, IntegrDataC uses the Aitken Interpolation algorithm.

Aitken interpolation algorithm.

Given a set of points:

$$\{ (x_i, y_i) \quad i = 0, 1, \dots, n \}$$

This method is used to find the interpolation point y_p at the value x_p . It is efficient as the Newton formula, and it is also very simple 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)

=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 of the integration interval.

The parameter "degree", from 1(default) to 10, set the degree of the Newton-Cotes formula.

degree = 1 coincides with the Trapezoidal rule; degree = 2 coincides with the Cavalieri-Simpson formula; 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 Bode's rule.

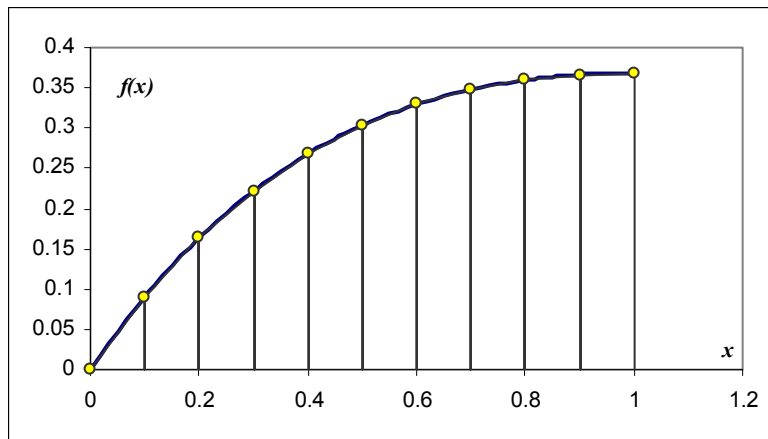
$$\int_0^1 x \cdot e^{-x} dx$$

The indefinite integral is known as the closed form:

$$\int x \cdot e^{-x} dx = -(x+1)e^{-x}$$

So we can compare the exact result, that is $1 - 2e^{-1} \cong 0.264241117657115356$

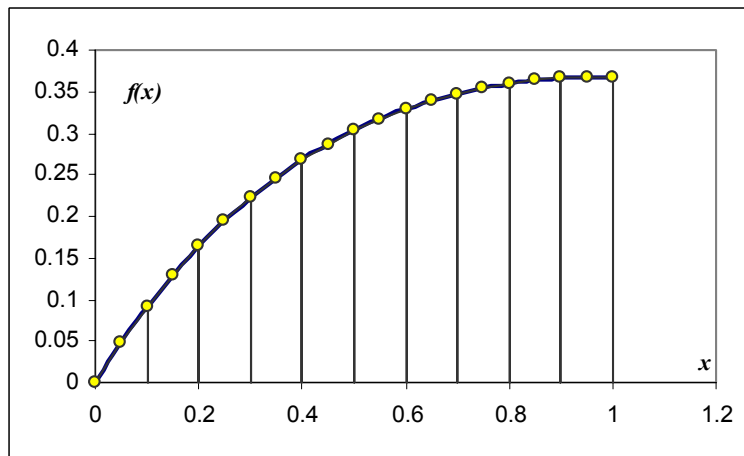
`Integr_nc("x*exp(-x)",0,1,10,1) = 0.263408098685072 (8.3E-04)`



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 $1E-3$

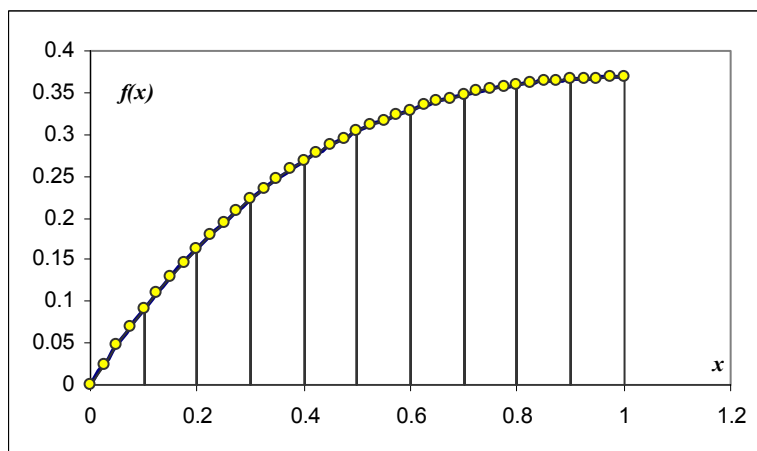
`Integr_nc("x*exp(-x)",0,1,10,2) = 0.264241039074082 (7.8E-08)`



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 methods¹

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. 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.

¹ "Improving Exact Integral from Symbolic Algebra System", R.J. Fateman and W. Kaham, University of California, Berkeley, July 18, 2000

Integration of oscillating functions (Filon formulas)

=Integr_fsin(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 are suitable to compute efficiently the following integrals.

$$\int_a^b f(x) \cos(kx) dx$$

$$\int_a^b f(x) \sin(kx) dx$$

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

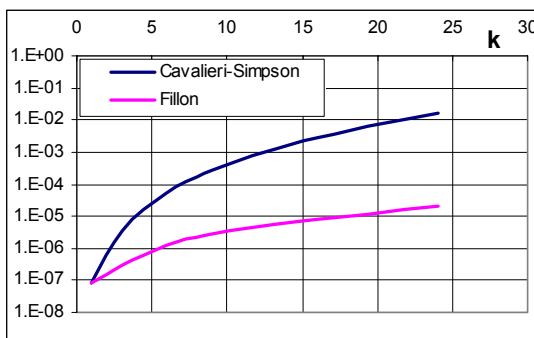
The parameter "Intervals" sets the number of sub-intervals of the integration interval.

Remember that the total number of the evaluation nodes 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 evaluate the following integral for several integer values k, with $0 < k < 25$

$$\int_0^{\pi} x^4 \cos(kx) dx$$

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 increases with the number k much more for the Cavalieri-Simpson rule than the Filon formula.

For $k = 24$ the first formula should have 400 nodes at least, for reaching the same accuracy of the Filon formula.

Example: evaluate the integral of the following oscillating function

$$\int_0^{2\pi} \frac{1}{x^2 + 1} \cdot \sin(8x) dx$$

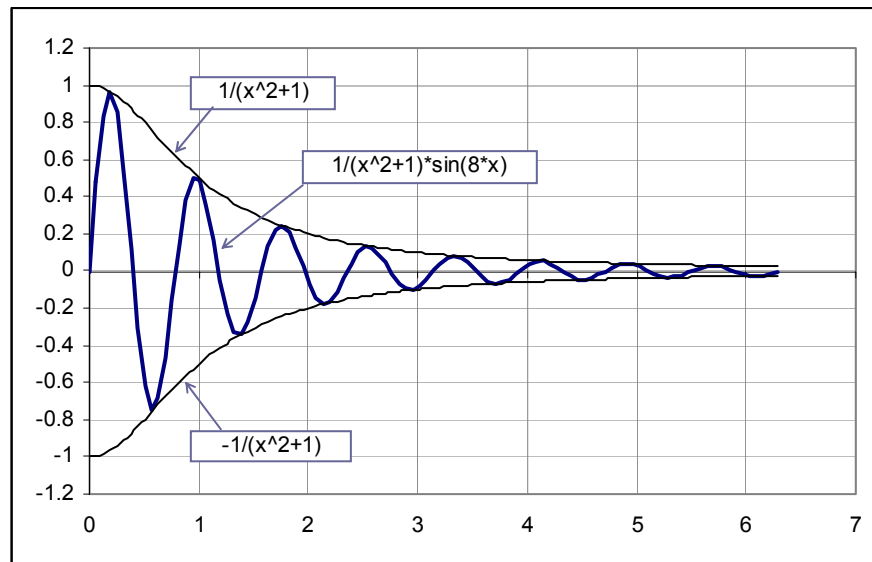
that can be rearranged as

$$\int_0^{2\pi} g(x) \cdot \sin(8x) dx$$

where

$$g(x) = \frac{1}{x^2 + 1}$$

The plot of the integration function and its 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	Fillon's integration of oscillating function					
3						
4	g(x)	a	b	k	Integral	
5	1/(x^2+1)	0	6.28318531	8	0.12692412	
6		=Integr_fsin(A5;B5;C5;D5)				
7						
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 functions¹ 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

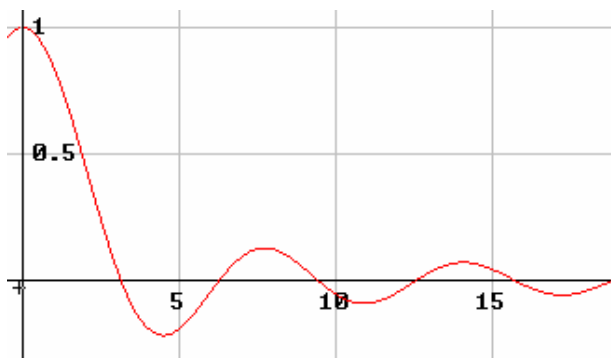
For finite integration see also [Integration of oscillating functions \(Filon formulas\)](#)

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

$$\int_0^{+\infty} \frac{\sin x}{x} dx = \int_0^{+\infty} \frac{1}{x} \sin x dx$$

That is the Fourier's sine transform of 1/x

	A	B	C
1	function	k	Integral
2	1/x	1	1.570796327
3			
4	=Fourier_sin(A2;B2)		

¹ 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

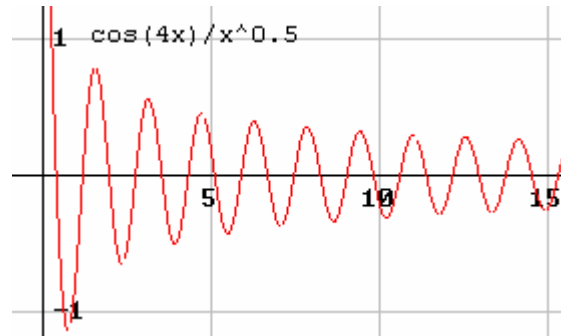
Xnumbers Tutorial

We see that the accuracy is better than 1E-15. Note that the function Fourier_sin automatically multiply the integration function f(x) for the factor sin(k*x), so we have only to pass the f(x) expression.

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 speaking 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	function	k	Integral
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.

Xnumbers Tutorial

The first integral can be evaluated with the function **integr** and the two last integrals are evaluated with the function **Fourier_cos** with $a = 1$. Let's see the following spreadsheet arrangement

	A	B	C	D	E
42	Evaluation of oscillating infinite integrals				
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

$$\int_0^{+\infty} \frac{\sin^4 x}{x^2} dx = \int_0^1 \frac{\sin^4 x}{x^2} dx + \int_1^{+\infty} \frac{3}{8x^2} dx + \int_1^{+\infty} -\frac{\cos(2x)}{2x^2} dx + \int_1^{+\infty} \frac{\cos(4x)}{8x^2} dx$$

The first and second integral can be evaluated with the function **integr** and the two last integrals are evaluated with the function **Fourier_cos** with $a = 1$.

	A	B	C	D	E
68	integration function	a	b	l	
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	l	
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	$I = l1 + l2 + l3 =$			0.785398163	

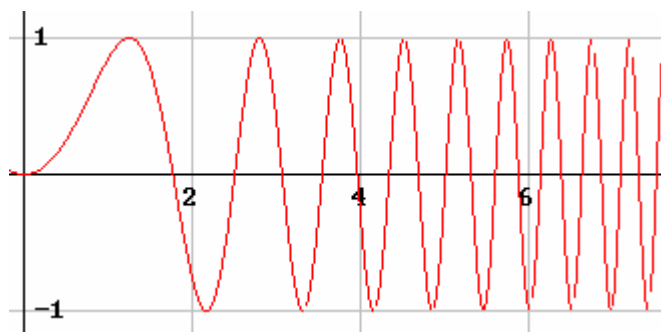
Compare the accuracy with the exact result $I = \pi / 2$

Xnumbers Tutorial

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

$$x^2 = t \quad \Rightarrow x = \sqrt{t} \quad \Rightarrow dx = \frac{1}{2\sqrt{t}} dt$$

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

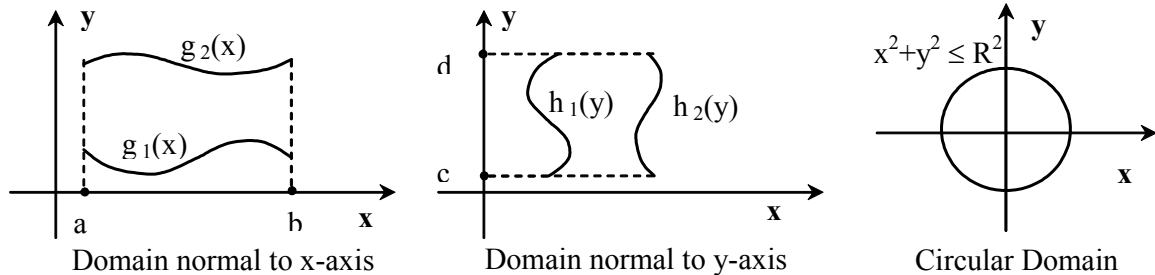
	A	B	C	D	E
82	integration function	a	k	I	
83	1/ (2 *x^0.5)	0	1	0.626657069	=Fourier_sin(A83;C83;B83)

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 macro **Integr2D** - suitable for integrating smooth functions $f(x, y)$ - and its function version **Integr_2D** use the same bidimensional Romberg algorithm, but the function is limited to about 65.000 points.

Macro for Double Integration

Integr2D()

This macro performs the numerical integration of a smooth, regular function $f(x, y)$ over a plane normal domain $D(x, y)$.

$$\int_{X_{\min}}^{X_{\max}} \int_{Y_{\min}}^{Y_{\max}} f(x, y) dy dx$$

The integration functions $f(x, y)$ and - eventually - also the bounding limits - X_{\min} , X_{\max} , Y_{\min} , Y_{\max} - can be written in symbolic expression. For further details about the math string see [Math formula string](#)

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.

Boundary limits can be:

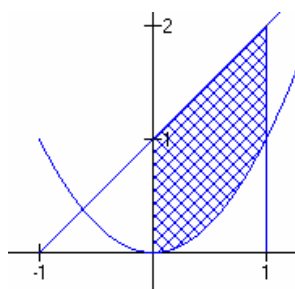
- constant numbers like 0 , 2 , 10 , 3.141 , etc.
- constant expressions like $1/2$, $\sqrt{2}+1$, π , $\sin(1/2*\pi)$, $\exp(1)$, etc.
- mono-variable functions like $x/2$, $3y-10$, x^2+x-1 , etc.

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-box of the macro panel or you can pass the cells containing the expressions. This second mode is more easy and straight

Let' see how it works.

Approximate the following double integral of the function $f(x,y) = \ln(1+x+y)$ in the closed region delimited by the given constrains



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 an x-nomal-domain domain. Verify that the given integral approximates the symbolic expression at the right

$$\iint_{D(x,y)} f(x,y) ds = \int_0^1 \int_{x^2}^{x+1} \ln(1+x+y) dy dx$$

$$-\frac{9\ln(3)}{4} + 7\ln(2) - \frac{\pi\sqrt{3}}{12} - \frac{17}{8}$$

	A2				
	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					

The macro assumes as default the following simple arrangement (but, of course, it is not obligatory at all)

Select the A2 cell and start the **Integr2D** macro.

Double Integral

Integration function $f(x,y)$

\$A\$2

Run

Help

min

max

x

\$B\$2

\$C\$2

y

\$D\$2

\$E\$2

Parameters

Error: 1E-12

Polar ☐

Rank: 10

Output to:

\$A\$4

As we can see, the entire input boxes are filled with the right references. The output result will start from the A4 cell

Optionally we can adjust the Error limit or the Rank. But usually the only thing to do is click on the "run" button

Warning: The Rank increase exponentially the computation effort, because $\text{points} = (2K)^2$

"Parameter" input box may be used to pass values of one or more parameters eventually present in the integration function

Activate "polar" for switching to the polar coordinate system

The macro outputs 5 results:

- 1) Integral
- 2) Relative error estimation
- 3) Total points evaluated
- 4) Elaboration time
- 5) Error message (if any)

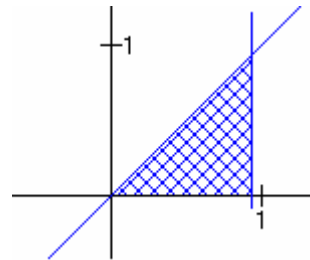
	A	B	C	D	E
1	$f(x,y)$	a	b	c	d
2	$\ln(1+x+y)$	0	1	x^2	$x+1$
3					
4	Integral	Err. rel.	Points	Time	
5	0.98225833	7E-13	4225	0.0547	

Example 2. Approximate the double integral of the function

$$[1 + 0.218(y - x)]^{-3.148} \cdot e^{-0.852x}$$

in the closed region delimited by

$$0 \leq x \leq 1, \quad 0 \leq y \leq x$$



The above expression can be written as

$$(1 + 0.218(y - x))^{-3.148} \cdot \exp(-0.852x)$$

or alternatively

$$(1 + 218/1000(y - x))^{-3148/1000} \cdot \exp(-852/1000x)$$

In both cases the result is obtained with high precision

	A	B	C	D	E
1	$f(x,y)$	a	b	c	d
2	$(1 + 0.218(y - x))^{-3.148} \cdot \exp(-0.852x)$	0	1	0	x
3					
4	Integral	Err. rel.	Points	Time	
5	0.3674358895634	5.77E-14	4225	0.523438	

Another elegant way to insert constant in the integration function is using parameters
Example. Evaluate numerically the following double integral

$$\int_0^\pi \int_0^\pi \sin(nx) \cos(my) dx dy$$

for $n = 0.75$ and $m = 0.125$

Set two cells, for example, F2 and G2, containing the values of n and m ; just above, in the cells F1 and G1, add their name " n " and " m " and pass the entire range F1:G2 as parameters.

Parameters

\$F\$1:\$G\$2

F	G
n	m
0.75	0.125

The result will be

	A	B	C	D	E	F	G
1	f(x, y)	Xmin	Xmax	Ymin	Ymax	n	m
2	sin(n*x)*cos(m*y)	0	pi	0	pi	0.75	0.125
3							
4	Integral	Err. rel.	Points	Time			
5	6.968335813	1.13E-13	4225	0.06055			
6							

Compare with the exact result $\frac{1}{3}\sqrt{128\sqrt{2} + 256}$

Macro for Triple Integration

Integr3D()

3D Integration in normal domains

This macro calculates the triple integral of a function $f(x, y, z)$ in a normal domain or in a spherical domain or in a cylindrical domain.

For this kinds of domains the triple integral can be decomposed by the following formulas

$$\iiint_D f(x, y, z) dx dy dz = \int_{x_{\min}}^{x_{\max}} \int_{y_{\min}}^{y_{\max}} \int_{z_{\min}}^{z_{\max}} f(x, y, z) dx dy dz$$

Where the functions:

- Xmin and Xmax may depend on variables y and z
- Ymin and Ymax may depend on variables x and z
- Zmin and Zmax may depend on variables x and y

The normal domain condition implies that, at least, one boundary must be constant

The macro can integrate in spherical coordinates system, using the formulas

$$x = \rho \sin \varphi \cos \theta$$

$$y = \rho \sin \varphi \sin \theta$$

$$z = \rho \cos \varphi$$

where θ , from 0 to 2π , is the rotation angle on the xy-plane and φ , from 0 to π , is the colatitude angle

The triple integral changes in

$$\iiint_D f(x, y, z) dx dy dz = \int_{\rho_{\min}}^{\rho_{\max}} \int_{\theta_{\min}}^{\theta_{\max}} \int_{\varphi_{\min}}^{\varphi_{\max}} f(x, y, z) \rho^2 \sin \varphi d\varphi d\theta d\rho$$

The macro can also integrate in cylindrical coordinates system, using the formulas

$$x = \rho \cos \theta$$

$$y = \rho \sin \theta$$

$$z = z$$

where θ , from 0 to 2π , is the rotation angle on the xy-plane
The triple integral changes in

$$\iiint_D f(x, y, z) dx dy dz = \int_{\rho_{\min}}^{\rho_{\max}} \int_{\theta_{\min}}^{\theta_{\max}} \int_{z_{\min}}^{z_{\max}} f(x, y, z) \rho d\rho d\theta dz$$

Functions $f(x, y, z)$ and – eventually – also the boundary functions may be written in symbolic expression..

For further details about the math string see [Math formula string](#)

Integration function can be:

- three-variate functions like x^2+y^2-x*z , $\log(1+x+y+z)$, $(x+z)/(1+x^2+y^2)$, etc.
- Constant numbers like 0, 2, 1.5, 1E-6, etc.
- Constant expressions like $1/2$, $\sqrt{2}+1$, $\sin(\pi/4)$, etc.

Boundary limits can be:

- Constant numbers like 0, 2, 10, 3.141, etc.
- Constant expressions like $1/2$, $\sqrt{2}+1$, π , $\sin(1/2*\pi)$, $\exp(1)$, etc.
- univariate or bivariate functions like $x/2$, $3y-10$, x^2+x-1 , $\sin(x+z)$, $\ln(x+y)$, etc.

A normal domain has, at least, two constant boundary limits.

The Integration function and the boundary limits can be passed to the macro directly or by reference. That is: we can write directly the symbolic expressions into the input fields, or you can pass the cells that containing the expressions (simpler), or even a mixed mode.

Let's see how it works

Example 1. Approximate numerically the following triple integral

$$\int_0^1 \int_0^2 \int_0^3 \sqrt{54xyz} \, dx dy dz$$

The integration domain is the parallelepiped of lengths 1, 2, 3

The macro assumes as default the following simple layout (but, of course, it is not obligatory)

	A	B	C	D	E	F	G
1	$f(x,y,z)$	Xmin	Xmax	Ymin	Ymax	Zmin	Zmax
2	$\text{sqr}(54*x*y*z)$	0	1	0	2	0	3
3							

Now select the cell A2 containing the integration function and start the macro from the Xnumbers toolbar **Macros > Integral > Triple**.

As we can see, all the input boxes are filled with the right references.

The output result will start from the A4 cell

Optionally we can adjust the Rank, increasing it, if we want to increase the final accuracy. But usually the only thing to do is click on the "run" button

Warning: The Rank increase exponentially the computation effort, because the number of max points is $(2 \cdot \text{Rank})^3$

Parameter input box may be used to pass values of one or more parameters eventually present in the integration function

The macro outputs 5 results:

- 1) Integral
- 2) Relative error estimation
- 3) Total points evaluated (function + boundaries)
- 4) Elaboration time
- 5) Error message (if any)

The results will appear as the following

	A	B	C	D	E	F	G
1	$f(x,y,z)$	Xmin	Xmax	Ymin	Ymax	Zmin	Zmax
2	$\text{sqr}(54 \cdot x \cdot y \cdot z)$	0	1	0	2	0	3
3							
4	Integral	Err. rel.	Points	Time			
5	32	1.737E-14	14165	0.83594			

Example 1. Approximate numerically the following triple integral

$$\iiint_D \frac{1}{\pi} (x^2 + y^2) dx dy dz$$

where D is the domain defined by the following implicit relation

$$x^2 + y^2 + z^2 \leq 1$$

This domain is a sphere of unitary radius. Therefore is better to pass to the spherical coordinates because the integration domain becomes much simpler, being

$$0 \leq \rho \leq 1 \quad , \quad 0 \leq \theta \leq 2\pi \quad , \quad 0 \leq \varphi \leq \pi$$

The macro automatically performs the spherical transformation; so we have only to set the right boundaries

Select the cell A2 and start the macro. Select "Polar" and set the Rank = 25. Press "Run"

	A	B	C	D	E	F	G
1	f(x,y,z)	Xmin	Xmax	Ymin	Ymax	Zmin	Zmax
2	(x^2+y^2)/pi	0	1	0	2*pi	0	pi
3							
4	Integral	Err. rel.	Points	Time			
5	0.533333335	2.088E-07	14865	0.92969			

Compare the result with the exact solution $8/15 = 0.53333...$

Double integration function

=Integr_2D (Fxy, Xmin, Xmax, Ymin, Ymax, [Polar],[ErrMax])

This function returns the numeric integral of a smooth regular function $f(x, y)$ over a plane normal domain $D(x, y)$.

$$\int_{X_{\min}}^{X_{\max}} \int_{Y_{\min}}^{Y_{\max}} f(x, y) dy dx$$

The integration functions $f(x, y)$ and – eventually – also the bounding limits – Xmin, Xmax, Ymin, Ymax -can be written in symbolic expression. For further details about the math string see [Math formula string](#)

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 $1/2$, $\sqrt{2}+1$, π , $\sin(1/2*\pi)$, $\exp(1)$, etc.
- monovariate functions like $x/2$, $3y-10$, x^2+x-1 , etc.

A normal domain has, at least, two constant boundary limits.

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.

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

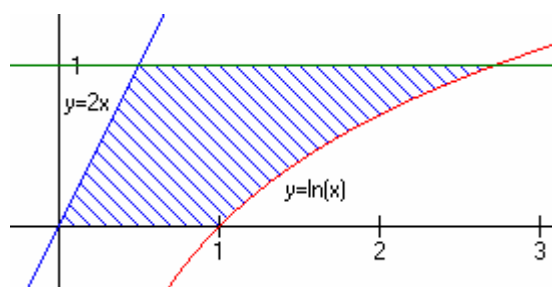
Xnumbers Tutorial

Note: 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-axis

The calculus 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							

Sometime, the integral may contain one or more parameters

Example

$$\int_0^1 \int_0^1 e^{(a \cdot x^2 + b \cdot y^2)} dx dy$$

	A	B	C	D	E	F	G	H
1	f(x, y)	Xmin	Xmax	Ymin	Ymax	a	b	
2	EXP(a*x^2+b*y^2)	0	1	0	1	-0.5	-0.125	
3								
4	Integral	Rel.Err.	Points	{=Integr_2D(A2,B2,C2,D2,E2,F1:G2)}				
5	0.821271447	5.89E-12	1089					
6								

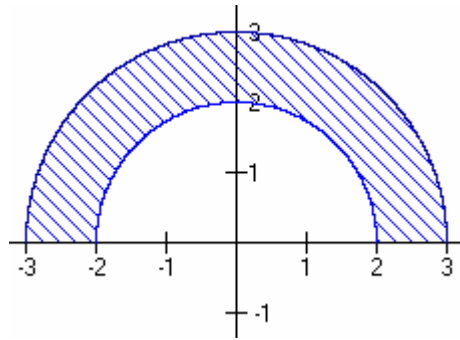
Note that the parameters "a" and "b" must be passed with their labels.

Xnumbers Tutorial

Example. Compute numerically the following integral

$$\iint_D \sqrt{x^2 + y^2} dx dy$$

Where the domain D is the half-circular region showed at the right



Set the parameter "Polar" to True.

	A	B	C	D	E
1	f(x, y)	ρ min	ρ max	θ min	θ max
2	sqr(x^2+y^2)	2	3	0	pi
3					
4	Integral	{=Integr_2D(A2,B2,C2,D2,E2,,TRUE)}			
5	19.89675347				

Compare the precision with the exact result

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. 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. Therefore 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)

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

	A	B	C	D
1				
2	Integr. function f(x)	a	b	
3	100*x^2*exp(-x)	0	inf	
4				
5				
6				

Choose "run" to start the integration routine. The result will be similar to the worksheet below (without formatting) where we have compared the results of both methods

	A	B	C	D	E	F
1						
2	Integration function f(x)	a	b			
3	100*x*exp(-x)	0	inf			
4						
5	Integral	Err. rel.	Points	Time		
6	100	1.6E-13	199	0.015625	(double exponential)	
7	100	6.57E-18	2048	0.039063	(variable step)	
8						

As we can see the integral is 100 with an 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

$$\int_{-\infty}^{+\infty} \frac{x^2 + 2x + 3}{x^4 + x + 1} dx$$

	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						

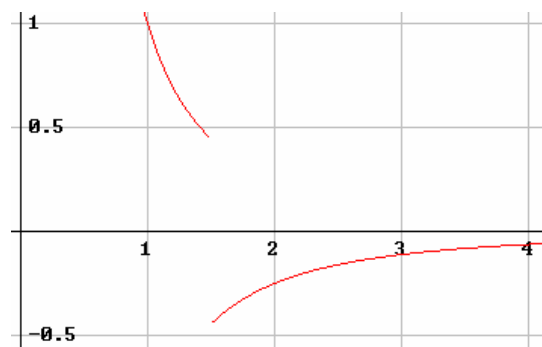
Note that, in this case, we have needed more than 10.000 evaluation points for the variable step method but only 344 for the DE method. The superiority is ever so evident? Not always. There are cases in which the adaptive quadrature schema works better. For example when the integration function has a 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 shown 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	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)	

As we can see the variable step method has find the result with high precision using about 7200 steps. The double exponential algorithm fails the convergence

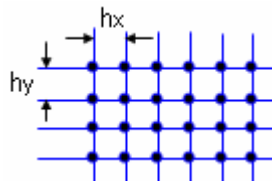
We have to put in evidence that using quadrature algorithms in a “blind” way, may lead to wrong result. We should always study the integration function to discover singularities, discontinuities, convergence rate, etc. If the integration function is “sufficiently” smooth, then the numeric integration 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.

Double Data integration

=IntegrData2D(Dataxy, hx, hy)

Given a bidimensional set of points (x_i, y_j) defined by a rectangular grid



$$D \equiv \{(x_i, y_j) \mid x_i = x_0 + ih_x, y_j = y_0 + jh_y, i = 0 \dots n, j = 0 \dots m\}$$

this function computes the numerical double integral over the domain D

$$\iint_D f(x, y) dx dy$$

"Dataxy" is a $(n \times m)$ rectangular array containing the function values $f(x_i, y_j)$

"hx" and "hy" are the grid intervals respectively of the x-axis and the y axis

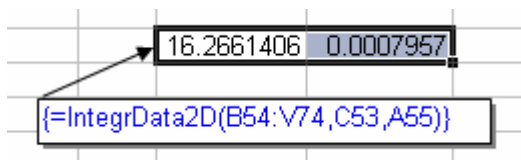
Using this function is very simple.

Example assuming to have the following rectangular dataset (x, y) in wich each cell represent a point of the grid and thus a value of the function $f(x_i, y_j)$ where $i = 0, 1, 2 \dots 20, j = 0, 1, 2 \dots 20$

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
53																								
54																								
55																								
56																								
57																								
58																								
59																								
60																								
61																								
62																								
63																								
64																								
65																								
66																								
67																								
68																								
69																								
70																								
71																								
72																								
73																								
74																								
75																								
76																								

the function `=IntegrData2D(B54:V74, C53, A55)` returns the value 16.2661406
 The scales x and y at the border of the range are not necessary for the function IntegrData2D.
 They are designed only for clarity.

This function can also return the estimate error as second, optional value. To see also the error
 select two adjacent cells and insert the function with the CTRL+SHIFT+ENTER keys



The given data set was generated by the function

$$f(x, y) = e^{-x/3} + e^{-y}$$

with $x_i = 0.3 i$ and $y_i = 0.2 j$

Therefore the above result is an approximation of the following integral

$$\int_0^3 \int_0^4 e^{-x/3} + e^{-y} dy dx$$

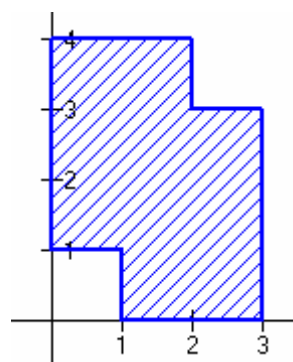
that is about 16.266082.

One interesting feature of this function is that
 it accepts piecewise-rectangular domains

For example. Assume you have to
 approximate the following integral

$$\iint_D \ln(x + 2y + 1)$$

where D is the domain shown at the right

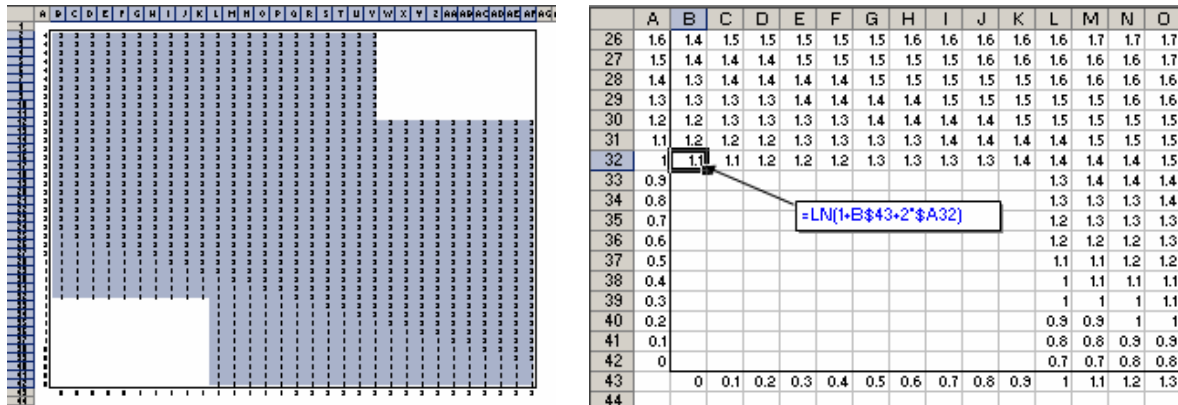


This integral could be split, of course, into three rectangular integrals

$$\int_0^2 \int_3^4 f(x, y) dy dx + \int_0^3 \int_1^3 f(x, y) dy dx + \int_1^2 \int_0^1 f(x, y) dy dx$$

which the result is about 18.18803389

But it could be approximated in a worksheet sampling the domain for $0 \leq x \leq 3$ and $0 \leq y \leq 4$ with a suitable grid, for example $h_x = 0.1$, $h_y = 0.1$



The left spreadsheet shows a grid of data points for a function. The right spreadsheet shows a zoomed-in view of the grid with a formula cell.

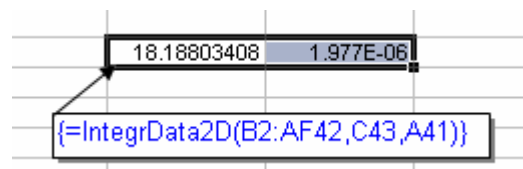
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
26	1.6	1.4	1.5	1.5	1.5	1.5	1.5	1.6	1.6	1.6	1.6	1.6	1.7	1.7	1.7
27	1.5	1.4	1.4	1.4	1.5	1.5	1.5	1.5	1.5	1.6	1.6	1.6	1.6	1.6	1.7
28	1.4	1.3	1.4	1.4	1.4	1.4	1.5	1.5	1.5	1.5	1.5	1.6	1.6	1.6	1.6
29	1.3	1.3	1.3	1.3	1.4	1.4	1.4	1.4	1.5	1.5	1.5	1.5	1.5	1.6	1.6
30	1.2	1.2	1.3	1.3	1.3	1.3	1.4	1.4	1.4	1.4	1.5	1.5	1.5	1.5	1.5
31	1.1	1.2	1.2	1.2	1.3	1.3	1.3	1.3	1.4	1.4	1.4	1.4	1.5	1.5	1.5
32	1	1.1	1.1	1.2	1.2	1.2	1.3	1.3	1.3	1.3	1.4	1.4	1.4	1.4	1.5
33	0.9											1.3	1.4	1.4	1.4
34	0.8											1.3	1.3	1.3	1.4
35	0.7											1.2	1.3	1.3	1.3
36	0.6											1.2	1.2	1.2	1.3
37	0.5											1.1	1.1	1.2	1.2
38	0.4											1	1.1	1.1	1.1
39	0.3											1	1	1	1.1
40	0.2											0.9	0.9	1	1
41	0.1											0.8	0.8	0.9	0.9
42	0											0.7	0.7	0.8	0.8
43		0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3
44															

The cells out of the domain must be left empty.

The entire range is now B2:AF42 but the function will compute the integral only over the not empty cells.

Note that in that case the scales x and y at the border of the range are used for calculating the function $f(x_i, y_i)$

The integral and its estimate error will be



The screenshot shows the result of the IntegrData2D function. The formula bar displays the formula `=IntegrData2D(B2:AF42,C43,A41)`. The result is displayed in two cells: 18.18803408 and 1.977E-06.

Note the good accuracy of this result.

IntegrData2D use the bidimensional trapezoidal formula with one-step of the Richardson extrapolation. This algorithm works very fine with smooth regular functions and its precision is similar the Cavalieri-Simpson formula

Series Evaluation

=xSerie(Funct, Id, a, b, [Param], [DgtMax])

Returns the numeric series of a function $f(n)$.

$$S = \sum_{n=a}^b f(n)$$

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.)

"a" and "b" are the limits of the sum.

The function may also have 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. Compute

$$\sum_{n=1}^{10} (-1)^{n+1} \cdot \frac{x^n}{n}$$

for $x = 2$, with standard precision (15 digits) and with 25 digits.

The function substitutes $x = 2$ and then, computes the series $f(n)$ for $n = 1, 2 \dots 10$

$$\sum_{n=1}^{10} (-1)^{n+1} \cdot \frac{2^n}{n} = 2^1 - \frac{2^2}{2} + \frac{2^3}{3} - \frac{2^4}{4} \dots - \frac{2^{10}}{10}$$

`xSerie("(-1)^(n+1)*x^n/n","n",1,10,2) = -64.8253968253968`

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	F(x, n)	where	from	to	for x =
2	$x^n/n!$	n	0	10	-1.5
3	Σ				
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					

The function xSerie accepts one or more parameters.

Example 3. Compute the following series where a and b are parameters

$$s = \sum_{n=1}^{10} \frac{b \cdot n + a}{n}$$

for a = 0.7 and b = 1.5, in standard precision

	A	B	C	D	E	F
1	F(n,a,b)	index	from	to	Σ	
2	(b*n+a)/n^2	n	1	10	5.478289793	
3		Parameters				
4		a	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 this calculation using the **xSerie** function.

Series acceleration with Δ^2

Many series are very slow to converge requiring therefore methods to accelerate their convergence. The Aitken's extrapolation formula (Δ^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	f(k)=	(-1)^k/(k+1)	
2			
3	k	Sk	Error
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 insert the formula

=Series(\$B\$1;"k";0;A4)

In the cell C4 we insert

=(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 Δ^2 extrapolation using the last 7 values of the sum

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 Δ^2 extrapolation at the last seven partial sums $S^{(12)}$, $S^{(11)}$, $S^{(10)}$ $S^{(6)}$ 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, a, b, [z0])

This function returns the numeric series of a complex function $f(z, n)$.

$$S = \sum_{n=a}^b f(z, n)$$

"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 "a" and "b" 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		fx {=cplxserie(B2;B3;B4;E2:F2)}				
	A	B	C	D	E	F
1	Complex Series				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, a, b, Id2, c, d, [Param], [DgtMax])

Returns the numeric double series of a function $f(n, m)$.

$$S = \sum_{m=a}^b \sum_{n=c}^d f(m, n)$$

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.)

"a" , "b" , "c" , "d" set the minimum and the maximum limits of the integer variables n and m.

The function may have 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. 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	Σ				
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], [smooth])

It returns the trigonometric series

$$f(t) = f(0) + \sum_{n=1}^N a_n \sin(n\omega t + \theta_n)$$

$$\omega = \frac{2\pi}{T}$$

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.

The optional parameter "offset" is the average level (default 0)

The optional parameter "Angle" sets the angle unit: (RAD (default), DEG, GRAD)

Optional parameter *smooth* (default False) applies the Lanczos factor to the trigonometric series

$$f(t) = f(0) + \sum_{n=1}^N \sigma_n a_n \sin(n\omega t + \theta_n)$$

$$\sigma_n = \frac{\sin(n\pi / N)}{n\pi / N}$$

	A	B	C	D	E	F
1	Harmonic Analysis				t	y(t)
2	t0 =	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)}					

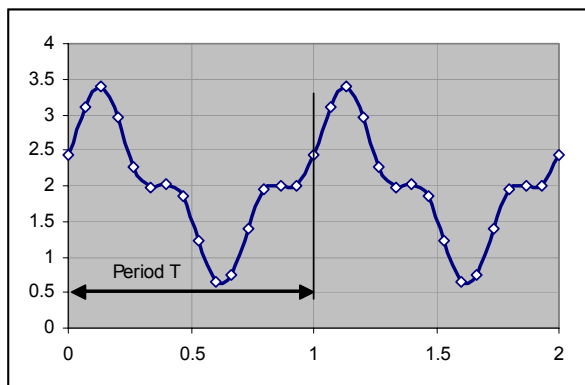
Here is a worksheet arrangement to tabulate a trigonometric series having a spectrum of max 8 harmonics (the formulas inserted are in blue)

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



Period T = 1

n°	Arm.	Amp	Phase
1	1	1	45
2	0	0	0
3	0.4	0.4	-45

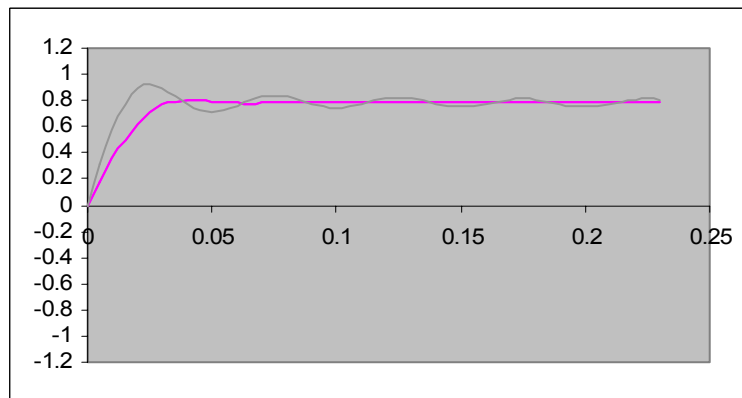
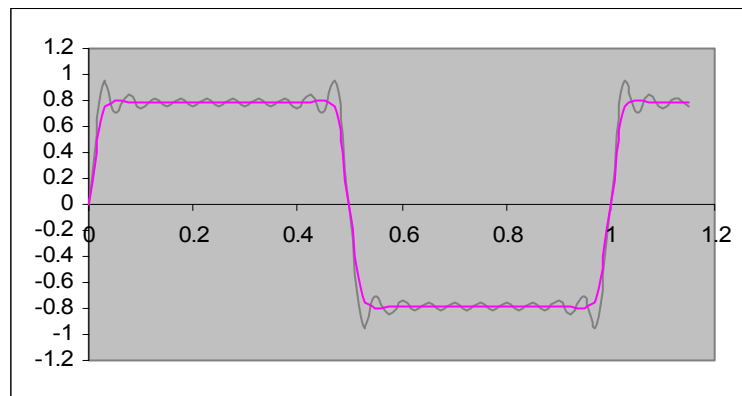
Note that we can always transform the cosine terms into sine with the following formula

$$\cos(\alpha) = \sin(\alpha + \pi / 2)$$

The “**ringing**” phenomenon is an overshoot of Fourier series occurring at simple discontinuities (jumps). The series does not converge uniformly to a discontinuous function in a small interval of the discontinuity point. It can be proven that, at each discontinuity point, the series either overshoots or undershoots by about 9% of the magnitude of the jump. The ringing can be removed with the Lanczos sigma factors.

Example. The phenomenon is illustrated for a square wave.

Given the spectrum of the first 20 harmonics $A_n = \{ 0, 1, 0, 1/3, 0, 1/5, 0, 1/7 \dots 1/19, 0 \}$ and $\theta_n = 0$, of the square wave with period $T = 1$, let's plot its Fourier series with and without Lanczos sigma factors (parameter smooth = true / false respectively)



As we can see, the series with Lanczos factors (pink line) shows a more stable and smooth behavior. Note, however, that ringing has interesting physical consequences. Consider, for example, a linear electric circuit in which, by means of a switch, a fast voltage transition is created: then the response of the circuit will really exhibit an overshoot

Trigonometric double serie

= Serie2D_trig(x, y, Lx, Ly, Spectrum, [offset], [Angle])

It returns the trigonometric double serie

$$f(x, y) = f_0 + \sum_{n=1}^N \sum_{m=1}^M a_{n,m} \cos(n\omega_x x + m\omega_y y + \theta_{n,m})$$

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" is 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.

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											
11		0	0.0625	0.125	0.1875	0.25	0.3125	0.375	0.4375	0.5	0.5625
12	0.0625	2.4098	2.3137	1.8754	1.4938	1.3579	1.3561	1.3559	1.4293	1.7115	2.0000
13	0.125	1.9791	1.6786	1.2051	0.9443	0.9664	1.0856	1.1895	1.3728	1.7101	2.0000
14	0.1875	1.5149	1.1235	0.7267	0.6323	0.7801	0.9426	1.0539	1.2293	1.4851	1.8000
15	0.25	1.1722	0.6902	0.3497	0.3461	0.5118	0.6252	0.6871	0.822	0.9863	1.2000
16	0.3125	0.6879	0.1209	-0.151	-0.07	0.1177	0.2242	0.3081	0.472	0.605	0.7500
17	0.375	0.051	-0.455	-0.537	-0.292	-0.026	0.1246	0.2576	0.4313	0.4839	0.5000
18		-0.326	-0.627	-0.466	-0.089	0.1856	0.2988	0.3798	0.447	0.3258	-0.1000

Discrete Convolution

Conv(f, g, h, [algo])

This function approximates the convolution of two sampled functions $f(t_i)$, $g(t_i)$

$$f * g = \int_{-\infty}^{+\infty} f(v)g(t-v)dv$$

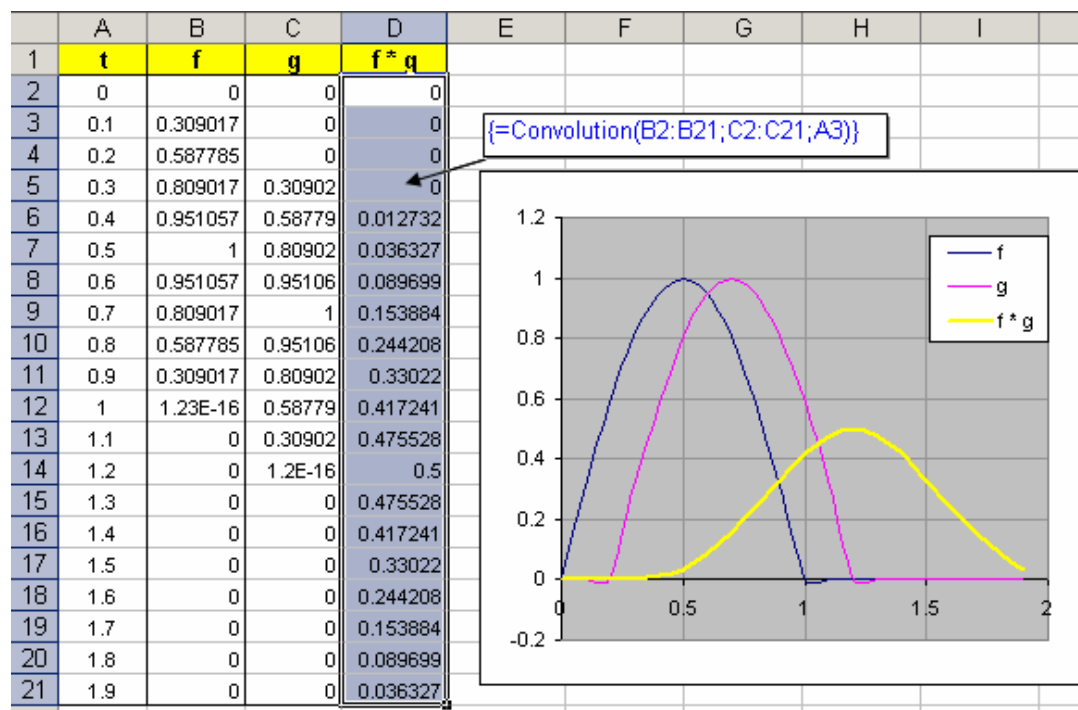
The parameters "f" and "g" are column-vectors

The parameter "h" is the sampling step.

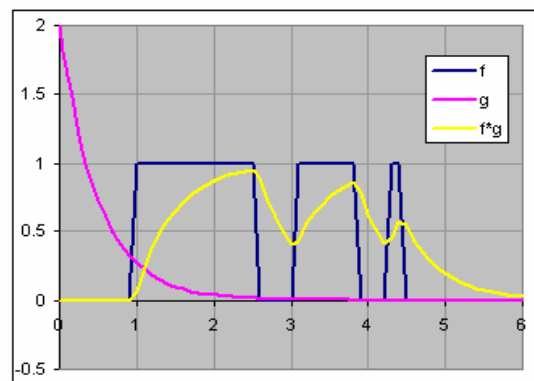
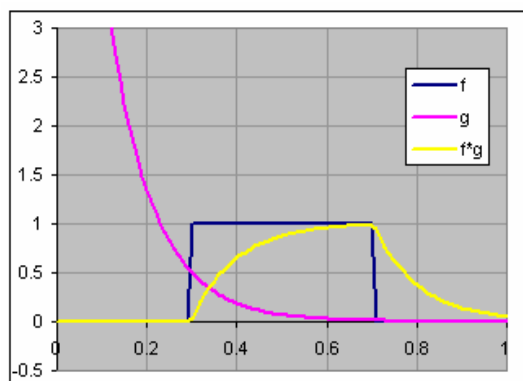
The optional parameter "algo" sets the algorithm: 0 = discrete, 1 = linear (default), 2 = parabolic.

The functions returns a vector with the same dimension of the two vectors f and g.

The convolutions is also called "Faltung"

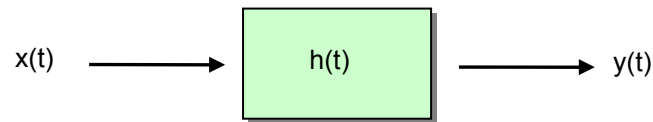


Here are other examples of convolution plots



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:

$$y(t) = \int x(v) h(t-v) dv$$

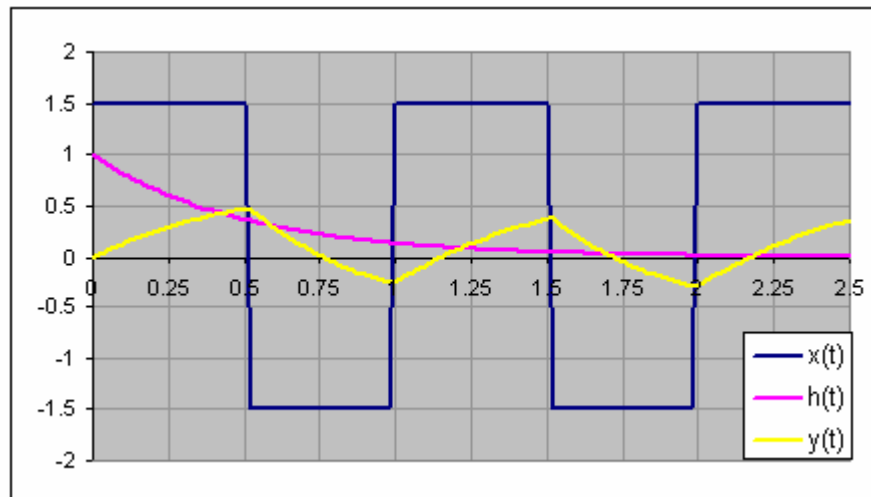
If the system is described by the following differential equation

$$y'(t) + k \cdot y(t) = x(t)$$

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. You can choose the size that you like in order to obtain the needed accuracy.

Interpolation

Polynomial interpolation

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

=PolyInterpCoef(xi, yi, [degree], [DgtMax])

The first function performs the interpolation of a given set of points (x,y), and returns the value at the point x.

The Input parameters Xi and Yi are vectors.

The parameter "degree" sets the interpolation polynomial degree (default = 1)

The optional parameter DgtMax sets the maximum digits of the multiprecision arithmetic. If omitted or 0, the functions works in faster double precision.

The second function returns an array containing the coefficients of the polynomial interpolation. Use CTRL+SHIFT+ENTER for inserting this function.

This function 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 $D(x_1, \dots, x_n)$ is the "divided difference", given by the following recursive formulas:

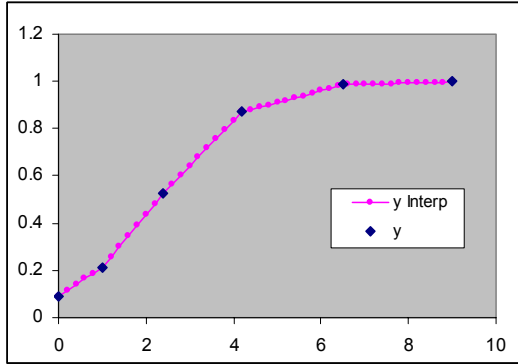
$$D(x_1, x_2) = \frac{y_1 - y_2}{x_1 - x_2}, \quad D(x_2, x_3) = \frac{y_2 - y_3}{x_2 - x_3}, \dots$$

$$D(x_1, x_2, x_3) = \frac{D(x_1, x_2) - D(x_2, x_3)}{x_1 - x_3}, \dots, D(x_1, \dots, x_m) = \frac{D(x_1, \dots, x_{m-1}) - D(x_2, \dots, x_m)}{x_1 - x_m}$$

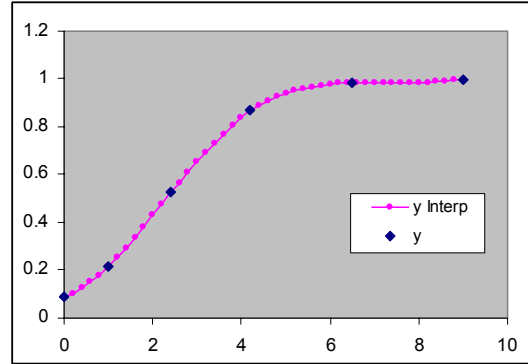
Example : Compute the sub-tabulation between a set of 6 given points (x, y) using a linear polynomial and a 3rd degree polynomial and a step of h = 0.2. Note that the given points are not equidistant.

	A	B	C	D	E	F
1	Sub-Tabulation			h	0.2	
2				degree	3	
3						
4	x	y		x	y	
5	0	0.09091		0	0.09091	
6	1	0.21373		0.2	0.10468	
7	2.4	0.52433		0.4	0.12434	
8	4.2	0.8696		0.6	0.14949	
9	6.5	0.98519		0.8	0.17948	
10	9	0.99877		1	0.21373	
11				1.2	0.25167	
12	=PolyInterp(D5,\$A\$5:\$A\$10,\$B\$5:\$B\$10,\$E\$2)					
13				1.6	0.3363	

Sub-Tabulation with Degree = 1

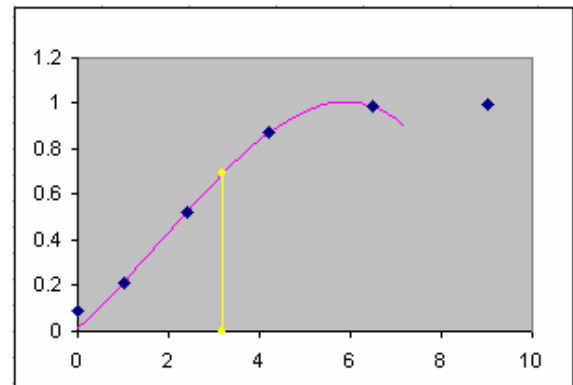


Sub-Tabulation with Degree = 3



If we wonder which is the 3rd degree polynomial used for interpolating the point $x = 3.2$ we can use the function PolyInterpCoef.

	A	B	C	D	E
1	Sub-Tabulation			x interp	degree
2				3.2	3
3					
4	x	y			coeff.
5	0	0.090909		a0	0.015403454
6	1	0.21373		a1	0.1775526
7	2.4	0.524334		a2	0.025344493
8	4.2	0.869599		a3	-0.00457028
9	6.5	0.985188			
10	9	0.998767			
11	{=PolyInterpCoef(D2,A5:A10,B5:B10,E2)}				
12					



The interpolation polynomial will be

$$p(x) = a_0 + a_1x + a_2x^2 + a_3x^3$$

with the coefficients $[a_0, a_1, a_2, a_3]$ returned from the function PolyInterpCoef

We can plot this polynomial by the function Polyn. The interpolation polynomial is shown in the above graph.

Note that the function has selected the 4 points $[x_2, x_3, x_4, x_5]$, discharging the first point x_1 and the last point x_6 , in order to get the most possible accurate interpolation.

Note. when the number of the points (x, y) is exactly $\text{degree}+1$, then the polynomial is univocally determined no matter which the interpolation point is: we can pass any number x that we like, for example $x = 0$.

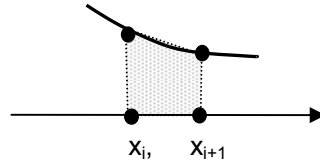
In the following example we determine the 2nd degree polynomial interpolating the given 3 points with 30 significant digits

	B7		fx {=PolyInterpCoef(0,A2:A4,B2:B4,2,30)}		
	A	B	C	D	
1	x	ln(x)			
2	1.5	0.405465108108164381978013115464			
3	2	0.693147180559945309417232121458			
4	3	1.09861228866810969139524523692			
5					
6		coefficients			
7	a0	-0.797379182837973346140493695565			
8	a1	0.97179523075948929164609610388			
9	a2	-0.113266024530264981933616597684			

Interpolation schemas

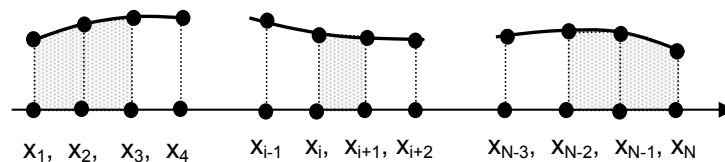
If the number N of the points (x, y) is exactly equal to $\text{degree}+1$, then the polynomial is univocally determined; but if $N > \text{degree}+1$ there are several different polynomials that can be used for interpolating a given point x . A popular method, adopted by the function PolyInterp is called "sliding centered polynomial"

Linear interpolation



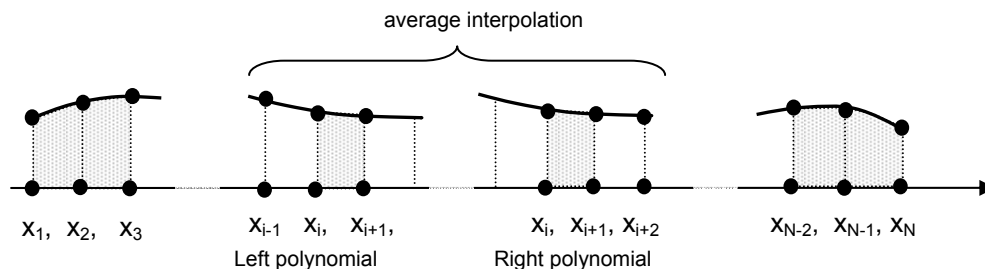
The linear interpolation is always performed between the nodes x_i, x_{i+1} where $x \in [x_i, x_{i+1}]$. For degree > 1 the interpolation strategy is more complicated.

Odd degree polynomial (example degree = 3)



The 3rd degree interpolation is performed between the nodes $\{x_{i-1}, x_i, x_{i+1}, x_{i+2}\}$ where $x \in [x_i, x_{i+1}]$. At the first of the segment, the interpolation is performed between the nodes $\{x_1, x_2, x_3, x_4\}$ where $x \in [x_1, x_3]$. At the end, the interpolation is performed between the nodes $\{x_{N-3}, x_{N-2}, x_{N-1}, x_N\}$ where $x \in [x_{N-2}, x_N]$.

Even degree polynomial (example degree = 2)



The 2nd degree interpolation is performed between the nodes $\{x_{i-1}, x_i, x_{i+1}\}$ and $\{x_i, x_{i+1}, x_{i+2}\}$ where $x \in [x_i, x_{i+1}]$. The first set gives the left interpolation polynomial while the second set gives the right interpolation polynomial. In the range $x \in [x_i, x_{i+1}]$ the final interpolation is obtained taking the average of the two polynomials. At the first of the segment, the interpolation is performed between the nodes $\{x_1, x_2, x_3\}$ where $x \in [x_1, x_2]$. At the end, the interpolation is performed between the nodes $\{x_{N-2}, x_{N-1}, x_N\}$ where $x \in [x_{N-1}, x_N]$.

The average interpolation schemas adopted for even degree polynomials can often increase the global accuracy but, on the other hand, it reduces the efficiency. This example helps to explain this trick.

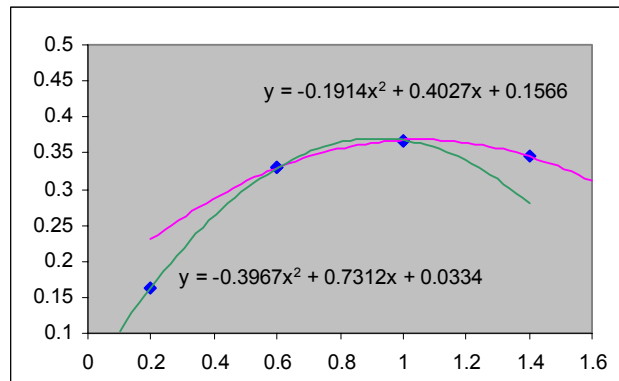
Assume to have the data set composed by 4 points; we want to perform the parabolic interpolation of a point $y(x)$ where $0.6 \leq x \leq 1$ and $y(0.6) = y_2$ and $y(1) = y_3$

i	x	y
1	0.2	0.163746
2	0.6	0.329287
3	1	0.367879
4	1.4	0.345236

Note that the given points belong to the function $y(x) = x e^{-x}$

Being the degree = 2, we can build two interpolation polynomials using the first 3 points and the last 3 points.

We have thus the left interpolation polynomial $P_L(x)$, covering the range $[0.2, 1]$, and the right interpolation polynomial $P_R(x)$ covering the range $[0.6, 1.4]$. The intersection range is $[0.6, 1]$ that is just the range to interpolate



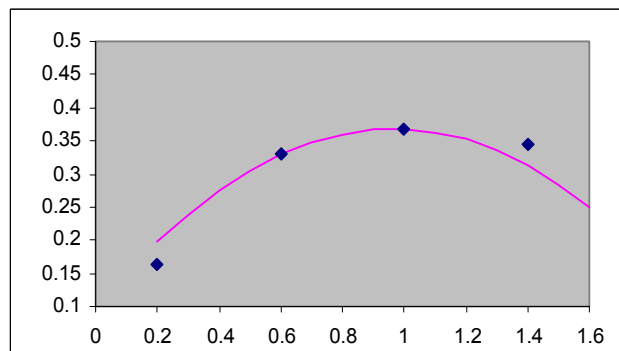
If we take the average

$$P_M(x) = [P_L(x) + P_R(x)] / 2$$

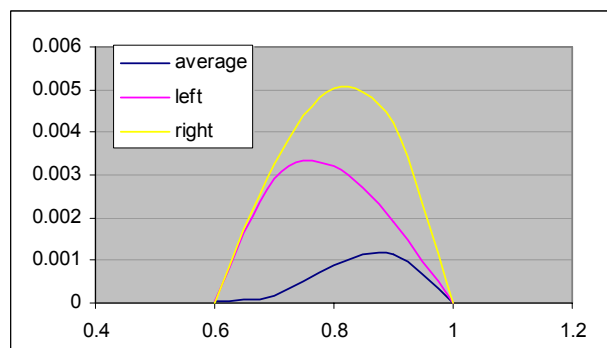
We obtain a new 2nd degree polynomial interpolating the points between $[0.6, 1]$

Observe that it satisfies the nodes constraints:

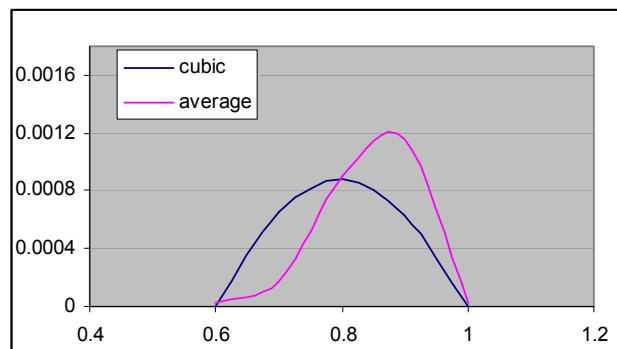
$$P_M(0.6) = y_2, \quad P_M(1) = y_3$$



If we compute the absolute errors $|y(x) - P_R(x)|$, $|y(x) - P_L(x)|$, $|y(x) - P_M(x)|$ in the range $0.6 \leq x \leq 1$, we observe a significant error reduction for the average polynomial, as shown in the following graph.



In this case the average-parabolic polynomial accuracy is comparable with the cubic polynomial.



But we have to consider that the simple cubic interpolation schema is more efficient than the average-parabolic. Generally odd degree interpolation schemas are preferable.

Interpolation with continued fraction

Fract_Interp_Coeff(xi, yi, [DgtMax])

Fract_Interp(x, xi, coeff, [DgtMax])

These functions perform the interpolation with the continued 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_Coeff** returns the coefficients vector $[a_0, a_1, a_2, a_3, a_4]$ of the continued 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

The optional parameter **DgtMax** sets the maximum digits of the multiprecision arithmetic. If omitted or 0, the functions works in faster double precision.

Example: find the continued fraction interpolation for the following (x, y) samples

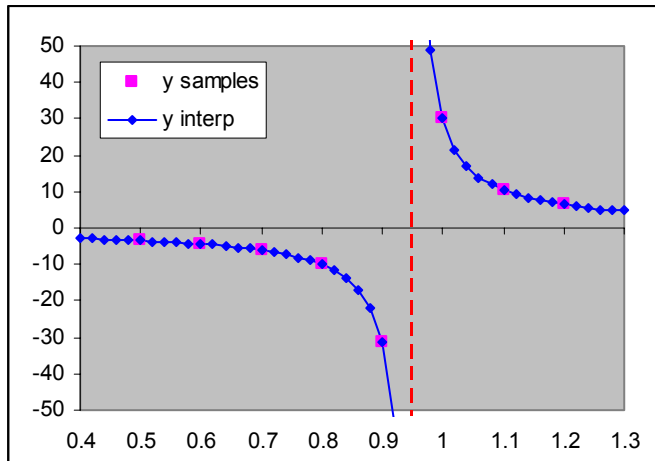
	A	B	C	D	E	F	G	H	I
1						h=	0.015	errmax=>	8.4E-15
2	n	xi	y samples	coefficients		x	y true	y interp	error
3	0	0.5	-3.461538462	-3.461538462		0.4	-2.9189189	-2.9189189	1.1E-15
4	1	0.6	-4.37037037	-0.110031348		0.415	-2.984748	-2.984748	7.4E-16
5	2	0.7	-6.073170732	2.989457243		0.43	-3.0553769	-3.0553769	5.8E-16
6	3	0.8	-10.15384615	1.284514107		0.445	-3.1312013	-3.1312013	2.8E-16
7	4	0.9	-31.22222222	1.472081218		0.46	-3.2126671	-3.2126671	4.1E-16
8	5	1	30	-6.70179E+11		0.475	-3.300278	-3.300278	1.3E-16
9	6	1.1	10.35483871	-4.0173E-13		0.49	-3.3946052	-3.3946052	1.3E-16
10	7	1.2	6.37037037	-3.33594E+12		0.505	-3.4962983	-3.4962983	1.3E-16
11	8	1.3	4.670886076	1.02373E+13		0.52	-3.6060991	-3.6060991	1.2E-16
12	9	1.4	3.735849057	0		0.535	-3.7248585	-3.7248585	1.2E-16
13						0.55	-3.8535565	-3.8535565	2.3E-16
14	{=FractInterpCoef(B3:B12,C3:C12)}					0.565	-3.9933279	-3.9933279	0
15						0.58	-4.1454933	-4.1454933	2.1E-16
16						{=FractInterp(F3:F63,B3:B12,D3:D12)}			
17						0.61	-4.4934647	-4.4934647	2E-16

These points are extracted from the following function

$$y = \frac{x^2 + 2}{x^2 - 0.9}$$

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 ($x \cong 0.95$) in the interpolation range. The continued fraction interpolation is just suitable for interpolating rational functions. The following graph shows very well how interpolate points fits the curve.



Interpolation with continued fraction

The pink dots are the given knots.
The blue line is obtained by interpolation

There is a pole at $x \approx 0.95$
(red dotted line)

Note that the points outside the range $[0.5, 1.2]$ keep also a good accuracy.
(extrapolation)

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 $x \approx 1.57$, near to the upper bound 1.5; so its presence suggests to adopt a fraction interpolation.

	A	B	C
1	x	tan(x)	interp coef
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			
10	{=Fract_Interp_Coef(A2:A8;B2:B8)}		

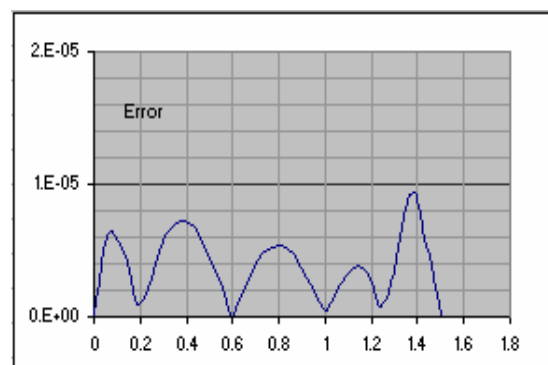
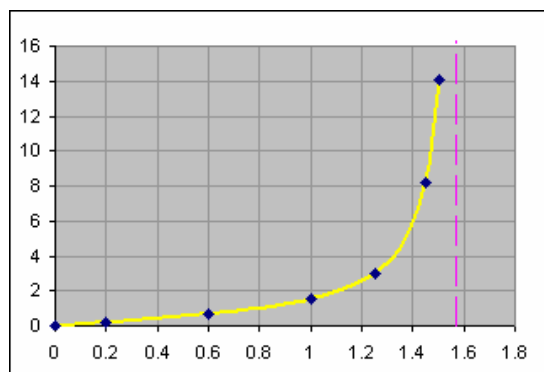
Assume to sample the function $\tan(x)$ at the values (0, 0.2, 0.6, 1, 1.25, 1.45, 1.5).

The column A contains the knots of the interpolations

In column B we have inserted the correspondent values of $\tan(x)$

In column C we have computed the coefficients of the fraction interpolation.

Using the function FracInterp 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 in the given range.



Interpolation with Cubic Spline

cspline_interp(Xin , Yin , Xtarget)

cspline_eval(Xin, Yin, Ypp, Xtarget)

These functions¹ 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 2nd 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 constraint).

The natural cubic spline has a continuous second derivative (acceleration). This characteristic is very important in many applied sciences (Numeric Control, Automation, etc...) when it is necessary to reduce vibration and noise in electromechanical motions. The drawback is that cubic spline is less efficient than other interpolation methods.

The function cspline_eval is faster than cspline_interp, because it 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 function)

Example:

	A	B	C	D	E
1	Original Data		Interpolated Data		
2	Xi	Yi	X	Y Interp	
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					

Cubic Spline 2nd derivatives

cspline_pre(Xin , Yin)

This function Returns the cubic spline 2nd 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 2nd derivative values. The first and the last values are zero (natural spline constraint).

The 2nd derivatives depend only by the given set of knots. So they can be evaluate only once for the whole range of the interpolation. By cspline_eval we can compute faster interpolation.

¹ These functions appear thanks to the courtesy of [Olgierd Zieba](#)

Example. Perform the sub-tabulation with $\Delta x = 0.1$ of the following table

	A	B	C	D	E	F
1						
2					sub-tabulation	
3						
4	x	knots	y''		x	y interp
5	-2	0.02999	0		-2	0.02999
6	-1.5	0.00003	0.11805		-1.9	0.02305
7	-1	0.08522	2.29168		-1.8	0.01635
8	-0.6	0.46400	1.18838		-1.7	0.01012
9	-0.3	0.83296	-2.94331		-1.6	0.00460
10	0.2	0.92262	-3.90098		-1.5	0.00003
11	0.4	0.71970	-1.15153		-1.4	-0.00269
12	0.8	0.23561	2.47036		-1.3	0.00013
13	1.2	0.01724	1.23449		-1.2	0.01282
14	1.6	0.00000	0.13416		-1.1	0.03974
15	2	0.02999	0		-1	0.08522
16	{=cspline_pre(A3:A13;B3:B13)}				-0.9	0.15244
					-0.8	0.23981
					-0.7	0.34459

The given table is in the range A3:B13

In the adjacent column C we have computed the 2nd 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.

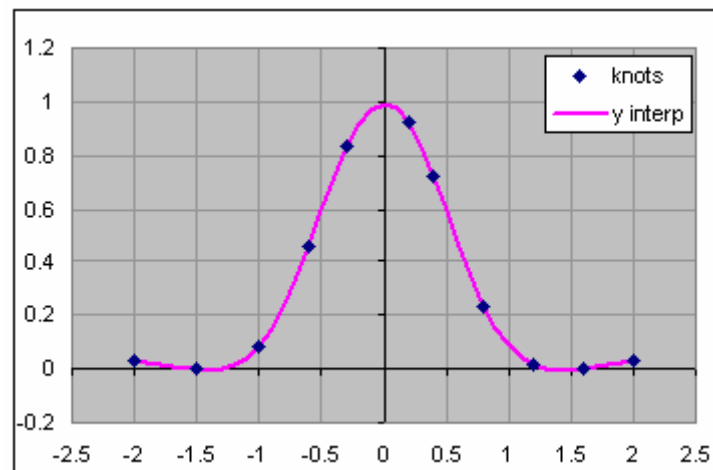
At the right we have set the new table with step 0.1.

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 plot shows the knots and the cubic spline fitting



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.

Cubic Spline Coefficients

cspline_coeff(Xin , Yin)

This function¹ 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	Knots			Cubic Spline coefficients			
2	X	Y		a3	a2	a1	a0
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							

¹ These functions appear thanks to the courtesy of Olgierd Zieba

2D Mesh 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 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							

{=Interp_Mesh(A1:G6)}

Regularization

As we can see, the use of this function is straight. Simply select the area where you want to insert the new table, insert the function Interp_mash and pass the old table as parameter.

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	

{=Interp_Mesh(A1:E6)}

Rescaling

We can obtain a sub-tabulated function in an 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

Macro Mesh Fill

This macro, accessible from the menu Macro / Regression, is useful for completing a 2D mesh by interpolation. Practically, given a (n x m) matrix with holes, it tries to recover the missing data. We could say a special case of bi-dimensional linear regression. Using this macro is very simple. Select the matrix with holes and start the macro.

Example

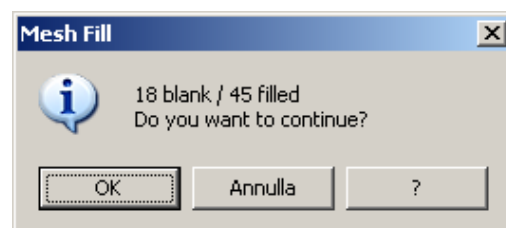
	A	B	C	D	E	F	G	H	I
1									
2		0.7	0.1	-0.5	-1.1	-1.7	-2.3	-2.9	
3		1.7	1.1	0.5	-0.1	-0.7	-1.3	-1.9	
4		2.7	2.1	1.5	0.9	0.3	-0.3	-0.9	
5		3.7	3.1	2.5	1.9	1.3	0.7	0.1	
6		4.7	4.1	3.5	2.9	2.3	1.7	1.1	
7		5.7	5.1	4.5	3.9	3.3	2.7	2.1	
8		6.7	6.1	5.5	4.9	4.3	3.7	3.1	
9		7.7	7.1	6.5	5.9	5.3	4.7	4.1	
10		8.7	8.1	7.5	6.9	6.3	5.7	5.1	
11									

Original data matrix complete filled.
It was generated by the following linear formula

$$a_{ij} = 2i - 3j$$

	A	B	C	D	E	F	G	H	I
12									
13		0.7	0.1	-0.5	-1.1	-1.7	-2.3		
14		1.7			-0.1	-0.7	-1.3		
15		2.7			0.9				
16		3.7	3.1	2.5	1.9		0.7	0.1	
17		4.7	4.1	3.5	2.9		1.7	1.1	
18		5.7	5.1	4.5		3.3	2.7	2.1	
19		6.7	6.1	5.5	4.9	4.3		3.1	
20		7.7			5.9		4.7	4.1	
21		8.7	8.1			6.3	5.7	5.1	
22									

the same data matrix with same missing values (holes). Select the matrix (range B13:H21) and start the macro



	A	B	C	D	E	F	G	H	I
12									
13		0.7	0.1	-0.5	-1.1	-1.7	-2.3	-2.9	
14		1.7	1.1	0.5	-0.1	-0.7	-1.3	-1.9	
15		2.7	2.1	1.5	0.9	0.3	-0.3	-0.9	
16		3.7	3.1	2.5	1.9	1.3	0.7	0.1	
17		4.7	4.1	3.5	2.9	2.3	1.7	1.1	
18		5.7	5.1	4.5	3.9	3.3	2.7	2.1	
19		6.7	6.1	5.5	4.9	4.3	3.7	3.1	
20		7.7	7.1	6.5	5.9	5.3	4.7	4.1	
21		8.7	8.1	7.5	6.9	6.3	5.7	5.1	
22									

The macro will try to rebuild the missing value (red) by linear interpolation, using the data of the nearest nodes. Of course, this process gives exactly the original matrix only in the linear case. For all other case we have an error: Generally, it is small if also the ratio between holes and filled nodes is small and if the original data is smooth.

Interpolation of tabulated data function

Given a tabulated data (x_i, y_i) , $i = 1 \dots N$, generally not equidistant, we have to interpolate the function 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 1st derivative, and continuous in the 2nd 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.

$$\begin{aligned} P_i(x_{i-1}) &= y_{i-1} & \text{for } i = 2 \dots N \\ P_i(x_i) &= y_i & \text{for } i = 2 \dots N \\ P'_i(x_i) &= P'_{i+1}(x_i) & \text{for } i = 2 \dots N-1 \\ P''_i(x_i) &= P''_{i+1}(x_i) & \text{for } i = 2 \dots N-1 \end{aligned}$$

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 2nd 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:

$$H_i = 6[(P_{i+1} - P_i)/(x_{i+1} - x_i) - (P_i - P_{i-1})/(x_i - x_{i-1})], \quad P''_1 = 0, \quad P''_N = 0$$

That gives the following tridiagonal matrix system

$2(x_3 - x_1)$	$(x_3 - x_2)$	0	0	...	0	P''_2	H_2
$(x_3 - x_2)$	$2(x_4 - x_2)$	$(x_4 - x_3)$	0	...	0	P''_3	H_3
0	$(x_4 - x_3)$	$2(x_5 - x_3)$	$(x_5 - x_4)$...	0	P''_4	H_4
0	0	$(x_5 - x_4)$	$2(x_6 - x_4)$	P''_5	H_5
...	$(x_N - x_{N-1})$...	
0	0	0	...	$(x_N - x_{N-1})$	$2(x_N - x_{N-2})$	P''_{N-1}	H_{N-1}

Another common way to write the interpolation polynomial is:

$$P(x) = a_{3i} (x - x_i)^3 + a_{2i} (x - x_i)^2 + a_{1i} (x - x_i) + a_{0i}, \quad x_i \leq x < x_{i+1}$$

for $i = 1 \dots (N-1)$

Where the coefficients are:

$$\begin{aligned} a_{3i} &= (P''_{i+1} - P''_i) / (x_{i+1} - x_i) / 6 \\ a_{2i} &= P''_i / 2 \\ a_{1i} &= (P_{i+1} - P_i) / (x_{i+1} - x_i) - (x_{i+1} - x_i) (2 P''_i + P''_{i+1}) / 6 \\ a_{0i} &= P_i \end{aligned}$$

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 2nd derivatives can be evaluated only once for any interpolate.

This example shows how the interpolation spline works.

X	Y
0	0
1	2
2.5	4
3	3
4	4
5	1

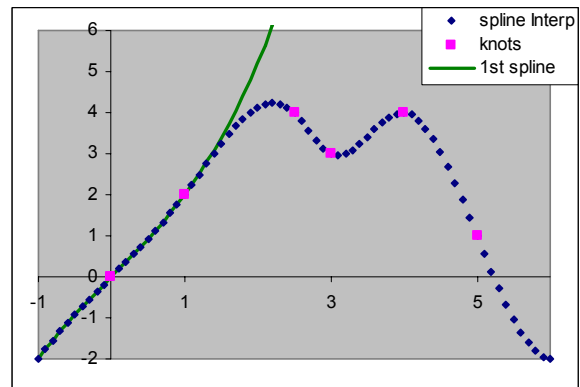
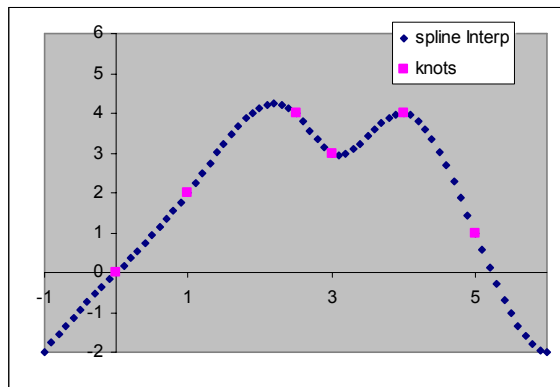
Assuming to have to sub-tabulate with a step $\Delta x = 0.1$ a given function known only in the following 6 points
Note that these points are unequal spaced

For these 6 knots we obtain 5 cubic polynomials having the following coefficients

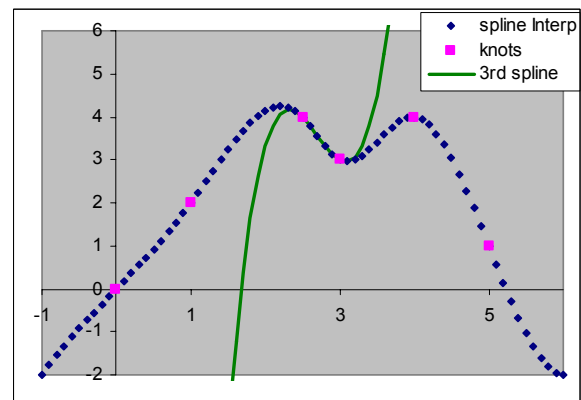
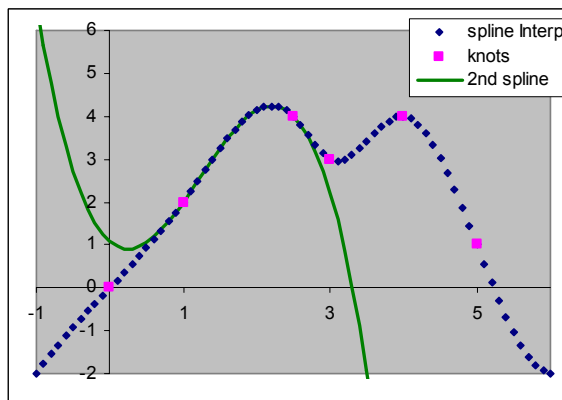
Polynomials	a3	a2	a1	a0	Range
1st spline	0.20148927	0	1.79851073	0	$0 \leq x < 1$
2nd spline	-0.8783764	0.60446781	2.40297854	2	$1 \leq x < 2.5$
3rd spline	5.54708717	-3.348226	-1.7126588	4	$2.5 \leq x < 3$
4th spline	-3.0718353	4.97240473	-0.9005694	3	$3 \leq x < 4$
5th spline	1.41436706	-4.2431012	-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.

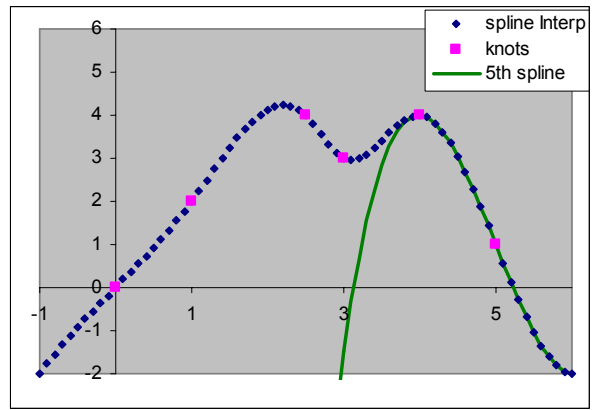
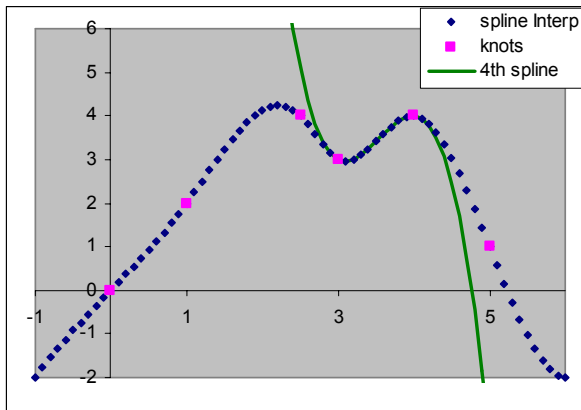
In the graphs below are shown the entire interpolation line (left) and the 1st spline (right).



In the graphs below are shown the 2nd spline (left) and the 3rd spline (right)



In the graphs below are shown the 4th spline (left) and the 5th spline (right)

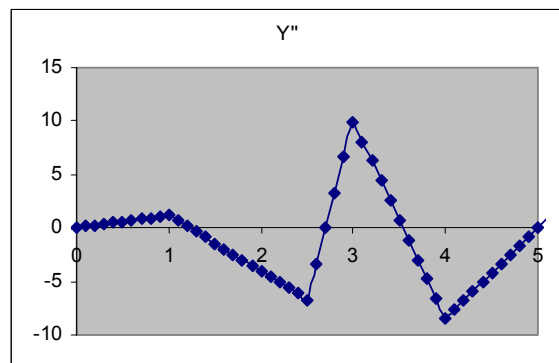
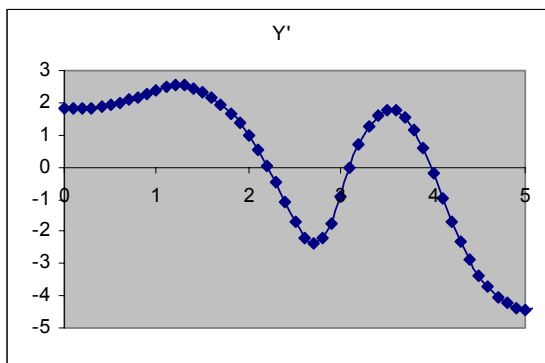


Let's examine the 1st and 2nd derivatives. We can compute them either analytically or numerically using – for example – the following derivative formulas:

$$y'(x_i) \cong (y_{i+1} - y_{i-1}) / 2\Delta x$$

$$y''(x_i) \cong (y_{i+1} - 2y_i + y_{i-1}) / \Delta x^2$$

In both ways, we get the following graphs



As we can see the 1st derivatives is smooth and the 2nd is continuous. This last feature is particularly appreciated in many fields of engineering. Although this algorithm is less efficient than other polynomial interpolation methods, it has the advantage of following the interpolated curve without the spurious oscillations that other schemes can create

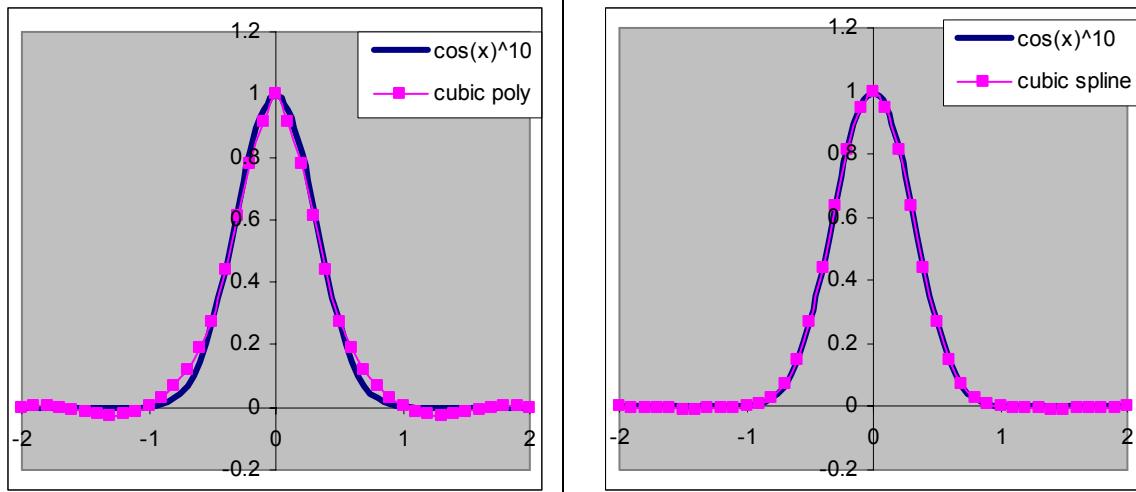
Other test functions

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

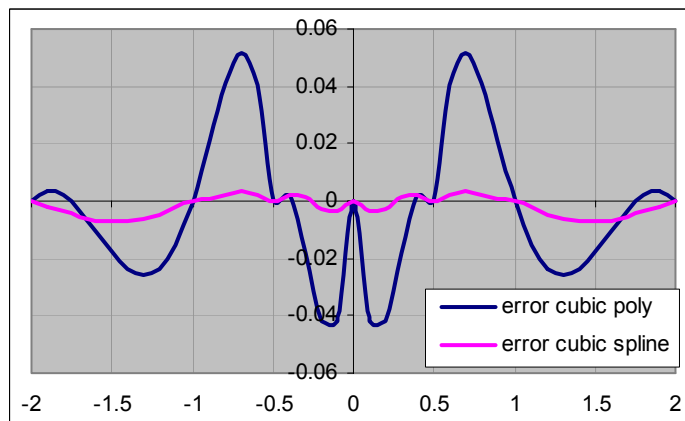
$$Y = (\cos(x))^{10}, \text{ for } x = -2, -1, -0.5, 0, 0.5, 1, 2$$

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

Continued fraction interpolation

Continued fractions are often a powerful ways of interpolation when we work near the functions poles.

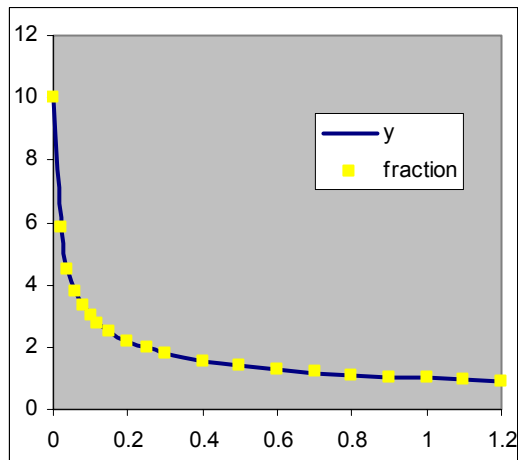
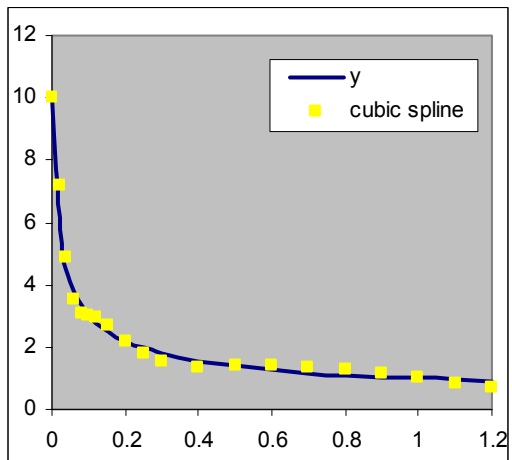
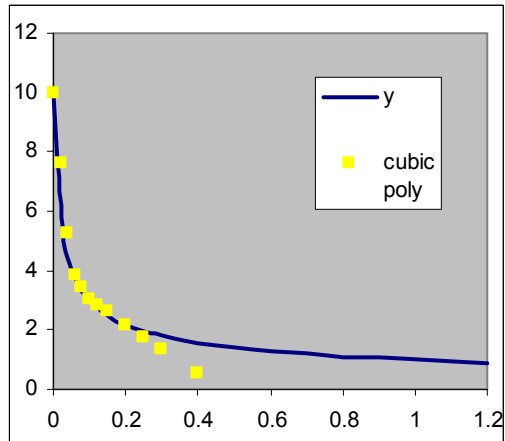
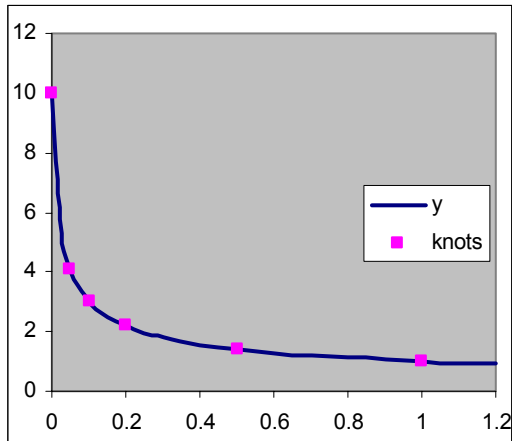
In XNUMBERS the continued fraction coefficients can be obtained by the function **FracInterpCoef** and the interpolation value with **FracInterp**

Example. Interpolate the following dataset

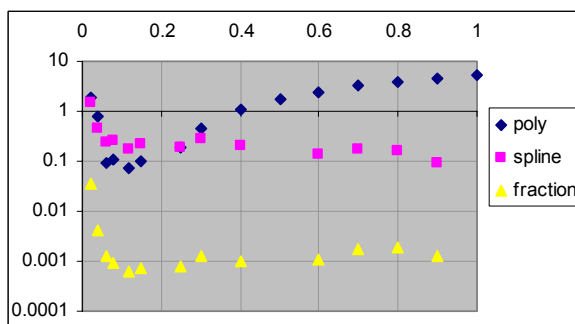
$$y(x) = 1/(x+0.01)^{1/2}, \text{ for } x = 0, 0.05, 0.1, 0.2, 0.5, 1, \text{ with step } \Delta x = 0.1$$

In the graphs below we have plotted the interpolated values obtained with three different methods: cubic polynomial, cubic spline and continued 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 $x > 1$)



Method	avg. err.
Cubic spline	0.22
Cubic poly	0.19
Contined 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 1st order ordinary differential system.

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y}) \quad , \quad \mathbf{y}(t_0) = \mathbf{y}_0 \quad \Leftrightarrow \quad \begin{cases} y'_1 = f_1(t, y_1, y_2 \dots y_n) \\ y'_2 = f_2(t, y_1, y_2 \dots y_n) \\ \dots \\ y'_n = f_n(t, y_1, y_2 \dots y_n) \end{cases} , \quad \begin{pmatrix} y_1(t_0) = y_{10} \\ y_2(t_0) = y_{20} \\ \dots \\ y_n(t_0) = y_{n0} \end{pmatrix}$$

ODE Runge-Kutta 4

= ODE_RK4(Equations, Varlnit, Step, [Par, ...])

This function integrates numerically a 1st order ordinary differential equation or a 1st order differential system, with the Runge-Kutta formula of 4th 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 "Varlnit" is a vector containing the initial values. It has two values for two variables [t0, y0].

For a system with n+1 variables, "Varinit" is an (n+1) vector [t0, 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 $y = e^{-x^2}$

We can arrange a worksheet like the following

	A	B	C	D	E	F	G
1							
2	starting values					step	differential equation
3							
4	x	y				h	diff. equation
5	0	1				0.1	$y' = -2*x*y$
6	0.1	0.99005					$\{=ODE_RK4(\$G\$5,A5:B5,$F\$5)\}$
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)

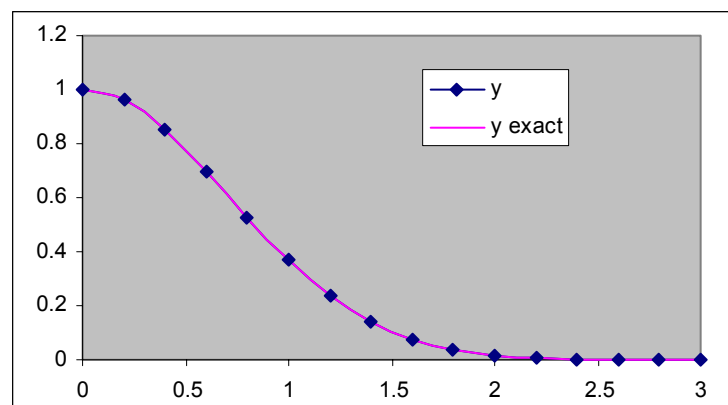
	x	y	y (exact)	error
5	0	1	1	0
6	0.1	0.99005	0.9900498	4.158E-10
7	0.2	0.960789	0.9607894	3.917E-09
8	0.3	0.913931	0.9139312	1.125E-08
9	0.4	0.852144	0.8521438	1.65E-08
10	0.5	0.778801	0.7788008	2.528E-09
11	0.6	0.697676	0.6976763	6.111E-08
12	0.7	0.612627	0.6126264	2.158E-07
13	0.8	0.527293	0.5272924	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

$\text{=ODE_RK4}(\$G\$5,A5:B5,$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 approximate solution (dots) fits very good the exact solution (pink curve).

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	diff. equation		h	k	n
2	$y' = -k \cdot x^n \cdot y$		0.1	2	1
3	=ODE_RK4(\$A\$2,A6:B6,\$C\$2,\$D\$2,\$E\$2)				
4					
5	x	y			
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

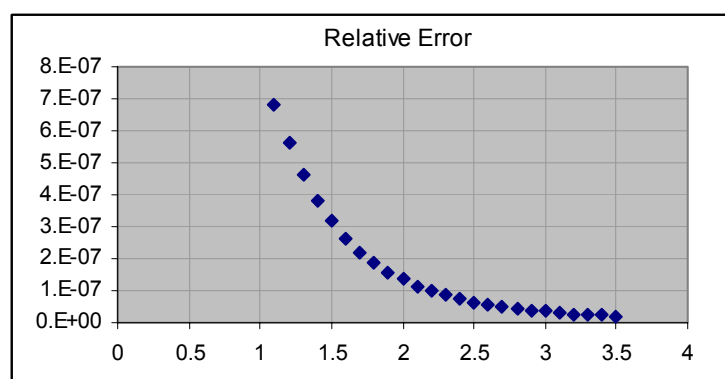
$$y' = a \cdot x^n - \frac{y}{x}, \quad y(1) = 0$$

	A	B	C	D	E
1	diff. equation		h	a	n
2	$y' = a \cdot x^n - y/x$		0.1	1	3
3	=ODE_RK4(\$A\$2,A6:B6,\$C\$2,\$D\$2,\$E\$2)				
4					
5	x	y			
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 $y = (x^5 - 1)/(5x)$, (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	

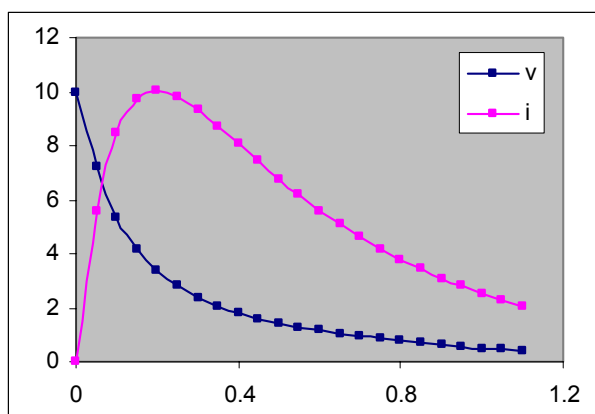
The computation can be arranged as following.

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 shows 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 st order
$e \approx -\frac{1}{12} h^3 y^{(3)}$	Error corrector 2 nd 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 th order
$e \approx -\frac{19}{720} h^5 y^{(5)}$	Error corrector 4 th 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 9th order and relative errors

Multi-step Predictor coefficients

Ord. \Rightarrow	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

Ord. \Rightarrow	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 $e \approx -k \cdot h^{\text{Ord}+1} y^{(\text{Ord}+1)}$ where k is given by the following table

Ord. \Rightarrow	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 2nd 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 require implicit formulas that, generally, can be solved only by iterative algorithms.

The 2nd order trapezoid formula is one of the most popular A-stable formula.

Predictor- Corrector

= ODE_PRE(y_n, f, h)

= ODE_COR(y_n, 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 "y_n" 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 2nd order

Now we see how arrange a PECE algorithm of 2nd 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 2nd 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.

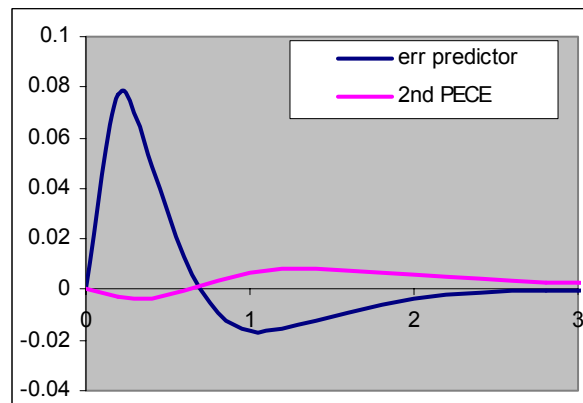
$$y = \frac{2}{1+x^2}$$

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 4th order

Now we solve the above differential equation with a 4th order PECE algorithm using the 4 steps Adams-Bashforth-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 preserve 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	Predictor-Corrector (PECE) of 4^o order				
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 the last four values of f(x,y), E6:E9 in this case.

"h" is the step B3.

	A	B	C	D	E
1	Predictor-Corrector (PECE) of 4^o order				
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	=E_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.

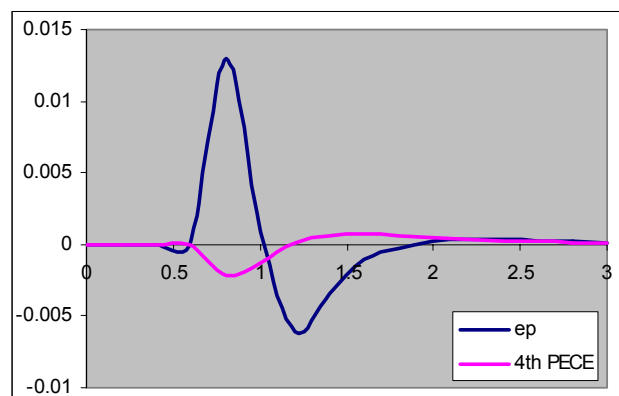
"f" is a vector of the last 3 values of f(x,y), E7:E9.

"fp" is the predicted value of f(x,y), C10 in this case.

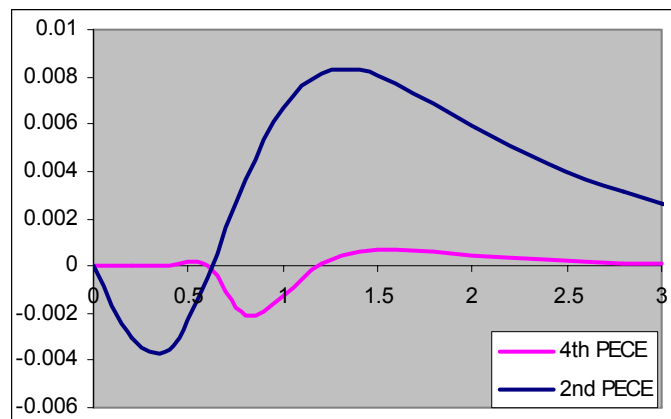
"h" is the step B3

Now the setting of the PECE algorithm of 4th order is completed. Select the 5th row and drag it down in order to calculate the steps you want.

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 2nd order algorithm of the previous example let's draw both the error curves in a same graph



As we can see, the 4th order algorithm is evidently more accurate than the 2nd order. On the other hand, the first one requires an extra work for providing 3 starting points.

ODE Predictor-Corrector 4

= ODE_PC4(Equations, Varlnit, Step, [Par, ...])

This function integrates numerically an ordinary differential equation of 1st order or differential system of 1st order, with the 4th order P(EC)^2 schema predictor-corrector of Adams-Bashforth-Moulton.

The input arguments are identical to the function ODE_RK4

The function can return all the solution points in an array. Before inserting the function, select as many rows that you want to fill. For example if you select a (p x n+1) array, the function will return p solution points of a n x n differential system.

Let's see how it works with an example

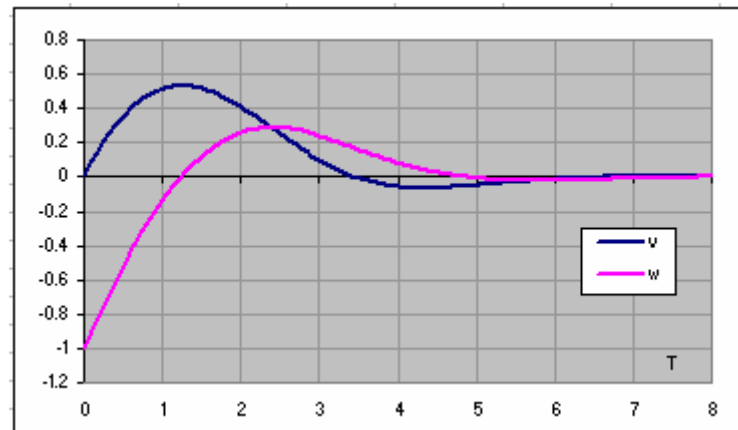
$$\begin{cases} v' = v(1-v)(v-1/2) - w \\ w' = v - w \end{cases} \quad \begin{cases} v(0) = 0 \\ w(0) = -1 \end{cases} \quad (\text{FitzHug-Naguno model})$$

Insert the equations in the cell A2 and A3. Insert the starting point [t0, v0, w0] = [0, 0, -1] in the range A7:C7

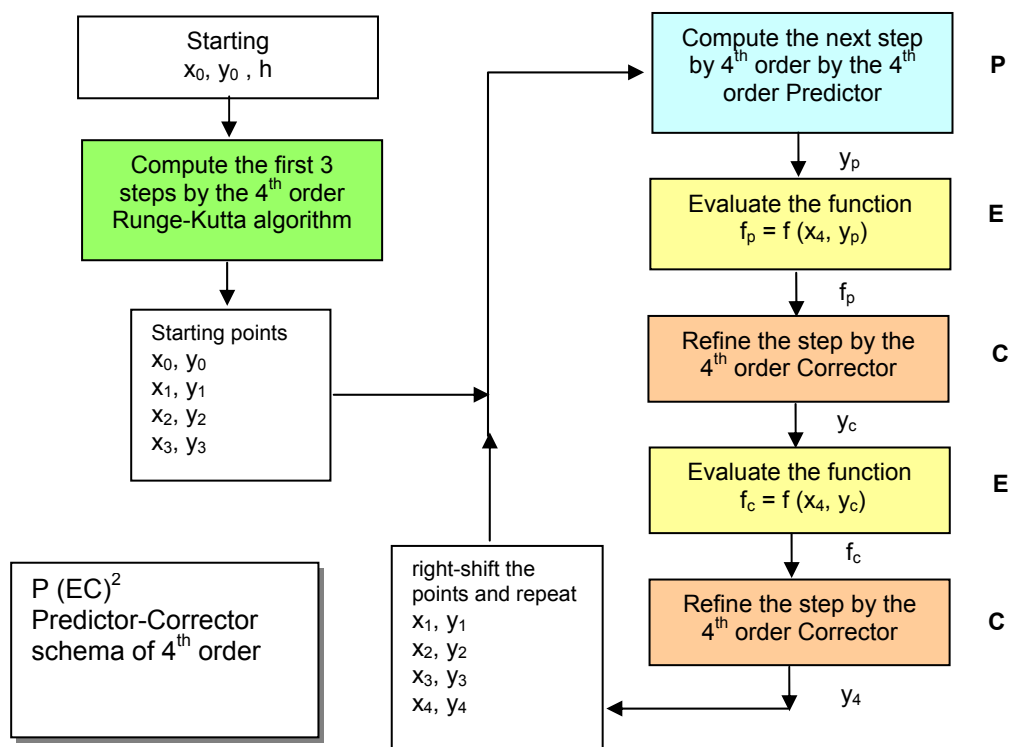
Select the range A8:C167 in order to get simultaneously 160 integration points and insert the function ODE_PC4 by the CTRL+SHIFT+ENTER sequence.

	A	B	C	D	E
1					
2	$v' = v*(1-v)*(v-1/2) - w$			h	
3	$w' = v - w$			0.05	
4					
5					
6	t	v	w		
7	0	0	-1		
8	0.05	0.048198	-0.95003		
9	0.1	0.093051	-0.90023		
10	0.15	0.134881	-0.85075		
11	0.2	0.17224	-0.80174		

The following plot shows the obtained functions $v(t)$ and $w(t)$ for $0 \leq t \leq 8$



The 4th order $P(EC)^2$ schema predictor-corrector of Adams-Bashforth-Moulton is detailed in the following flow-graph



The initial extra 3 steps, necessary for starting the predictor-corrector algorithm, are calculated by the function ODE_RK4 requiring 4 function evaluations for step. After that, the PC schema require only 2 function evaluations for step. Conventionally we measure the effort (EF) counting the number of function evaluation for n steps.

$$EF = 12 + 2 \cdot n \text{ (for PC schema), } EF = 4 \cdot n \text{ (for RK algorithm)}$$

For $n \gg 1$ the effort reduction of the $P(EC)^2$ schema is about 50%

ODE Implicit Predictor-Corrector

ODE_PC2I(Equations, VarInit, Step, [Par, ...])

This function integrates numerically an ordinary differential equation of 1st order or a differential system of 1st order, with the 2th order implicit Predictor-Corrector method. Thanks to its large stability it is suitable for solving "stiff" problems.

The input arguments are identical to the function ODE_RK4

The function can return all the solution points in an array. Before inserting the function, select as many rows that you want to fill. For example if you select a (p x n+1) array, the function will return p solution points of a n x n differential system.

This function uses the 1st order Euler formula as predictor and the 2nd order trapezoidal formula as the corrector.

$y_{n+1} = y_n + h \cdot f(t_n, y_n)$	Predictor (Euler formula)
$y_{n+1} = y_n + h / 2 \cdot (f(t_n, y_n) + f(t_{n+1}, y_{n+1}))$	Corrector (Trapezoidal formula)

The second non-linear equation is started with the predicted value and then solved respect to the variable y_{n+1} till the convergence.

This algorithm, despite its low order, exhibits a large stability and, thus, it is suitable for solving "stiff" problems

Let's see how it works with an example

Solve numerically the following 2nd order Cauchy's problem for $1 \leq x \leq 7$ and $k = 100$

$$\frac{d^2 y}{dx^2} + (k+1) \frac{dy}{dx} + k y = 0 \quad , \quad y(1) = 1 \quad , \quad y'(1) = -0.5$$

First of all, we transform the problem in a 1st order differential equations system taking

$$y_1 = y \quad , \quad y_2 = y'$$

$$\begin{cases} y_1' = y_2 \\ y_2' = -k y_1 - (k+1) y_2 \end{cases} \quad \begin{cases} y_1(1) = 1 \\ y_2(1) = -0.5 \end{cases}$$

A possible arrangement may be the following

	A	B	C	D	E
1	Stiff Differential System				
2	$y_1' = y_2$			k	h
3	$y_2' = -k \cdot y_1 - (k+1) \cdot y_2$			100	0.2
4					
5			[=ODE_PC2I(A2:A3;A8:C8;E3;D3)]		
6					
7	x	y1	y2		
8	1	1	-0.5		
9	1.2	0.82645	-1.235537		
10	1.4	0.66942	-0.334711		
11	1.6	0.55324	-0.827095		
12	1.8	0.44813	-0.224063		
13	2	0.37035	-0.553675		
14	2.2	0.29998	-0.149992		
15	2.4	0.24792	-0.370642		
16	2.6	0.20082	-0.100408		
17	2.8	0.16596	-0.248116		
18	3	0.13443	-0.067215		
19	3.2	0.1111	-0.166094		

We have written in cell A2 and A3 the differential equations

In cell D3 we have inserted the k parameter and in the cell E3 the h step

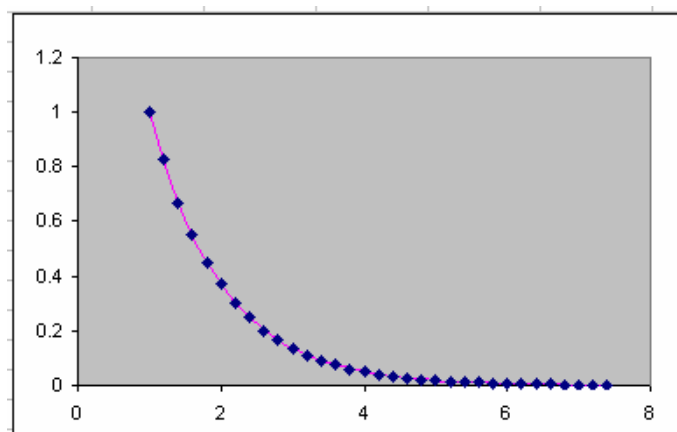
In the range A8:C8 we have put the starting values for x and y1 and y2.

note that we have also added the labels "k", "x", "y1", "y2" just above their values.

Labels are necessary for the correct assignment in the symbolic equation.

Now select the range A9:C40 and insert the function ODE_PC2I passing its arguments

The scatter plot of the 30 points (x, y1) is shown in the following graph



Compare the approximate solution (dotted line) with the exact one (pink continue line). The fitting, even with only 30 points, looks good. If we try to solve this problem with the 4th order Runge-Kutta algorithm we have to choose a step less than 0.025 for avoiding the instability and, thus, we need more than 250 points for reaching the same integration interval.

The Lotcka-Volterra Model

The following two-dimensional differential system

$$\begin{cases} \frac{dx}{dt} = a_1 x - a_2 x y \\ \frac{dy}{dt} = -a_3 y + a_4 x y \end{cases}$$

is called the Lotcka-Volterra model or also "prey-predator" model. It is very useful in biology, chemistry and many other fields. The numerical integration of this ODE family requires a stable algorithm otherwise the result are not acceptable

The function $x(t)$ and $y(t)$, depending from the time, may represent different things. In a biological models, for example, they simulate respectively the population of pray (rabbits) and predator (foxes) at time t . The proportionality constants a_1 , a_2 , a_3 , and a_4 are positive. It is known that its solution leads to oscillate steady-state.

. Let's see with a practical example

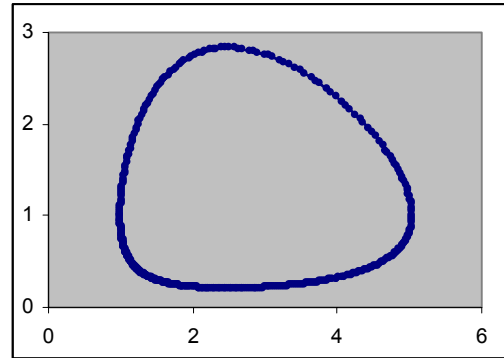
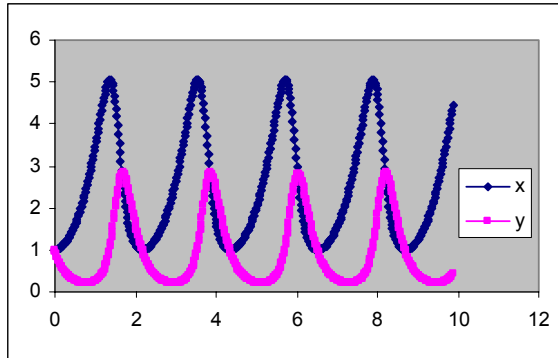
$$\begin{cases} x' = 2x - 2xy \\ y' = -5y + 2xy \end{cases} \quad \begin{cases} x(0) = 1 \\ y(0) = 1 \end{cases}$$

	A	B	C	D
1	$x' = 2*x - 2*x*y$		h	
2	$y' = -5*y + 2*x*y$		0.02	
3	$(=ODE_PC2I(A1:A2;A6:C6;C2))$			
4				
5	t	x	y	
6	0	1	1	
7	0.02	1.001166	0.941769	
8	0.04	1.004602	0.887009	
9	0.06	1.010194	0.835583	
10	0.08	1.017845	0.787347	

Insert the equation definition in the cells A1 and A2, the step in the cell C2 = 0.02 and the starting values [0, 1, 1] in the range A6:C6

Do not miss the labels "t", "x", "y". Then, select the range A7:C407 and insert the function ODE_PC2I with the ctrl-shift-enter keys sequence

The following graph at the left shows the result of the function $x(t)$ and $y(t)$. By performing the scatter plot of x-y variable we have the steady-state plot at the right. It shows a closed loop, a limit cycle.

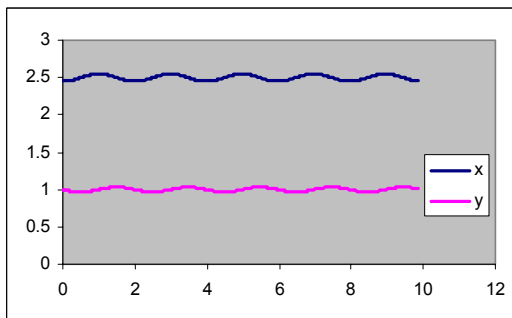


We can easily change the zoom of the graph by changing the integration step h . Note that the oscillations are quite deep. The above system has two equilibrium points where $dx/dt = 0$ and $dy/dt = 0$ simultaneously, that can be found solving the algebraic system

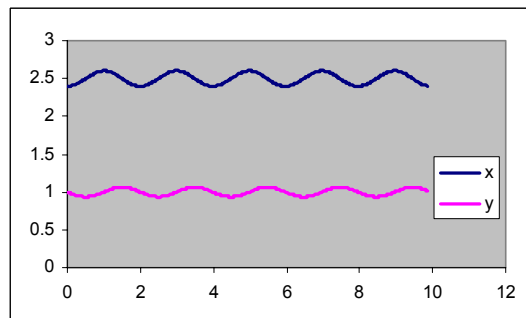
$$\begin{cases} 2x - 2xy = 0 \\ -5x + 2xy = 0 \end{cases} \Rightarrow \begin{cases} x = 0 \\ y = 0 \end{cases} \quad \begin{cases} x = 2.5 \\ y = 1 \end{cases}$$

The non trivial solutions means that if the initial populations start from $x_0 = 2.5$ and $y_0 = 1$, then there will be no oscillation and the population fraction x/y will be always constant in the time. On the contrary, the populations become progressively to oscillate when the initial populations are different from the equilibrium point $[2.5, 1]$. Here some examples:

$x(0) = 2.45$ and $y(0) = 1$



$x(0) = 2.4$ and $y(0) = 1$



Differential Systems

OD Linear System

= ODE_SYSL(A, y0, step, [b])

This function integrates numerically a system of ordinary linear differential equations of 1st order with constant coefficients, starting from an initial value. For example, the general form of a 3x3 linear differential system is:

$$\begin{cases} y_1' = a_{11}y_1 + a_{12}y_2 + a_{13}y_3 + b_1 \\ y_2' = a_{21}y_1 + a_{22}y_2 + a_{23}y_3 + b_2 \\ y_3' = a_{31}y_1 + a_{32}y_2 + a_{33}y_3 + b_3 \end{cases}$$

where all the coefficients a_{ij} and b_i are constant.
Such system can be put in the following handy matrix form.

$$\bar{y}' = [A] \cdot \bar{y} + \bar{b}$$

where

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad \bar{y} = [y_1, y_2, y_3]^T$$

$$\bar{b} = [b_1, b_2, b_3]^T$$

and with the initial condition:

$$\bar{y}_0 = [y_1^{(0)}, y_2^{(0)}, y_3^{(0)}]^T$$

The constant term b is optional. If omitted the system is called "homogeneous"

This function uses the exponential expansion method that, for this kind of differential systems, is both accurate and efficient.

The function returns an $(n \times m)$ array containing all the nodes of the integration: m is the number of equations; n is the numbers of the nodes. The function automatically sets n equal to the rows of the range that you have selected before inserting it.

Let's see how it works practically

Solve the following homogeneous differential system with constant coefficients

$$\bar{y}' = [A] \cdot \bar{y}$$

where

$$A = \begin{bmatrix} -20 & -10 & 19 \\ 16 & 6 & -16 \\ -2 & -2 & 1 \end{bmatrix} \quad \bar{y}_0 = [1, 0, 0]^T$$

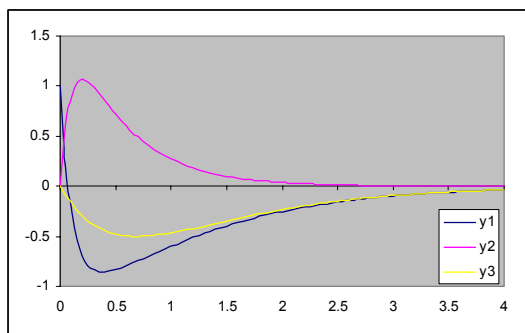
For $0 \leq x \leq 4$ and $h = 0.05$

The numerical solution can be arranged as in the following worksheet

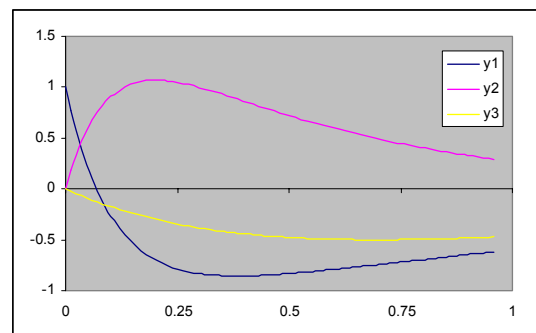
The initial values are in the first row (range B7:D7). The column "x" was added only for clarity but it is not indispensable at all. Select the range B8:D87, then insert the function ODE_SYSL giving the suitable parameters A, y_0 , h . Then press CTRL+SHIFT+ENTER

	A	B	C	D	E	F	G
1					-20	-10	19
2	h	0.05		A =	16	6	-16
3					-2	-2	1
4		{=ODE_SYSL(E1:G3;B7:D7;B2)}					
6	x	y1	y2	y3			
7	0	1	0	0			
8	0.05	0.21544	0.59661	-0.0928			
9	0.1	-0.2552	0.9017	-0.1722			
10	0.15	-0.5343	1.03538	-0.2398			
11	0.2	-0.6965	1.06997	-0.2968			
12	0.25	-0.7869	1.04889	-0.3445			
13	0.3	-0.8333	0.99805	-0.384			
14	0.35	-0.8524	0.93278	-0.4162			

All the 240 cells will be filled with the nodal solutions of y1, y2, y3.
The scale can be easily arranged simply by changing the parameter h



h = 0.05



h = 0.012

Compare with the exact solutions

$$\begin{cases} y_1 = -2e^{-x} + e^{-2x} + 2e^{-10x} \\ y_2 = 2e^{-2x} - 2e^{-10x} \\ y_3 = -2e^{-x} + 2e^{-2x} \end{cases}$$

High order linear ODE

The function ODE_SYSL can also be used to solve high order linear ODE with constant coefficients, that in general can be written as

$$y^{(n)} + a_{n-1}y^{(n-1)} + \dots + a_2y'' + a_1y' + a_0y = b$$

with the initial conditions

$$y(x_0) = y_0, y'(x_0) = y_0', y''(x_0) = y_0'', \dots, y^{(n-1)}(x_0) = y_0^{(n-1)}$$

As known, such ODE can be transformed into a linear differential system of 1st order.

$$\begin{bmatrix} y_1' \\ y_2' \\ \vdots \\ y_{n-1}' \\ y_n' \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \vdots & 0 \\ 0 & 0 & 1 & \vdots & 0 \\ \dots & \dots & \dots & \ddots & \dots \\ 0 & 0 & 0 & \vdots & 1 \\ -a_0 & -a_1 & -a_2 & \vdots & -a_{n-1} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ b \end{bmatrix}$$

having the initial conditions

$$\bar{y}_0 = [y_1(0), y_2(0), y_3(0), y_n(0)]^T = [y(0), y'(0), y''(0), y^{(n-1)}(0)]^T$$

Xnumbers Tutorial

For example assume that you have to calculate the solution of following IVP problem:

$$y''' + 5y'' + 9y' + 5y = 0 \quad , \text{ with } y(0) = 2, \quad y'(0) = -3, \quad y''(0) = 4$$

Introducing the auxiliary variables $y_1 = y$, $y_2 = y'$, $y_3 = y''$, we get the following equivalent differential system

$$\begin{bmatrix} y_1' \\ y_2' \\ y_3' \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -5 & -9 & -5 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \quad \text{with} \quad \begin{cases} y_1(0) = 2 \\ y_2(0) = -3 \\ y_3(0) = 4 \end{cases}$$

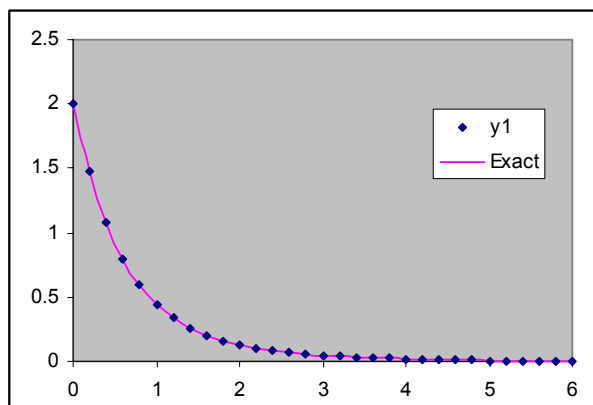
Observe that the last row of the matrix contains the coefficients of the given ODE with the opposite sign; besides of that, it has only all "1" in the upper subdiagonal.

Insert the initial values in the first row (range B10:D10). The column "x" was added only for clarity but it is not indispensable at all. Select the range B11:D40, where you want to output the results and insert the function ODE_SYSL giving the suitable parameters: A, y0, h.

Then press CTRL+SHIFT+ENTER

The selected area will be filled with the numerical solution of the given system

	A	B	C	D	E
1	A =	0	1	0	
2		0	0	1	
3		-5	-9	-5	
4					
5	h =	0.2			
6					
7		{=ODE_SYSL(B1:D3;B10:D10;B5)}			
8					
9	x	y1	y2	y3	
10	0	2	-3	4	
11	0.2	1.47569	-2.2658	3.32229	
12	0.4	1.08418	-1.673	2.61181	
13	0.6	0.7974	-1.2161	1.97484	
14	0.8	0.58999	-0.8755	1.45064	
15	1	0.444	-0.628	1.04277	



Observe that the solution $y_1(x)$ is also the solution $y(x)$ of the given ODE. Compare with the exact solution

$$y(x) = e^{-x} + e^{-2x} \cos x$$

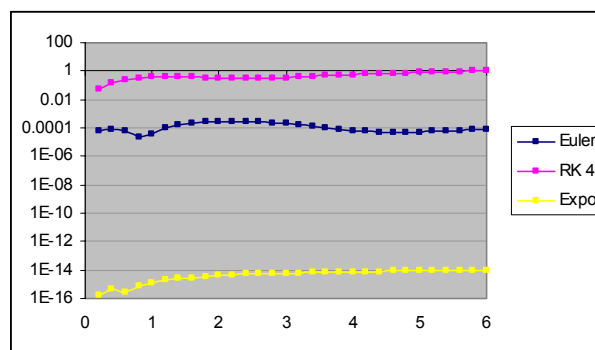
Of course the above differential system can be also solved with other methods.

In order to show the accuracy of the exponential method we put in a graph the average relative error

$$e(x) = |y_i - y(x_i)| / |y(x_i)|,$$

obtained in the same condition, with 3 different methods: Exponential, Euler, and Runge-Kutta 4.

The graph is eloquent. The error of the Exponential method is several times more accurate than the others



Clearly it takes advantages using dedicated methods for linear differential equations.

Another important feature of the exponential method is its high stability

Let's try the following test stiff system

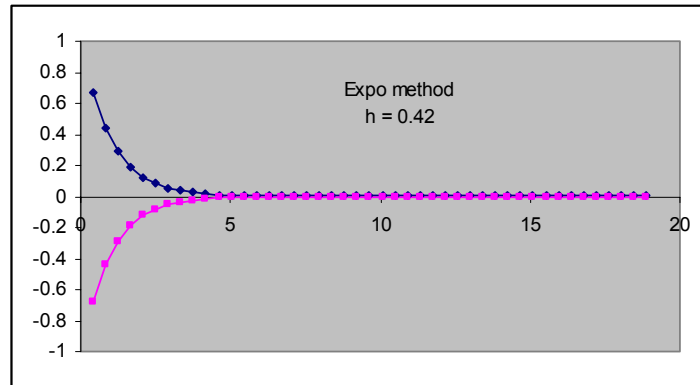
$$\begin{cases} y_1' = y_2 \\ y_2' = -20y_1 - 21y_2 \end{cases} \quad \begin{cases} y_1(0.4) = 0.6706555 \\ y_2(0.4) = -0.6770293 \end{cases}$$

The exact solution is

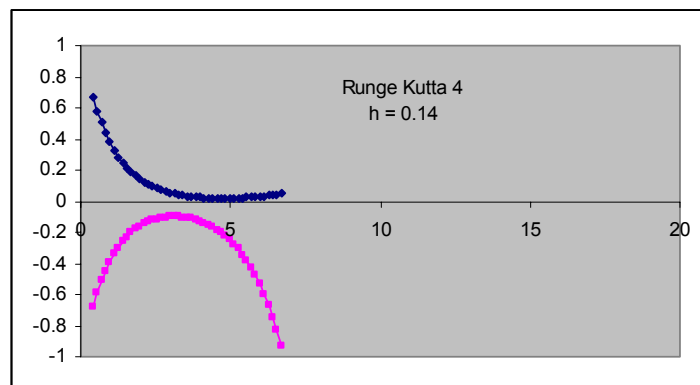
$$\begin{cases} y_1 = e^{-x} + e^{-20x} \\ y_2 = -e^{-x} - 20e^{-20x} \end{cases}$$

In the following graph we shows the numerical result obtained with three different integration methods, Exponential, Runge Kutta of 4th order and Euler, using different integration steps

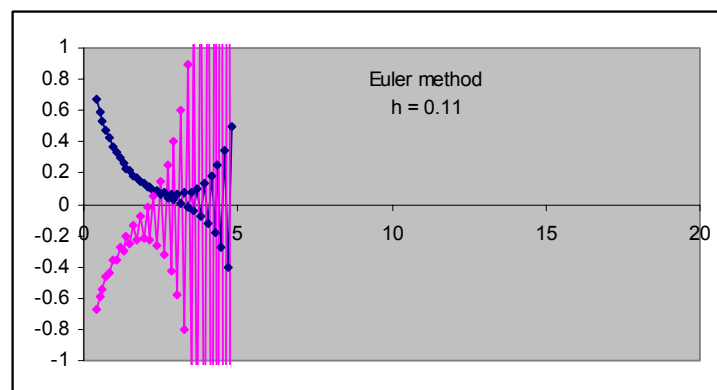
The Expo method reaches the integration length of about $x = 19$ with a step $h = 0.4$



The Runge-Kutta method, using a step $h = 0.14$ gives accurate only for $x < 3$; after that the solution begins to diverge.



The Euler method, with a step $h = 0.11$, begins to oscillate even from the first steps. At this step the Euler method is completely unstable.



As we can see the stability step of the Exponential method is here about 3 times greater than the others methods.

ODE for integral function solving

The integral function $y(x)$ that we want to calculate is defined as

$$y(x) = \int_{x_0}^x f(x) dx$$

\Leftrightarrow

Taking the derivatives of both sides and remembering Leibniz's rule, we have

$$y'(x) = f(x) \quad , \quad y(x_0) = 0$$

As we can see, the computing of any integral function is equivalent of solving an ODE.

Example. Plot the integral function of $f(x) = x e^{-x}$

	A	B	C
1	$y' = x * \exp(-x)$		
2	h	0.15	
3			
4	{=ODE_RK4(A1,A6:B6,B2)}		
5	x	y	y'
6	0	0	0
7	0.15	0.010186	0.1291062
8	0.3	0.036936	0.2222455
9	0.45	0.075439	0.2869327
10	0.6	0.121901	0.329287
11	0.75	0.173358	0.3542749

Insert in the cell A1 the expression

$$y' = x * \exp(-x)$$

Choose a suitable step, for example $h = 0.15$.

Select the range A7: B54 and insert the

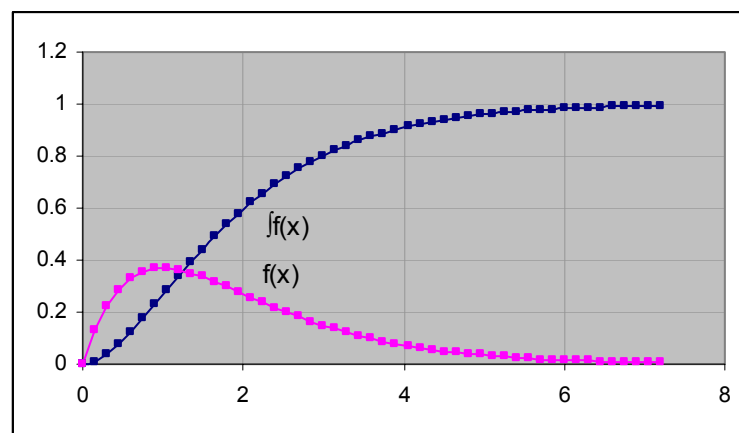
function ODE_RK4 with ctrl+shift+enter

In the adjacent column C we insert the

integration function $f(x) = x * \exp(-x)$

Now select all the range A7:C54 and plot together.

The result is shown in the following graph



Comparing with the exact integral function $y(x) = 1 - (x+1) e^{-x}$, we have obtained a good solution with an average error of about $5E-7$. For this example we have taken about 50 points.

Remark. This method works fine when the function $f(x)$ is smooth together with its derivative and has no singularity in the integration range. Otherwise we have to choose some quadrature algorithm as the double exponential method (see chap Integration for further details)

Example. Solve the following linear differential equation of 3rd order.

$$y''' + 5y'' + 33y' + 29y = 0 \quad \text{with} \quad y(0) = 2, \quad y'(0) = -3, \quad y''(0) = -20$$

This linear equation with constant terms can be transformed into the equivalent differential system:

$$\begin{bmatrix} y_1' \\ y_2' \\ y_3' \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -29 & -33 & -5 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \quad \text{with} \quad \begin{cases} y_1(0) = 2 \\ y_2(0) = -3 \\ y_3(0) = -20 \end{cases}$$

	A	B	C	D	E	F	G
1	$y''' + 5y'' + 33y' + 29y = 0$			$y(0)=2$	$y'(0) = -3$	$y''(0) = -20$	
2							
3		h =	0.05	$=\text{ODE_SYSL}(\text{B5:D7};\text{B10:D10};\text{B3})$			
4							
5	A =	0	1	0			
6		0	0	1			
7		-29	-33	-5			
8							
9	x	y1	y2	y3			
10	0	2	-3	-20			
11	0.05	1.8279	-3.8239	-12.982			
12	0.1	1.6233	-4.3044	-6.3333			
13	0.15	1.4028	-4.4697	-0.4229			
14	0.2	1.1809	-4.3634	4.4941			
15	0.25	0.9701	-4.0392	8.2743			
16	0.3	0.7796	-3.5556	10.874			
17	0.35	0.6163	-2.9708	13.336			

Solving numerically this differential system is very rapid.

Simply write the system matrix A in the worksheet, for example, in the range B5:D7. Insert the row starting values, [2,-3,-20] in the range B10:D10, where you want to begin the functions tabulation. Choose a suitable step that you like, for example h = 0.05. Then, select the range B11:D60 and insert the function

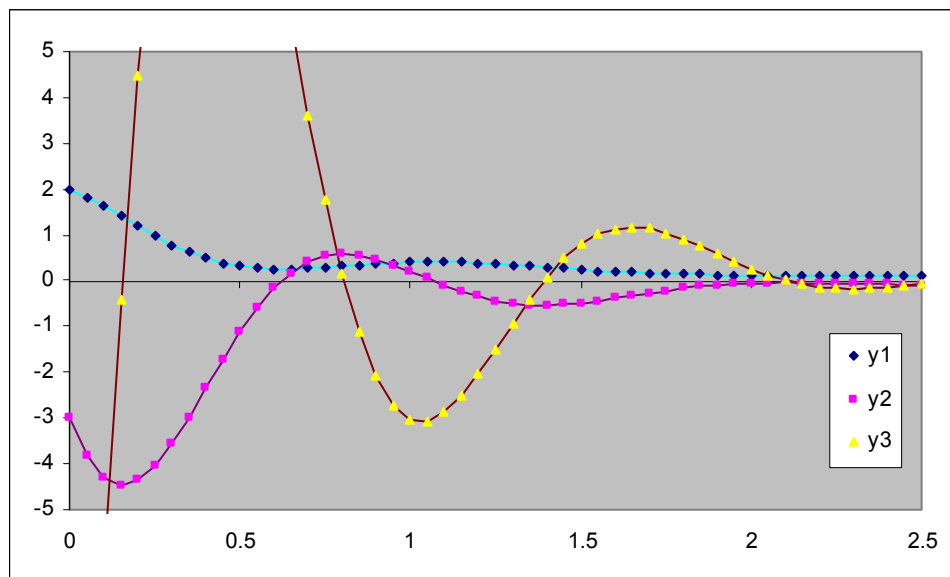
$=\text{ODE_SYSL}(\text{B5:D7};\text{B10:D10};\text{B3})$

And give the ctrl+shift+enter key sequence.

The selected cells will be automatically filled with the calculated solutions. Compare with the exact solution

$$y = e^{-x} + e^{-2x} \cos(5x)$$

The following graph shows the exact solutions (continue lines) compared with the calculated solutions (dotted lines). We note a global good fitting.



Macro ODE Solver

This macro performs the numerical integration of ordinary differential equations systems (ODE) using the *Runge-Kutta-Fehlberg* method of 4th order.

It is useful for solving ODE with initial values (Cauchy problem)

$$y' = f(x, y) \quad , \quad y(x_0) = y_0 \quad x_0 \leq x \leq x_{\max}$$

as well an ODE system

$$\mathbf{y}' = \mathbf{f}(x, \mathbf{y}) \quad , \quad \mathbf{y}(x_0) = \mathbf{y}_0 \quad x_0 \leq x \leq x_{\max}$$

where $\mathbf{y} = [y_1, y_2 \dots y_n]^T$ is the vector of depend variables

The macro works with equation defined by worksheet formulas (on site) or even with equation defined by symbolic string. The macro implements a control of the local error and changes the integration step consequently Therefore the precision can be set independently from the tabulating step. The independence of the integration step from the tabulation step is the main feature of this macro.

The macro implement a predefined simple input schema. Let's see how to use it with a simple example

Example 2. Find a numerical solution of this simple differential system

$$\begin{cases} y_1' = y_2 \\ y_2' = -4y_1 - 5y_2 \end{cases} \quad \text{with} \quad \begin{cases} y_1(0) = 0 \\ y_2(0) = 2 \end{cases} \quad \text{and with} \quad 0 \leq x \leq 5$$

Arrange the worksheet reserving two columns for $[y_1, y_2]$ and the two adjacent columns for the derivatives $[y_1', y_2']$. In the first column set a x-grid with 50 step of increment $\Delta x = 0.1$. Then select the area A4:E55 and start the macro ODE Solver.

	A	B	C	D	E
4	x	y ₁	y ₂	y ₁ '	y ₂ '
5	0	0	3	3	-15
6	0.1				
7	0.2				
8	0.3				
9	0.4				
10	0.5				
11	0.6				
12	0.7				
13	0.8				
14	0.9				
15	1				
16	1.1				
17	1.2				
18	1.3				
19	1.4				

ODE Solver

1st-order Vector Differential Equations Solver

Initial values

Equations Source

☒ Worksheet
 ☐ Symbolic

Differential Equations y' = f(x, y, ...)

D5:E5

Err.max

1E-6

Variables: x, y, ...

A5:C5

Steps

50

Starting values: x0, y0, ...

A5:C5

Length

5

☒ y'

Run

?

Differential Equations links to the cells where are the equation formulas: = C5, = -(4*B5+5*C5).

Variables are the cells (x, y₁, y₂) referenced by the formulas; in that case A4:C5

Starting values indicate the starting cells of the table (they may or not may coincide with the variables cells).

If we have followed the above schema, the input fields are correctly filled.

We have only to set the required precision: Err. Max = 1E-6

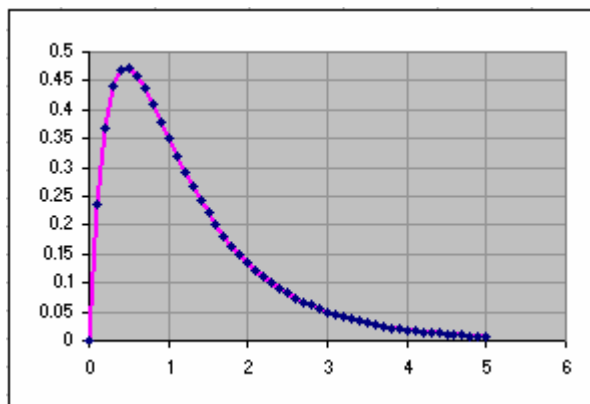
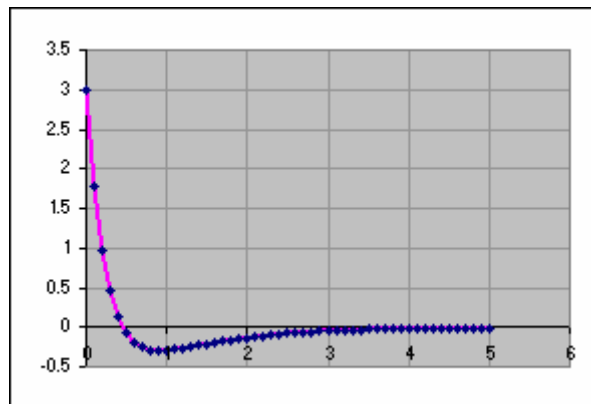
Optionally, if we want the macro to fill also the derivative columns, activate the check box



Press "Run" to start the RKF integration algorithm. After few seconds and about 1400 function evaluations, the macro output its result. Each solution node has a relative error less then 1E-6. Compare it with the exact solution

$$y_1 = e^{-x} - e^{-4x} \quad y_2 = -e^{-x} + 4e^{-4x}$$

The following graphs shows the exact solution (pink line) and the calculated solution (dotted line)

Solution $y_1(x)$ Solution $y_2(x)$

Note the general good fitting. It is a remarkable result overall if we consider the relative few points in the initial range where both functions have high swinging.

For further details see "ODE tutorial" by Foxes Team, 2006

Macro ODE - Slope Grid

This macro generate and visualize the slope $y'(x)$ of a 1st ODE solution over a rectangular domain. It is didactically useful for studying the 1st order differential equation $y' = f(x, y)$

For example we have to study the following equation

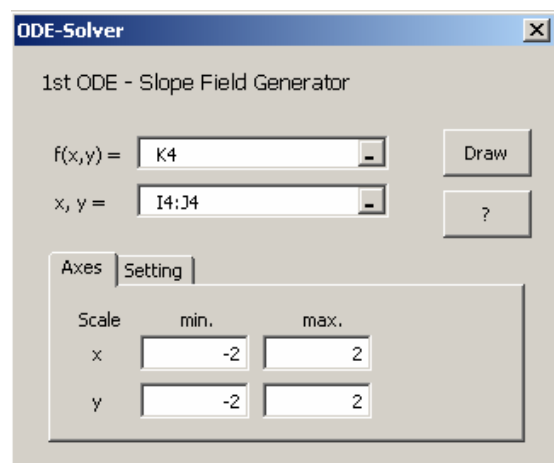
$$y' + y = e^{-x} \quad \Rightarrow \quad y' = xe^{-x} - y$$

Therefore we have to plot the slope given by the bivariate function $f(x, y) = e^{-x} - y$ in a suitable rectangular domain, for example in $D = \{-2 \leq x \leq 2, \{-2 \leq y \leq 2\}$

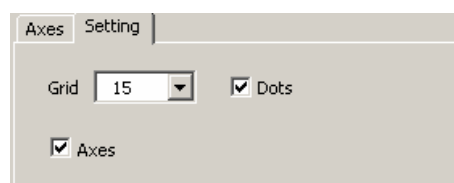
For using this macro, set the variables x and y in adjacent cells, for example I4 and J4. Then, in the adjacent right cell K4, insert the formula defining y' . thus: =EXP(-I4)-J4

	H	I	J	K
1				
2				
3		x	y	$f(x,y)$
4		1.75	1.75	-1.576226
5				

Select the range I4:K4 and start the macro ODE Slope Field



Option Tab



The Grid number, from 4 to 24, set the density of the grid. The dots check box adds the grid points to the plot. The axes check box adds the x-y axes

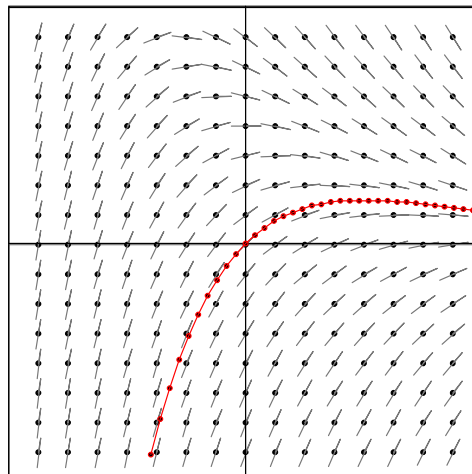
In the Axes tab we can set the domain D. Click the "Draw" button to generate the plot

The macro generates a graphical object showing the slope y' of each point of the grid.

Every solution $y(x)$ of the differential equation follows the direction of the slope field. Therefore the field gives a global view of the solutions crossing the given domain.

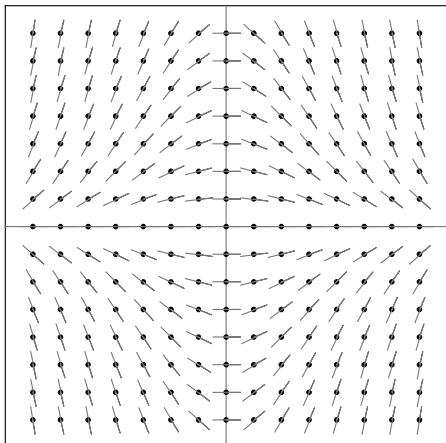
For example, the solution crossing the point $(0,0)$ is $y = e^{-x}x$ (red line)

We can see that this solution evolves following the slope directions. In this case, it never crosses the slope in any point.

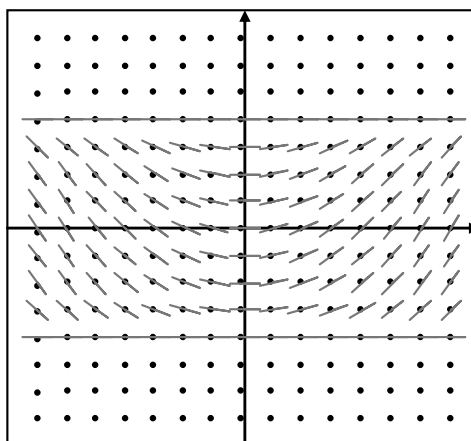


Other interesting examples

$$y' = -2xy$$



$$y' = x\sqrt{1-y^2}$$



Nonlinear Equations

Bisection

=Zero_bisec(a, b, func, [step], [param])

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

Param contains labels and values for parameters substitution (if there are)

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

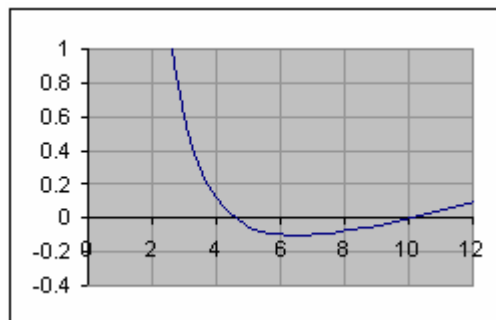
$$[a_2, b_2] \text{ where } a_1 < a_2 < x_0 < b_2 < b_1.$$

After several iterations, 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 in the interval $2 < x < 9$



	B6		f_x {=Zero_bisec(B5;C5;\$B\$2)}	
	A	B	C	D
1				
2	f(x) =	3*log(x)+2/log(x)-5		
3				
4	 b-a 	a	b	
5	7	2	9	
6	6.217E-15	4.641588834	4.641588834	
7				

Starting the algorithm with $a = 2$ and $b = 9$ we get

$$x_0 = 4.64158883361278$$

The root approximates the exact zero $x_0 = 100^{1/3}$ with an 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 taking the difference $|b-a|$

Secant

=Zero_sec(a, b, func, [step], [param], [DgtMax])

Approximates the zero of a monovariable function $f(x)$ with the secant method

$$f(x) = 0$$

This function needs two starting points $[a, b]$ bracketing the zero.

This version accepts also a worksheet function.

The parameter "Func" is a math string containing the symbolic function $f(x)$ or a range of two adjacent cells containing the worksheet functions $[f(a), f(b)]$ (see example).

Examples of correct symbolic function definitions are:

$$-2 \cdot \ln(x) \quad , \quad 2 \cdot \cos(x) - x \quad , \quad 3 \cdot x^2 - 10 \cdot \exp(-4 \cdot x) \quad , \quad \text{etc.}$$

The optional parameter "Step" sets the maximum number of iterations. If omitted, the function iterates until the convergence. Step = 1 is useful for investigation about the algorithm.

"Param" contains labels and values for parameters substitution (if there are)

Both parameters "step" and "param" are ignored if func is not a symbolic function.

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

At the first step, the function returns a new segment

$$[a_1, b_1] \quad \text{where} \quad a_1 < x_0 < b_1$$

At the second step, the function return a new segment

$$[a_2, b_2] \quad \text{where} \quad a_1 < a_2 < x_0 < b_2 < b_1.$$

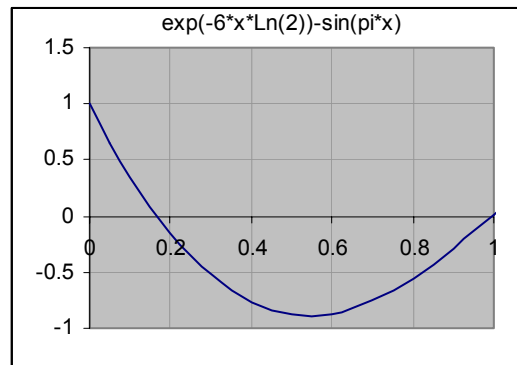
After several iterations, the interval $[a_n, b_n]$, with $n \gg 1$, will be very closed to the value x_0

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 zeros into the interval $0 < x < 0.5$



Starting the algorithm with $a = 0$ and $b = 0.5$ we get $x_0 = 0.166666666666667$

	A	B	C	D
1				
2	f(x) =	exp(-6x*ln(2))-sin(pi*x)		
3				
4	 b-a 	a	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	f(x) =	exp(-6x*ln(2))-sin(pi*x)				
3						
4	 b-a 	a	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 this convergence is much faster than the one of the bisection method. On the other hand, it is no guaranteed that the zero always remains bracketed into the interval.

Zero finder for worksheet function

Example we want to find the zero of the function $f(x) = x - \cos(x)$, using the standard worksheet function = COS(x). Arrange a worksheet like the following

	A	B	C	D	E
1	a	b	f(a)	f(b)	 a-b
2	0	1	-1	0.459697694	
3	1	0.685073357			
4					
5	{=Zero_sec(A2,B2,C2:D2)}		=A2-COS(A2)	=B2-COS(B2)	
6					
7					

Xnumbers Tutorial

Insert in A3:B3 the array function `zero_sec(a, b, func)` using the sequence Ctrl+Shift+Enter
In this case, none parameter is necessary because the function always returns one step of the secant algorithm.

Now insert the functions in the range C3:D3 by simply dragging-down the range C2:D2

	A	B	C	D	E
1	a	b	f(a)	f(b)	a-b
2	0	1	-1	0.459697694	
3	1	0.685073357	0.459697694	-0.089299276	0.3149266
4					
5					=ABS(A3-B3)
6					

Insert the function `=ABS(A3-B3)` in E3. This is not necessary but it is usefull for checking the convergence. Now, let's select the last row A3:E3 and drag it down until the convergence shows the minimal error

	A	B	C	D	E
1	a	b	f(a)	f(b)	a-b
2	0	1	-1	0.459697694	
3	1	0.685073357	0.459697694	-0.089299276	0.3149266
4	0.685073357	0.736298998	-0.089299276	-0.004660039	0.0512256
5	0.736298998	0.739119362	-0.004660039	5.7286E-05	0.0028204
6	0.739119362	0.739085112	5.7286E-05	-3.52926E-08	3.425E-05
7	0.739085112	0.739085133	-3.52926E-08	-2.66787E-13	2.109E-08
8	0.739085133	0.739085133	-2.66787E-13	0	1.594E-13
9	0.739085133	0.739085133	0	0	0
10					

We can compute this result also in multiprecision. In this case we have use, of course, the equivalent

x-functions; in cell D2 = `xsub(A4, xcos(A4))`, E2 = `xsub(B4, xcos(B4))` and in cell E3 = `ABS(xsub(A5, B5))`.

In the range A3:B3 insert the array function `{Zero_sec(A4, B4, C4:D4, , , 30)}`.

The parameter DgtMax = 30 switches the function in multiprecision mode

Following the other steps we get the result below

	A	B	C	D	E
1	a	b	f(a)	f(b)	a-b
2	0	1	-1	0.459697694131860282599063392558	0.314926643
3	1	0.685073357326045096500576206326	0.459697694131860282599063392558	-0.089299276481860004547947773132	0.05122564
4	0.685073357326045096500576206326	0.736298997613653998501782889252	-0.089299276481860004547947773132	-0.004660039038142629626359295782	0.002820364
5	0.736298997613653998501782889252	0.73911936191162923523880635759	-0.004660039038142629626359295782	0.000057285991106021233302021666	3.42498E-05
6	0.73911936191162923523880635759	0.739085112127463889571060046119	0.000057285991106021233302021666	-0.000000035292622787723775849748	2.10875E-08
7	0.739085112127463889571060046119	0.739085133215001266215496512418	-0.000000035292622787723775849748	-0.00000000000000000000000000000001	1.59375E-13
8	0.739085133215001266215496512418	0.739085133215160641656054183912	-0.00000000000000000000000000000001	0.00000000000000000000000000000001	7.42096E-22
9	0.739085133215160641656054183912	0.739085133215160641655312087673	0.00000000000000000000000000000001	-0.00000000000000000000000000000001	0
10	0.739085133215160641655312087673	0.739085133215160641655312087673	-0.00000000000000000000000000000001	-0.00000000000000000000000000000001	
11					

As we can see, the final value is

$$x = 0.739085133215160641655312087673$$

that is the zero of the function $f(x) = x - \cos(x)$ with 30 significant digits

Derivatives

First Derivative

=Diff1(x, func, [lim], [param])

Approximates the first derivative of an univariate function $f(x)$ at the given point x

$$f'(x) = \frac{d}{dx} f(x)$$

The parameter "func" 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 $\text{lim} = 1$, it approaches from the right; if $\text{lim} = -1$, it approaches from the left; if $\text{lim} = 0$, it approaches centrally. That is, it returns the following derivatives

$$f'(x) = \begin{cases} \lim_{h \rightarrow 0^-} f(x) = f'(x^-) \\ \lim_{h \rightarrow 0} f(x) = f'(x) \\ \lim_{h \rightarrow 0^+} f(x) = f'(x^+) \end{cases}$$

"Param" contains labels and values for parameters substitution (if there are)

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, func, [param])

It approximates the second derivative of an univariate function $f(x)$ at the given point

$$f''(x) = \frac{d^2}{dx^2} f(x)$$

The parameter "func" 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.

"param" contains labels and values for parameters substitution (if there are)

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, [param])

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.

"param" contains labels and values for parameters substitution (if there are)

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	df/dx =	-0.055555556		
6	df/dy =	-0.277777778		
7				

Jacobian matrix

=Jacobian (p, func, [param])

Approximates the Jacobian's matrix of a multivariate vector-function $\mathbf{F}(x, y, z)$ at the given point (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.

"param" contains labels and values for parameters substitution (if there are)

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, [param])

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:

$-2 \cdot \ln(x+3y)$, $2 \cdot \exp(-x) \cdot \cos(3 \cdot t)$, $3 \cdot x^2 - y^2 + z^2$, $(x^2 + y^2)^{(1/3)}$, etc.

Xnumbers Tutorial

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.

"Param" contains labels and values for parameters substitution (if there are)

This function returns a square a matrix (n x n) of the second derivatives

The accuracy of this function is about 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		{=hessian(B1:B3;B4)}				
6						
7						

Parameters. All the differential functions accept parameters into the string definition. They must be different from the canonical variables "x", "y", "z", "t" but apart this, can have any name that you like.

Example: Calculate the Hessian matrix of the function

$$f(x, y, z) = \frac{k \cdot z}{x^2 + y^2 + a}$$

for (x, y, z) = (1, -3, 4) and for parameters k = 1.5 and a = -1

	A	B	C	D	E	F	G	H	I
2	f(x,y,z)		x	y	z		k	a	
3	k*z/(x^2+y^2+a)		1	-3	4		1.5	-1	
4									
5	{=Hessian(C3:E3,A3,G2:H3)}		=Hessian				-20	-48	-9
6							-48	108	27
7							-9	27	0
8									
9									

Note the "param" range G2:H3: the parameters must be always passed with their labels "k" and "a". On the contrary the variables do not need labels but must always be passed in the order x, y, z.

Using the trick of the scaling for 243, the inverse of the determinant, we get an integer matrix, that represent the numerator of the Hessian itself. Then, simplifying each element for 243, we obtain the Hessian matrix in the following exact fractional form

$$\begin{bmatrix} -\frac{20}{243} & -\frac{16}{81} & -\frac{1}{27} \\ -\frac{16}{81} & \frac{4}{9} & \frac{1}{9} \\ -\frac{1}{27} & \frac{1}{9} & 0 \end{bmatrix}$$

Non-linear equation solving with derivatives

Derivatives play a strategic role in solving non-linear equation and non-linear system. The most efficient algorithms use the derivatives information in order to speed up the convergence to the final solution. From the point of view of numeric calculus, derivatives are rarely used because they tend to magnify the truncation error. This is general true and a many rootfinding algorithm avoid the derivatives. In solving non-linear problem, however, the derivatives can be very useful because they can greatly improve the convergence without influence the final result accuracy, that depends only by the evaluation function $f(x)$.

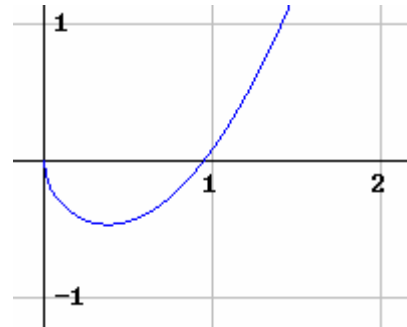
Let's see an example.

Solve the following equation $x^2 = \sqrt{\sin x}$ with an accuracy better than $1e-25$.

First of all we build the function

$$f(x) = x^2 - \sqrt{\sin x}$$

and draw its plot. The point x where $f(x) = 0$ is the solution of the given equation. We see that the zero exists and it is near the point 1. We note also that in the interval $[0.5, 1.5]$ the function is monotonic.



In this interval the Newton-Raphson iterative algorithm, starting from $x = 1.5$, should work fine.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad n = 0, 1, 2, \dots$$

To implement this algorithm we need the evaluation function $f(x)$ with about 30 significant digits. For that, it comes in handy the multiprecision function `xeval`. For the derivative we have two ways: computing the function $f'(x)$ by hand and evaluating it by `xeval` or approximating the derivative by the function `diff1` in standard precision. Because we are a bit lazy and the derivative is not so immediate, we chose the second way. A simple spreadsheet arrangement may be the following.

	A	B	C	D
1	Non linear equation solving			
2			<code>=xeval("x-f/d",B6:D6)</code>	
3	f(x) =	<code>x^2-sqr(sin(x))</code>	<code>=xeval(\$B\$3,B7)</code>	<code>=Diff1(B7,\$B\$3)</code>
4				
5	n	x	f(x)	f'(x)
6	0	1.5	1.25125329206847723850668540128	2.964587016
7	1	1.07793335625983199521740412906	0.223333983281382979568107523153	1.903817461
8	2	0.960624849631271235775168557437	0.01751021881416934136889068507	1.604772064
9	3	0.949713506390634122542849402982	0.000152648793721466707141738172	1.576786679
10	4	0.949616696342844112460225922115	0.000000012026119161818663594115	1.576538231
11	5	0.949616688714662934543206760606	0.0000000000000000074715651610379	1.576538211
12	6	0.949616688714662947150983031913	0.000000000000000000000000000025	1.576538211
13	7	0.949616688714662947150983031754	-0.000000000000000000000000000001	1.576538211

As we can see the convergence is superb!. After few iteration the solution is

$$x \cong 0.9496166887146629471509830317 \quad \text{with } |f(x)| < 1e-28$$

This excellent result has been obtained in spite of the limited 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

$$\begin{cases} f(x, y) = 0 \\ g(x, y) = 0 \end{cases} \Rightarrow \begin{pmatrix} x \\ y \end{pmatrix}_{n+1} = \begin{pmatrix} x \\ y \end{pmatrix}_n - \begin{bmatrix} f_x & f_y \\ g_x & g_y \end{bmatrix}_n^{-1} \begin{pmatrix} f \\ g \end{pmatrix}_n$$

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

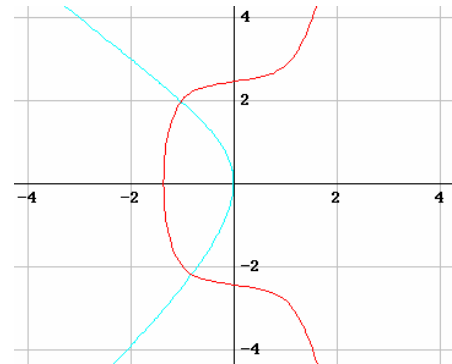
$$\begin{cases} -x^5 + y^2 - x - 6 = 0 \\ e^{x-y} + x + y - 1 = 0 \end{cases}$$

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 another 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			
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		

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:B8.

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 converges 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

$$A_{dB} = 20 \log_{10}(A)$$

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) ¹

any base (2 - 36) \Rightarrow any base (2 - 36)

xBaseChange ²

Any base (2 .. >1 million)

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

¹ The function `baseChange` appears thanks to Richard Huxtable

² The function `xBaseChange` appears thanks to Ton Jeursen

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.

	A	B	C	D
1	Examples using the changeBase function			
2	Convert a number into many different bases			
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 consequently its results have the same format; the cell D4 is formatted with 2 decimals and its result has the same format.

Multiprecision Base Conversion

= xBaseChange(xBC_OldNum, [xBC_NewBase], [deep])

This function¹ converts a number from any base (2 .. >1 million). xBC_OldNum is the number to convert in the following formats:

3642455[7] (integer number in base 7)
 10C2AE[16] (integer number in base 16)
 1098414 (integer number in base 10)
 213.02[4] (decimal number in base 4)

The oldnumber is assumed to be base 10; otherwise oldbase is provided between square brackets [].

xBC_NewBase is the new base. If omitted it is assumed 10 as default.

If the base > 35 the letters are no more sufficient and the digits are represented as group of digits separate with colon :

9:34:9:23[36] (integer number in base 36)
 6:54:122:42:65[200] (integer number in base 200)
 100:368.356:980[1000] (decimal number in base 1000)

deep is an optional parameter (0 to 50) setting the max number of decimals showed (default 50)

Example: xBaseChange("9:34:9:23[36]", 7)
 Results in: 3642455[7]

Example: xBaseChange(1098414, 16)
 Results in: 10C2AE[16]

¹ The function xBaseChange appears thanks to Ton Jeursen.

Xnumbers Tutorial

Example: `xBaseChange(1098414 , 46)`

Results in: `11:13:4:26[46]`

Example: `xBaseChange("1F0C2AE[16]")`

Results in: `32555694[10]`

Example `xBaseChange(10092.102 , 2048 , 6)`

Results in `4:1900.208:1835:16:786:884:1507...[2048]`

Note that, as in the last case, the digits groups can have different length: one , two, or more digits

If we want to force all the groups having the same number of digits simply insert the new base with prefix "0"

Example `xBaseChange(10092.102 , "02048" , 6)`

Results in `0004:1900.0208:1835:0016:0786:0884:1507...[02048]`

Log Relative Error

= mjkLRE(q, c, NoSD)

= xLRE(q, c, NoSD, [DgtMax])

This function¹ 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 π . 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`

This means that two values are close for about 10 significant digits. LRE metric rejects non significant digits. Look at this example:

Certified value $C = 0.000133333333333333$

¹ 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

Approx. value $Q = 0.000133333333333311$

$\text{mjkLRE}(C, Q, 15) = 12.8$

The two numbers appear exact up to the 17th 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 numeric 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 1E-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 1E-14 for any $x \neq 0$

This version¹ accept also negative argument

For definition is

$$Ei(-x) = E_1(x) \quad x > 0$$

Exponential integral En(x)

expn_integr(x, n)

Returns the exponential integral of n-th order

$$E_n(x) = - \int_1^{\infty} \frac{e^{-xt}}{t^n} dt$$

Accuracy: about 1E-14 for $x > 0$ and $n > 0$

¹ This extension appears thanks to the courtesy of Hans Günter

Euler-Mascheroni Constant γ

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, [Prec])

Returns the gamma function with precision up to 30 digits.

Definition:

$$\Gamma(x) = \int_0^{\infty} t^{x-1} e^{-t} dt$$

Optional parameter Prec sets the maximum precision from 15 (default) to 30 digits

When Prec = 15 the routine works in faster double arithmetic.

This routine uses the excellent Lanczos series approximation¹

$$\Gamma(x) \cong \frac{\sqrt{2\pi}}{e^g} \left(\frac{x+g+\frac{1}{2}}{e} \right)^{x+\frac{1}{2}} \left(c_0 + \sum_{i=1}^N \frac{c_i}{x+i} \right)$$

where: g = 607/128, ci are the Lanczos' coefficients and 14 < N < 22.

Relative accuracy in double precision is better than 10⁻¹⁴, (except very near to the poles x = 0, -1, -2, -3, etc.). In extended precision the accuracy is better then 1E-30

xGamma(5.1) = 27.9317537383684

xGamma(5.1, 30) = 27.9317537383683833586731052773

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
0.0001	9.99942288323162E+3	1.00E-15
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 This high accuracy algorithm has been extracted form a very good note by Paul Godfrey, Intersil , C.2001

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 5E-15 in any case.

You can convert in double only the values with $x \leq 170$, 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 efficece and accuracy, this function can also be used to calculate the factorial of a big integer number, using the relation

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

Example:

xfact(10002) =	2.84711361574652325360317551421E+35667	30 digits, slower
xgamma(10003) =	2.84711361574651E+35667	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. 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 1E-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

(with a rel. error = 3.846E-10)

As we can see, In this case, the error is more than 500,000 times bigger that 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 1E-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 1E-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(a, b)

Returns the beta function

$$B(a, b) = \int_0^1 t^{a-1} (1-t)^{b-1} dt$$

Relative accuracy is better than 1E-14, for $a > 0$ and $b > 0$

Incomplete Gamma function

xGammal(a, x, [sel])

Returns one of the following Incomplete Gamma functions specified by the "sel" parameter (default 1)

Sel =1	Sel =2	Sel = 3	Sel = 4 (Tricomi)
$\gamma(a, x) = \int_0^x t^{a-1} e^{-t} dt$	$\Gamma(a, x) = \int_x^\infty t^{a-1} e^{-t} dt$	$P(a, x) = \frac{\gamma(a, x)}{\Gamma(a)}$	$\gamma^*(a, x) = \frac{\gamma(a, x)}{x^a \Gamma(a)}$

Relative accuracy is better than 1E-14 for : $a > 0, x \geq 0$

Incomplete Beta function

xBetal(x, a, b, [sel])

Returns one of the following Incomplete Beta functions specified by the "sel" parameter (default 1)

Sel = 1	Sel = 2
$B_x(a, b) = \int_0^x t^{a-1} (1-t)^{b-1} dt$	$I_x(a, b) = \frac{B_x(a, b)}{B(a, b)}$

where B(a, b) is the Beta function

Relative accuracy is better than 1E-14 for $a > 0, b > 0, 0 \leq x \leq 1$.

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 1E-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 of integer order

BesselJ (x, [n])	Bessel function of 1° kind, order n: $J_n(x)$
BesselY (x, [n])	Bessel function of 2° kind, order n: $Y_n(x)$
BesseldJ (x, [n])	First derivative of Bessel functions of 1° kind, order n: $J'_n(x)$
BesseldY (x, [n])	First derivative of Bessel functions of 2° kind, order n: $Y'_n(x)$
Bessell (x, [n])	Modified Bessel function of 1° kind, order n: $I_n(x)$
BesselK (x, [n])	Modified Bessel function of 2° kind, order n: $K_n(x)$
BesseldI (x, [n])	First derivative of mod. Bessel functions of 1° kind, order n: $I'_n(x)$
BesseldK (x, [n])	First derivative of mod. Bessel functions of 2° kind, order n: $K'_n(x)$

Relative accuracy is better than 1E-13, for $x > 0$ and n any integer

These routines¹ 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 is better 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 1E-13, for $x > 0$

Sine Integral Si(x)

SinIntegral(x)

Returns the sine integral defined as:

$$si(x) = \int_0^x \frac{\sin(t)}{t} dt$$

Relative accuracy is better than 1E-13, for $x > 0$

¹ 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.

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 1E-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 1E-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

Hypgeom(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 1E-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 results surrounding the prime number, series, integrals, etc.

Relative accuracy is better than 1E-14, for any $s \neq 1$

It uses the fast Borwein formula.

Definition. For $|s| > 1$ the function is defined as:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s}$$

Analytic continuation. It can be defined for $0 < s < 1$ by the following analytic continuation:

$$\zeta(s) = \frac{1}{1-2^{1-s}} \sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k^s}$$

For $s < 0$ the function is defined by the following relation:

$$\zeta(1-s) = 2(2\pi)^{-s} \cos\left(\frac{1}{2}s\pi\right) \Gamma(s) \zeta(s)$$

Same 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 $(\pi^2)/6 - 5/4$

Airy functions

= AiryA(x)

= AiryB(x)

= AiryAD(x)

= AiryBD(x)

Return the Airy functions A(x), B(x) and theirs derivatives A'(x), B'(x)

$$A(x) = \frac{1}{3^{2/3}\pi} \sum_{n=0}^{\infty} \frac{\Gamma\left(\frac{1}{3}(n+1)\right)}{n!} (3^{1/3}x)^n \sin\left(\frac{2(n+1)\pi}{3}\right)$$

$$B(x) = \frac{1}{3^{1/6}\pi} \sum_{n=0}^{\infty} \frac{\Gamma\left(\frac{1}{3}(n+1)\right)}{n!} (3^{1/3}x)^n \left| \sin\left(\frac{2(n+1)\pi}{3}\right) \right|$$

Variable x is real. Relative accuracy is better than 1E-14

Elliptic Integrals

= IElliptic1(φ, k)

= IElliptic2(φ, k)

Return the Jacobian Elliptic Integrals of 1st and 2nd kind

$$IElliptic1 = F(\phi, k) = \int_0^{\phi} \frac{d\theta}{\sqrt{1 - k^2 \sin^2 \theta}}$$

$$IElliptic2 = E(\phi, k) = \int_0^{\phi} \sqrt{1 - k^2 \sin^2 \theta} d\theta$$

Variable φ is real, 0 < k < 1. Relative accuracy is better than 1E-14

Kummer confluent hypergeometric functions

= Kummer1(a, b, z)

= Kummer2(a, b, z)

They return the Kummer confluent hypergeometric functions of 1st and 2nd kind, also called $M(a, b, z)$, and $U(a, b, z)$ respectively.

The confluent hypergeometric functions give the complete solutions of the confluent hypergeometric differential equation.

$$z \frac{d^2 w}{dz^2} + (b - a) \frac{dw}{dz} - aw = 0$$

having the complete solution

$$w = q_1 M(a, b, z) + q_2 U(a, b, z)$$

The parameters "a" and "b" can be positive or negative but always different from 0, -1, -2, -3 ...
The variable z must be positive for Kummer2

For $a > 0$ and $b > 0$, the confluent hypergeometric functions have an integral representation (Abramowitz and Stegun 1972, p. 505)

$$M(a, b, z) = \frac{\Gamma(b)}{\Gamma(b-a)\Gamma(b)} \int_0^1 e^{zt} t^{a-1} (1-t)^{b-a-1} dt$$

$$U(a, b, z) = \frac{1}{\Gamma(a)} \int_0^\infty e^{-zt} t^{a-1} (1+t)^{b-a-1} dt$$

a	b	z	M(a,b,z)	U(a,b,z)
0.5	0.2	1	5.85803416181	0.70928623024
0.5	0.2	5	601.287114305	0.40255172241
0.5	0.2	10	112016.644576	0.29849708947
0.5	0.2	20	3063046479.13	0.21689337800
0.5	0.2	30	7.6396125E+13	0.17882728338
8.2	1.4	1	41.1797940694	1.39280467E-06
-0.9	0.4	10	-345.535973205	7.72952303674

Integral of sine-cosine power

= IntPowSin(n, x)

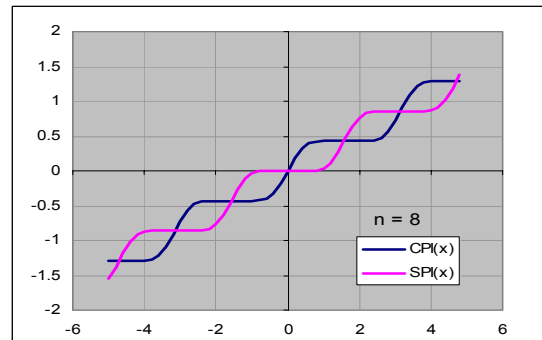
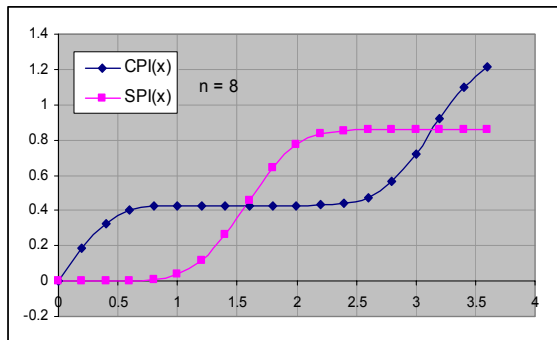
= IntPowCos(n, x)

These functions return the integral of sine and cosine raised to the n integer power

$$SPI(x, n) = \int_0^x (\sin t)^n dt$$

$$CPI(x, n) = \int_0^x (\cos t)^n dt$$

Variable $x \geq 0$ is real, $n = 0, 1, 2, 3 \dots$ is integer. Relative accuracy is better than $1E-15$



These functions use the fast recursive algorithms

$$SPI(x, n) = -\frac{\cos x (\sin x)^{n-1}}{n} + \frac{n-1}{n} SPI(x, n-2),$$

$$SPI(x, 1) = 1 - \cos x, \quad SPI(x, 0) = x$$

$$CPI(x, n) = \frac{\sin x (\cos x)^{n-1}}{n} + \frac{n-1}{n} CPI(x, n-2),$$

$$CPI(x, 1) = \sin x, \quad CPI(x, 0) = x$$

The recursive formula of $CPI(x, n)$ is very stable and accurate for all values $x \geq 0$, $n \geq 0$, $n \in \mathbb{N}$. On the contrary, the recursive formula of $SPI(x, n)$ degrades precision for small value of x . In that case, the function automatically switches on the power expansion series

$$\int_0^x (\sin t)^n dt = y^{n+1} \left(\frac{1}{n+1} + \sum_{k=1}^{\infty} \frac{a_k}{2k+n+1} y^{2k} \right), \quad x < 1$$

where

$$a_k = \prod_{i=1}^k \left(\frac{2i-1}{2i} \right) = \frac{1 \cdot 3 \cdot 5 \dots (2k-1)}{2 \cdot 4 \cdot 6 \dots (2k)}, \quad y = \sin x$$

Spherical Bessel functions of integer order

= BesselSphJ(x, [n])

= BesselSphY(x, [n])

These functions return the Spherical Bessel functions of 1st and 2nd kind of integer order $n = 0, 1, 2, \dots$ (default $n = 0$), and for any real $x > 0$, defined as:

$$j_n(x) = \sqrt{\frac{\pi}{2x}} J_{n+\frac{1}{2}}(x) \quad \text{Spherical Bessel functions of 1st kind}$$

$$y_n(x) = \sqrt{\frac{\pi}{2x}} Y_{n+\frac{1}{2}}(x) \quad \text{Spherical Bessel functions of 2nd kind (also Neuman function)}$$

The Spherical Bessel functions are the solutions of the Helmholtz radial equation in spherical coordinates

$$x^2 \frac{d^2 y}{dx^2} + 2x \frac{dy}{dx} + (x^2 - n^2 - n)y = 0$$

n	x	$j_n(x)$	$y_n(x)$
0	0.5	0.958851077208406	-1.755165123780750
1	1	0.301168678939757	-1.381773290676040
2	3	0.298637497075734	-0.267038335264499

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],)

xeval(Formula, [Var1, Var2 ...])

These functions return the evaluation of a math expression in multiprecision arithmetic. They use the same algorithm¹ 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 **xeval** 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 cell references, like A1, \$B\$2, etc.

Example. Tabulate the following function for x = 1, 1.5, 2, ... with 30 significant digits

$$f(x) = \frac{1+x}{\sqrt{1+x^2}}$$

¹ 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.

	A	B	C	D
1				
2		$f(x) = (1+x) / \text{sqr}(1+x^2)$		
3				
4	x	f(x)		
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 correspondent 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	function	x	y	result
3	$(\ln(y)+x*y) / \text{sqr}(y)$	2.4	5.5	6.35540594073030048985687628091
4				=xeval(A3;B2:C3)
5				
6				

variables range with header B2:C3

Note. If we pass the range B3:C3 without 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

Argomenti funzione

xeval

Formula A3 = "(ln(y)+x*y) / sqr(y)"

VarArray =

DgtMax =

AngleSet =

Multiprecision Expression Evaluation.

= "y x "

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"

Xnumbers Tutorial

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 any 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 **xevall** only assigns the cell value to the variable on top to the correspondent column. Otherwise it assigns the value to the variable on the left to the correspondent row.

Complex Expression Evaluation

=cplxeval(Formula, [Var1, Var2 ...])

This function¹ 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 $z = 2 - i$

$$z^2 + (3+i)z + (2+5i)$$

Note that we use the complex rectangular format, i.e. "3+2j", only in the symbolic math formula.

	A	B	C	D
1	f(z)	z		Result
2	$z^2 + (3+2j)z + 2+5j$	2		13
3		-1		2
4				
5	{=cplx_eval(A2:B2:B3)}			

When we pass a complex value to a 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.

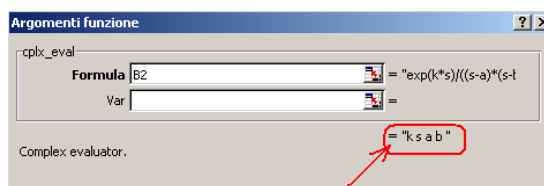
Example. Compute the expression for the given complexes values

$$F(s) = \frac{e^{ks}}{(s-a)(s-b)}$$

$$s = 1 + j, a = 1 - 4j, b = 3 + 6j, k = -0.5$$

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				



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**

¹ This function uses the clsMathParserC class by A. De Grammont and L. Volpi

Multiprecision Excel Formula Evaluation

= xCalc(Formula, [DgtMax])

This function¹ is useful for performing multiprecision calculation directly in a worksheet. Formulas can be entered just like in Excel and it may also contain references to other cells. The optional parameter "DgtMax" – from 1 to 200 – sets the maximum number of precision digits (default=30).

This smart functions has several advantages: it is easy to use and can handle long formula; the formula is not changed at all and can keep the references to other cells. The only drawback remains its speed because the function have to parse the formula and evaluate the result using the multiprecision arithmetic; therefore the evaluation takes about 5-6 times slower then basic x-functions.

Of course, not all the Excel functions can be converted. The functions recognized are.

+ - * / ^ ABS ACOS ACOSH ASIN ASINH ATAN ATANH
 COMBIN COS COSH EXP FACT INT LN LOG MOD PI
 ROUND SIGN SIN SINH SQRT TAN TANH

Nota. The functions are indicated with their original English names, that usually are different from the local names.

Let's see how it works. Assume to have in a cell the following Excel formula

= (A2*EXP(B2)+A4*EXP(B4))/(A2^2+A4^2)

	D2				
	A	B	C	D	E
1					
2	22	-0.5	y =	0.014082079	
3					
4	-16	-1.7			
5					

Note that the formula references the values of other 4 cells: A2, B2, A4, B4
 The formula value, calculated in standard precision, is 0.0140820785524786

We can get the value in 30 significant digits applying the xCalc function

=xCalc((A2*EXP(B2)+A4*EXP(B4))/(A2^2+A4^2))

For that, simply, select the cell D2 and move the mouse to the input field; insert xCalc(...) just around the original formula.

	A	B	C	D	E	F
1						
2	22	-0.5	y =	1.40820785524786228590556437374E-2		
3						
4	-16	-1.7				
5						

The result in the cell D4 has now a precision of 30 significant digits
 Observe that the the external cells links remain active and the formula is unchanged as in a standard Excel formula

¹ The function xCalc appears thanks to Ton Jeursen.

Xnumbers Tutorial

Note that, using the x-function, the above formula should be translated as

```
=xdiv(xadd(xmult(A2,xexp(B2)),xmult(A4,xexp(B4))),xadd(xpow(A2,"2"),xpov(A4,"2")))
```

The result is equivalent but it is evident the great advantage to keep the original formula.

Math expression strings

Functions like **Integr**, **Series**, **xeval**, **xeval**, **cpixeval** operate with symbolic math expressions by the aid of **clsMathParser** and **claMathparserC** (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 multiprecision 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<2) , (x<=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 ⇔ 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`

Note: This version support also the comma "," as decimal separator, depending on your system internation option setting.

`1,3333 -0,00025 1,2345E-12`

Logical expressions are now supported

`x<1 x+2y >= 4 x^2+5x-1>0 t<>0 (0<x<1)`

Logical expressions return always 1 (True) or 0 (False). Compact expressions, like "0<x<1", are now supported; you can enter: (0<x<1) as well (0<x)*(x<1)

Numerical range can be inserted using logical symbols and Boolean functions. For example:

For $2<x<5$ insert $(2<x)*(x<5)$ or also $(2<x<5)$
 For $x<2$, $x>=10$ insert $\text{OR}(x<2, x>=10)$ or also $(x<2)+(x>=10)$
 For $-1<x<1$ insert $(x>-1)*(x<1)$, or $(-1<x<1)$, or also $|x|<1$

Piecewise Functions. Logical expressions can also be useful for defining a piecewise function, such as:

$$f(x) = \begin{cases} 2x-1-\ln(2) & x \leq 0.5 \\ \ln(x) & 0.5 < x < 2 \\ x/2-1+\ln(2) & x \geq 2 \end{cases}$$

The above function can be written as:

$$f(x) = (x \leq 0.5) * (2*x-1-\ln(2)) + (0.5 < x < 2) * \ln(x) + (x \geq 2) * (x/2-1+\ln(2))$$

The parser adopts a new algorithm for evaluating math expressions depending on logical expressions, which are evaluated only if the logical conditions are true (Conditioned-Branch algorithm). Thus, the above piecewise expression can be evaluated for any real value x without any domain error. Note that without this features the formula could be evaluated only for x>0. Another way to compute piecewise functions is splitting it into several formulas (see example 6)

Math Constants supported are: Pi Greek (π), Euler-Napier (e)

```
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 29d 59m 60s

```
sin(29d 59m 60s) => 0.5      ,      29d 59s 60m => 30
```

Note This format is only for sexagesimal degree. It is independent from the unit set

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 $\sqrt{-1}$

Functions

Functions are called by their function-name followed by parentheses. Arguments can be: numbers, variables, expressions, or even other functions.

```
sin(x)      log(x)      cos(2*pi*t+phi)      atan(4*sin(x))
max(a,b)     root(x,y)    comb(n,k)      beta(x,y)
```

Xnumbers Tutorial

For functions having more than one argument, the successive arguments are separated by commas "," or alternatively, by semicolon ";" if your system has comma as decimal separator.

`max(a;b)` `root(x;y)` `comb(n;k)` `beta(x;y)` (only if comma "," is decimal sep.)

`max(a,b)` `root(x,y)` `comb(n,k)` `beta(x,y)` (only if point "." is decimal sep.)

List of basic functions and operators

The following table lists all functions and operators recognized in a math expression string

Use: R = Real, C = Complex, M = Multiprecision

The functions with M can be evaluated either in standard or multiprecision

The functions with C alone must be used with **cplxeval** only

The functions with R alone must be used with **xeval**, **xeval** setting DgtMax = 0 (standard precision)

Function	Use	Description	Note
-	R C M	subtraction	
!	R C M	factorial	5!=120 (the same as fact)
%	R M	percentage	35% = 0.35 , 100+35% =103.5
*	R C M	multiplication	
/	R C M	division	35/4 = 8.75
\	R C M	integer division	35\4 = 8
^	R C M	raise to power	3^1.8 = 7.22467405584208
	R C M	absolute value	-5 =5 (the same as abs)
+	R C M	addition	
<	R M	less than	return 1 (true) 0 (false)
<=	R M	equal or less than	returns 1 (true) 0 (false)
<>	R M	not equal	returns 1 (true) 0 (false)
=	R M	equal	returns 1 (true) 0 (false)
>	R M	greater than	returns 1 (true) 0 (false)
>=	R M	equal or greater than	returns 1 (true) 0 (false)
abs(x)	R C M	absolute value	abs(-5)= 5
acos(x)	R C M	inverse cosine	argument -1 ≤ x ≤ 1
acosh(x)	R C M	inverse hyperbolic cosine	argument x ≥ 1
acot(x)	R M	inverse cotangent	
acoth(x)	R M	inverse hyperbolic cotangent	argument x<-1 or x>1
acsc(x)	R M	inverse cosecant	
acsch(x)	R M	inverse hyperbolic cosecant	
alog(z)	C	complex exponential with base 10	10^z
and(a,b)	R	logic and	returns 0 (false) if a=0 or b=0
arg(z)	C	polar angle of complex number	-pi < a ≤ pi
asec(x)	R M	inverse secant	
asech(x)	R M	inverse hyperbolic secant	argument 0 ≤ x ≤ 1
asin(x)	R C M	inverse sine	argument -1 ≤ x ≤ 1
asinh(x)	R C M	inverse hyperbolic sine	
atanh(x)	R C M	inverse hyperbolic tangent	argument -1 < x < 1
atn(x), atan(x)	R C M	inverse tangent	
beta(x,y)	R C	beta	argument x>0 y>0
betal(x,a,b)	R	Beta Incomplete function	x >0 , a >0 , b >0
cbr(x)	R	cube root	cbr(2) =1.25992104989487, also 2^(1/3)
clip(x,a,b)	R	Clipping function	return a if x<a , return b if x>b, otherwise return x.
comb(n,k)	R C M	combinations	comb(6,3) = 20
conj(x)	C	conjugate	
cos(x)	R C M	cosine	argument in radians
cosh(x)	R C M	hyperbolic cosine	
cot(x)	R M	cotangent	argument (in radians) x≠ k*π with k = 0, ±1, ±2...
coth(x)	R M	hyperbolic cotangent	argument x>2
csc(x)	R M	cosecant	argument (in radians) x≠ k*π with k = 0, ±1, ±2...
csch(x)	R M	hyperbolic cosecant	argument x>0
dec(x)	R M	decimal part	dec(-3.8) = -0.8
deg(x)	R M	degree sess. conversion	conv. degree (60) into current unit of

Xnumbers Tutorial

			angle
digamma(x)	R C	digamma	argument $x > 0$
Ei(x)	R C	exponential integral function	argument $x > 0$
Ein(x,n)	R	Exponential integral of n order	$x > 0$, $n = 1, 2, 3, \dots$
Elli1(x)	R	Elliptic integral of 1st kind	$\forall \phi$, $0 < k < 1$
Elli2(x)	R	Elliptic integral of 2st kind	$\forall \phi$, $0 < k < 1$
erf(x)	R C	error Gauss's function	argument $x > 0$
erfc(x)	R C	error Gauss's function for complex value	
eu	R M	Euler-Mascheroni constant	Gamma constant 0.577..
exp(x)	R C M	exponential	$\exp(1) = 2.71828182845905$
fact(x)	R M	factorial	argument $x > 0$
fix(x)	R M	integer part	$\text{fix}(-3.8) = 3$
FresnelC(x)	R	Fresnel's cosine integral	$\forall x$
FresnelS(x)	R	Fresnel's sine integral	$\forall x$
gamma(x)	R C	gamma	argument $x > 0$
gammaln(x)	R C	logarithm gamma	argument $x > 0$
gcd(a,b)	R	greatest common divisor	The same as mcd
grad(x)	R	degree cent. conversion	conv. degree (100) into current unit of angle
HypGeom(x,a,b,c)	R	Hypergeometric function	$-1 < x < 1$ $a, b > 0$ $c \neq 0, -1, -2, \dots$
im(z)	C	imaginary part of complex number	
int(x)	C	integer part	$\text{int}(-3.8) = 4$
integral(f,z,a,b)	C	Def. integral of complex function f(z)	$\text{Integral}('1/z^2', 'z', 1-i, 1+i)$
inv(x)	C	inverse of a number	$1/x$
lcm(a,b)	R M	lowest common multiple	The same as mcm
ln(x), log(x)	R C M	logarithm natural	argument $x > 0$
max(a,b,...)	R M	maximum	$\max(-3.6, 0.5, 0.7, 1.2, 0.99) = 1.2$
min(a,b,...)	R M	minimum	$\min(13.5, 24, 1.6, 25.3) = 1.6$
mcd(a,b,...)	R M	maximun common divisor	$\text{mcm}(4346, 174) = 2$
mcm(a,b,...)	R M	minimun common multiple	$\text{mcm}(1440, 378, 1560, 72, 1650) = 21621600$
mod(a,b)	R M	modulus	$\text{mod}(29, 6) = 5$ $\text{mod}(-29, 6) = 1$
nand(a,b)	R	logic nand	returns 1 (true) if $a=1$ or $b=1$
neg(z)	C	opposit of complex	$-z$
nor(a,b)	R	logic nor	returns 1 (true) only if $a=0$ and $b=0$
not(a)	R	logic not	returns 0 (false) if $a \neq 0$, else 1
nxor(a,b)	R	logic exclusive-nor	returns 1 (true) only if $a=b$
or(a,b)	R	logic or	returns 0 (false) only if $a=0$ and $b=0$
pi	R M	Pi greek	3.141....
PolyCh(x,n)	R	Chebycev's polynomials	$\forall x$, orthog. for $-1 \leq x \leq 1$
PolyHe(x,n)	R	Hermite's polynomials	$\forall x$, orthog. for $-\infty \leq x \leq +\infty$
PolyLa(x,n)	R	Laguerre's polynomials	$\forall x$, orthog. for $0 \leq x \leq 1$
PolyLe(x,n)	R	Legendre's polynomials	$\forall x$, orthog. for $-1 \leq x \leq 1$
rad(x)	R M	radians conversion	converts radians into current unit of angle
re(x)	C	real part of complex number	
rnd(x)	R	random	returns a random number between x and 0
root(x,n)	R C M	n-th root	argument $x \geq 0$ (the same as $x^{1/n}$)
round(x,d)	R M	round a number with d decimal	$\text{round}(1.35712, 2) = 1.36$
sec(x)	R M	secant	argument (in radians) $x \neq k\pi/2$ with $k = \pm 1, \pm 2, \dots$
sech(x)	R C M	hyperbolic secant	argument $x > 1$
serie(....)	C	serie expansion of complex value z	$\text{Serie}('z/n', 'n', 1, 10, 'z', 1+3i)$
sgn(x)	R C	sign	returns 1 if $x > 0$, 0 if $x=0$, -1 if $x < 0$
sin(x)	R C M	sin	argument in radians
sinh(x)	R C M	hyperbolic sine	

Xnumbers Tutorial

sqr(x)	R C M	square root	$\text{sqr}(2) = 1.4142135623731$, also $2^{(1/2)}$
tan(x)	R C M	tangent	argument (in radians) $x \neq k\pi/2$ with $k = \pm 1, \pm 2, \dots$
tanh(x)	R C M	hyperbolic tangent	
xor(a,b)	R	logic exclusive-or	returns 1 (true) only if $a \neq b$
Bessell(x,n)	R	Bessel's function of 1st kind, nth order, mod.	$x > 0$, $n = 0, 1, 2, 3, \dots$
BesselJ(x,n)	R	Bessel's function of 1st kind, nth order	$x \geq 0$, $n = 0, 1, 2, 3, \dots$
BesselK(x,n)	R	Bessel's function of 2nd kind, nth order, mod.	$x > 0$, $n = 0, 1, 2, 3, \dots$
BesselY(x,n)	R	Bessel's function of 2nd kind, nth order	$x \geq 0$, $n = 0, 1, 2, 3, \dots$
J0(x)	R	Bessel's function of 1st kind	$x \geq 0$
Y0(x)	R	Bessel's function of 2nd kind	$x \geq 0$
I0(x)	R	Bessel's function of 1st kind, modified	$x > 0$
K0(x)	R	Bessel's function of 2nd kind, modified	$x > 0$
Si(x)	R	Sine integral	$\forall x$
Ci(x)	R	Cosine integral	$x > 0$
zeta(x)	R	Riemman's zeta function	argument $x < -1$ or $x > 1$
AiryA(x)	R	Airy function $Ai(x)$	$\forall x$, example $\text{cbr}(2) = 1.2599$, $\text{cbr}(-2) = -1.2599$
AiryB(x)	R	Airy function $Bi(x)$	argument (in radian) $x \neq k\pi/2$ with $k = \pm 1, \pm 2, \dots$
Mean(a,b,...)	R M	Arithmetic mean	$\text{mean}(8,9,12,9,7,10) = 9.1666$
Meanq(a,b,...)	R M	Quadratic mean	$\text{meanq}(8,9,12,9,7,10) = 9.300$
Meang(a,b,...)	R M	Arithmetic mean	$\text{meang}(8,9,12,9,7,10) = 9.035$
Var(a,b,...)	R M	Variance	$\text{var}(1,2,3,4,5,6,7) = 4.6666$
Varp(a,b,...)	R M	Variance pop.	$\text{varp}(1,2,3,4,5,6,7) = 4$
Stdev(a,b,...)	R M	Standard deviation	$\text{Stdev}(1,2,3,4,5,6,7) = 2.1602$
Stdevp(a,b,...)	R M	Standard deviation pop.	$\text{Stdevp}(1,2,3,4,5,6,7) = 2$
Step(x,a)	R	Haveside's step function	Returns 1 if $x \geq a$, 0 otherwise
DSBeta(x, a, b, [j])	R	Beta distribution (j=1 cumulative)	$0 < x < 1$, $a > 0$, $b > 0$,
DSBinomial(k, n, p, [j])	R	Binomial distribution (j=1 cumulative)	k integer, n integer, $0 < p < 1$
DSCauchy(x, m, s, n, [j])	R	Cauchy (generalized) distribution (j=1 cumul.)	n integer, $s > 0$
DSChi(x, r, [j])	R	Chi distribution (j=1 cumulative)	r integer, $x > 0$
DSErlang(x, k, l, [j])	R	Erlang distribution (j=1 cumulative)	k integer, $x > 0$
DSGamma(x, k, l, [j])	R	Gamma distribution (j=1 cumulative)	$x > 0$, $k > 0$, $l > 0$
DSLevy(x, l, [j])	R	Levy distribution (j=1 cumulative)	$x > 0$, $l > 0$
DSLogNormal(x, m, s, [j])	R	Log-normal distribution (j=1 cumulative)	$x > 0$, $m \geq 0$, $s > 0$
DSLogistic(x, m, s, [j])	R	Logistic distribution (j=1 cumulative)	$x > 0$, $m \geq 0$, $s > 0$
DSMaxwell(x, a, [j])	R	Maxwell-Boltzman distribution (j=1 cumulative)	$x > 0$, $a > 0$
DSMises(x, k, [j])	R	Von Mises distribution (j=1 cumulative)	$k > 0$, $-\pi < x < \pi$
DSNormal(x, m, s, [j])	R	Normal distribution (j=1 cumulative)	$s > 0$
DSPoisson(k, z, [j])	R	Poisson distribution (j=1 cumulative)	k integer, $z > 0$
DSRayleigh(x, s, [j])	R	Rayleigh distribution (j=1 cumulative)	$x > 0$, $s > 0$
DSRice(x, v, s, [j])	R	Rice distribution (j=1 cumulative)	$x > 0$, $v \geq 0$, $s > 0$
DSStudent(t, v, [j])	R	Student distribution (j=1 cumulative)	v integer degree of freedom
DSWeibull(x, k, l, [j])	R	Weibull distribution (j=1 cumulative)	$x > 0$, k integer, $l > 0$

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)

Note: the arguments separator of the functions changes automatically from "," to ";" if your system decimal separator is comma ",".

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.

E4		fx =(B4-0.51)^2+(C4-0.35)^2				
	A	B	C	D	E	F
1						
2						
3		x	y		f(x,y)	
4		0	0		0.3826	
5						
6						
7		Minimize by changing these cells			Function to minimize =(B4-0.51)^2+(C4-0.35)^2	
8						

For optimization, you can choose two different algorithms

<p><u>Downhill-Simplex</u>¹</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</p>
<p><u>Divide-Conquer 1D</u></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 $x > 0$

$$f(x) = e^{-3x} \sin(3x) + e^{-x} \cos(4x)$$

¹ The Downhill-Simplex of Nelder and Maid routine appears by the courtesy of Luis Isaac Ramos Garcia

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

$= \text{SIN}(3*B6) * \text{EXP}(-2*B6) + \text{COS}(4*B6) * \text{EXP}(-B6)$

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		$=\text{SEN}(3*B6)*\text{EXP}(-2*B6) + \text{COS}(4*B6)*\text{EXP}(-B6)$			
	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

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 with some examples¹.

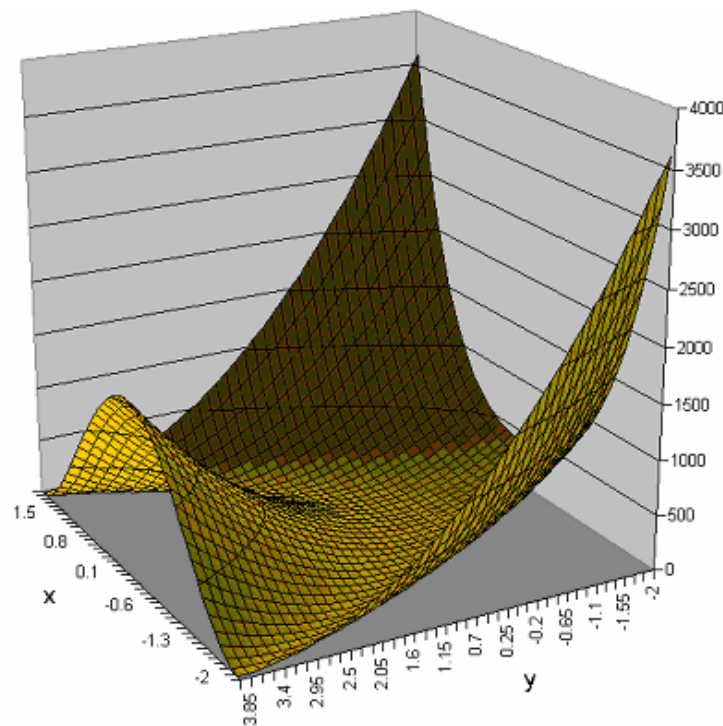
¹ The following examples are extracted from "Optimization and Nonlinear Fitting", Foxes Team, Nov. 2004

Example 1 - Rosenbrock's parabolic valley

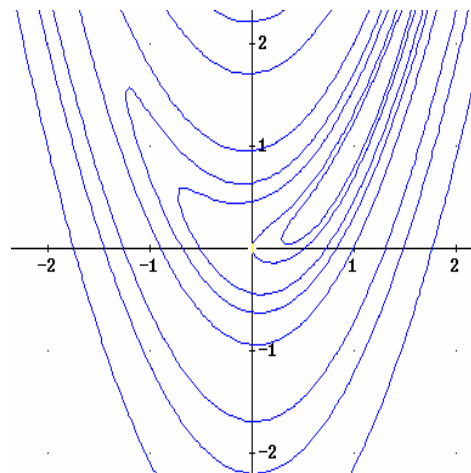
This family of test functions is well known to be a minimizing problem of high difficult

$$f(x, y) = m \cdot (y - x^2)^2 + (1 - x)^2$$

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

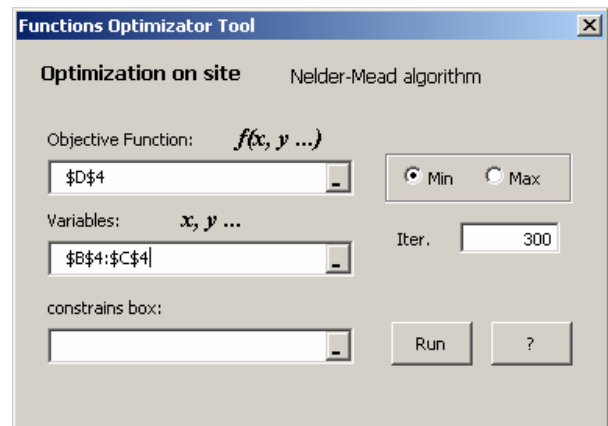
Select the cell D4 - containing the objective function - and start the macro **"Downhill-Simplex"**.

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 the macro must not change. So insert only the range B4:C4 into the variables field.

	A	B	C	D
1	Minimum searching			
2	Rosenbrock's parabolic valley			
3	m	x	y	f(x,y)
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



Leave empty the constraint input box and press "Run"

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

Xnumbers Tutorial

	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
7			
8			
9			
10			
11			
12			

Formula bar: $f_x = A2^2 + 2 \cdot A2 \cdot B2 - 4 \cdot A2 + 4 \cdot B2^2 - 10 \cdot B2 + 7$

Functions Optimizer Tool

Optimization on site Nelder-Mead algorithm

Objective Function: $f(x, y \dots)$

 ☒ Min ☐ Max

Variables: $x, y \dots$

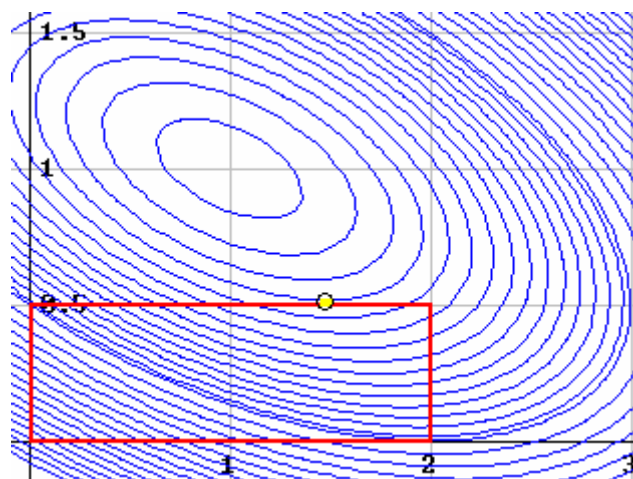
 Iter.

constrains box:

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

$$f(x, a, k) = a \cdot e^{-k \cdot x}$$

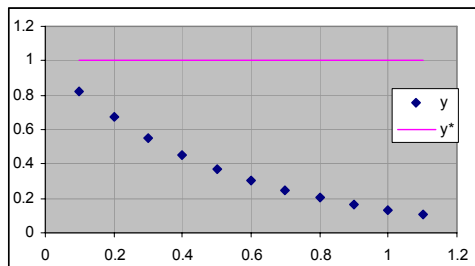
The goal of the regression is to find the best couple of parameters (a, k) that minimizes the sum of the absolute errors between the regression model and the given data set.

$$AS = \sum |y_i - f(x_i, a, k)|$$

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			

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)

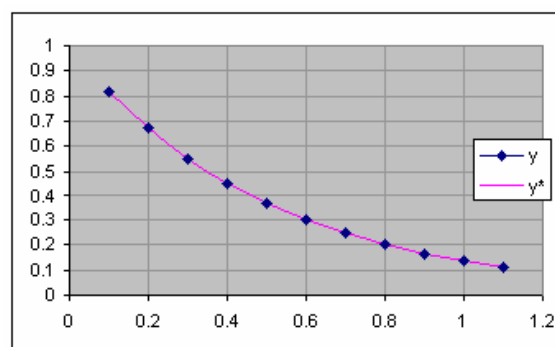
Start the Downhill-Simplex and insert the appropriate ranges: objective function = G3 and variables = E3:F3

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.



Example 4 - Optimization of Integral function

The method of optimization on-site used by Xnumbers macros and by the Excel Solver is very flexible. This is especially useful when the optimization function has a complicated form.

Assume, for example, to have to maximize the following integral respect to the parameter α

$$\int_0^1 x(\alpha - x)e^{-\alpha x} dx$$

	A	B	C
1	f(x, α)	$x*(2*\alpha-x)*\exp(-\alpha*x)$	
2	xmin	0	
3	xmax	1	
4	α	0	
5	integr	-0.333333333	
6			
7		=Integr(B3,B4,B5,A6:B6)	

The function **integr** returns the integral in the cell B5. The integration limits and the parameter α are contained in the cells B2, B3 and B4 respectively.

Start the optimization macro that you like, setting the objective cell B5 and changing the cell B4

Here, we have choose three different algorithms. They seem work very fine for this problem

Algorithm	α (max)	error
Divide-conquer	2.210360464	9.8E-09
Downhill-Simplex	2.210360527	5.4E-08
Excel Solver	2.210361438	9.6E-07

Another example. Find the maximum of the following function for $0 < \omega < 2$

$$f(\omega) = \int_0^{\pi} \sin(\omega x) dx$$

	A	B	C	D	E
1	F(x, ω)	xmin	xmax	ω	Integral
2	$\sin(\omega*x)$	0	3.14159265	0.74201929	2.276433706
3					
4	ω min	0			
5	ω max	2			

Objective cell:

Equal to: ☒ Max ☐ Min ☐

Changing the cells:

Constraints:

Here we use the Excel Solver for finding the value of the cell D2 (the variable ω) maximizing the cell E2 (the integral), subjected to the constraints $0 < \omega < 2$.

The final result $w = 0.74201929194068$ has an high precision, better 1E-8.

How to call Xnumbers functions from VBA

There are two ways for calling Xnumbers functions from other user VBA macros

- by reference method
- by Application.Run method

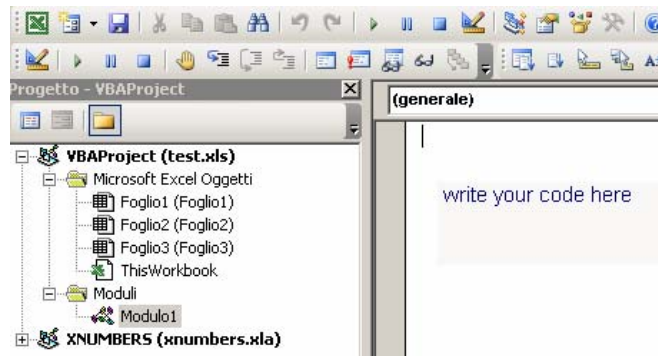
Both methods have advantages and drawbacks. Let's see.

Reference method

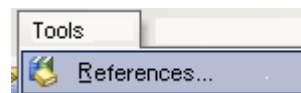
Prepare a new worksheet called, for example **test.xls**

First of all, after we have installed and loaded Xnumbers.xla in Excel, turn on the VBA editor by shift+F11 and add a new module.

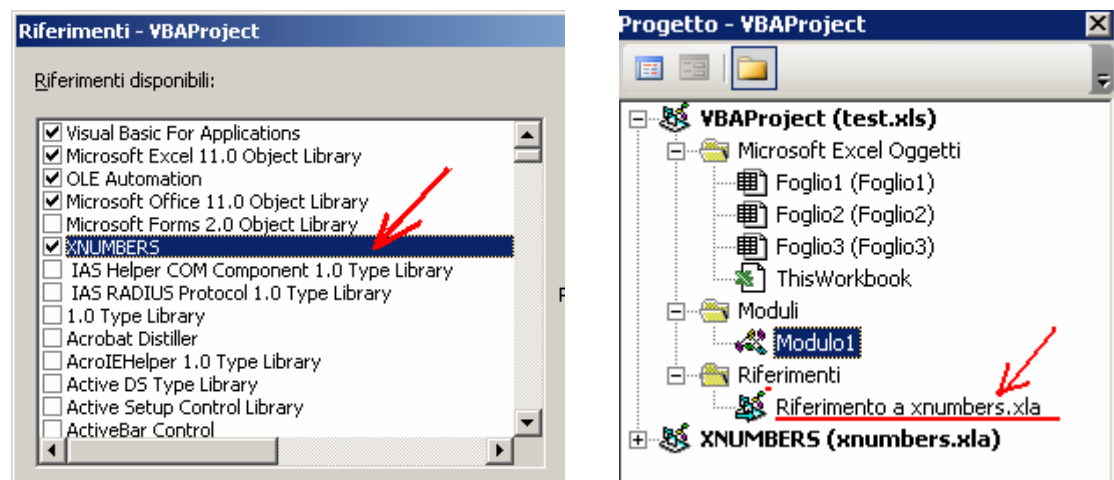
Our VBA editor should look like this. In particular we must see the **XNUMBERS (xnumbers.xla)** project and our new project **VBAProject (test.xls)**



Now we add a reference link in our **VBAProject** to **XNUMBERS** project
From the menu **tool/ references...**



Switch on the box of XNUMBERS library and click OK



Note that a references object has been added to our project
From now on all functions (except the functions declared private) of XNUMBERS are available in our macros. Of course all functions that you use in worksheet are public. But not all the functions that we can use in a worksheet can be called from VBA and vice versa.

The link that we have added is permanent. This means that when we load our test.xls file, the Xnumbers.xla will also be automatically loaded.

If you want to remove the link to Xnumbers.xla simply return to menu tools/references and deselect the box

Now prepare the following simple macro that reads two numbers from the sheet and returns the division in multiprecision

Xnumbers Tutorial

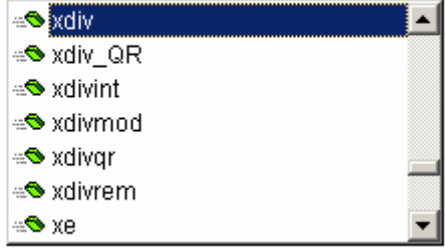
	A	B	C
1	Test multiprecision division		
2	DigitMax =>	60	
3	a =>	277465	
4	b =>	94889	
5	a / b =>		
6			

```

Sub Test()
Dim a, b, c, DgtMax
DgtMax = Range("B2")
a = Range("B3")
b = Range("B4")
c = XNUMBERS.xdiv(a, b,
DgtMax)
Range("B5") = "'" & c
End Sub

```

```
c = xnumbers.xdiv|
```



Note that the above code works also if we write the compact statement

```
c = xdiv(a, b, DgtMax)
```

The XNUMBERS prefix is not necessary; it is useful only - at the developing time - to generate automatically the list of the functions when we write the dot "." after XNUMBERS.

The result of the Test macro will be:

	A	B	C	D	E	F	G
1	Test multiprecision division						
2	DigitMax =>	60					
3	a =>	277465					
4	b =>	94889					
5	a / b =>	2.92410079145106387463246530156287873199211710524929127717649					
6							

Of course the x-functions can be nested as any other function.

For example assume now to have to compute the expression = a * Ln(b)

The code will be

```

Sub Test1()
Dim a, b, c, DgtMax
DgtMax = Range("B2")
a = Range("B3")
b = Range("B4")
c = xmult(a, xln(b, DgtMax), DgtMax)
Range("B5") = "'" & c
End Sub

```

The output of the Test1 macro will be

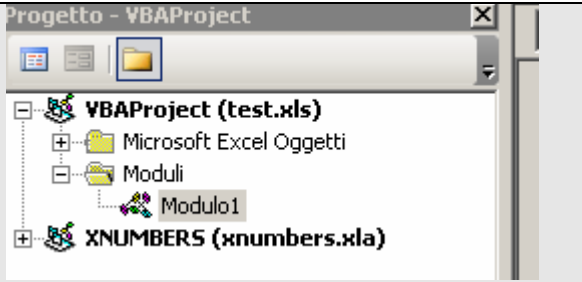
	A	B	C	D	E	F	
1	Test multiprecision formula						
2	DigitMax =>	60					
3	a =>	277465					
4	b =>	94889					
5	a*Ln(b) =>	3179877.38471700824468227743355663784462595461781151776495728					
6							

As we have seen calling x-function using the reference method is very easy and straightforward because it works as we have "virtually" copied all the Xnumbers.xla code in our project. The main drawback is that the link set in our workbook is static. This means that if we move or install the Xnumbers.xla addin in another directory, also the link must be removed and then rebuilt.

Application.Run method

A method for calling external functions that always works, independently where you have installed the addin, uses the Excel Application.Run method. Using this method we do not need to add any reference link. We have only to load the addin in Excel.

Therefore our VBA environment looks like as simple as possible and the code of the macro Test will be modified as the following

<pre>Sub test() Dim a, b, c, DgtMax DgtMax = Range("B2") a = Range("B3") b = Range("B4") With Application c = .Run("xdiv", a, b, DgtMax) End With Range("B5") = "'" & c End Sub</pre>	
---	--

The first argument of the "Run" method is the name of the function that we want to call, that is "xdiv" in our case (note the quotes around the name).

The other arguments are, in sequence, the parameters that we pass to the function xdiv.

The "Run" method returns in the variable c the result of the xdiv function.

This method is both simple and efficient. The only drawback is that it cannot accept nested functions. Thus the Test1 macro becomes a bit more tricky.

```
Sub test1()  
Dim a, b, c, DgtMax  
DgtMax = Range("B2")  
a = Range("B3")  
b = Range("B4")  
With Application  
    c = .Run("xLn", b, DgtMax)  
    c = .Run("xmult", a, c, DgtMax)  
End With  
Range("B5") = "'" & c  
End Sub
```

The final result, of course, is the same.

References & Resources

- "MSIEVE: A Library for Factoring Large Integers", Jason Papadopoulos , v1.17, March 2007
- "LAPACK -- Linear Algebra PACKage 3.0", Updated: May 31, 2000
- "EISPACK Guide, Matrix Eigensystem Routines", Smith B. T. et al. 1976
- "Numerical Analysis" F. Sheid, McGraw-Hill Book Company, New-York, 1968
- "Handbook for Autom. Computation - Linear Algebra", vol II, Wilkinson, Martin, Peterson, 1970
- "Matrix Analysis and Applied Linear Algebra", C. D. Mayer, Siam, 2000
- "Linear Algebra", J. Hefferon, Saint Michael's College, Colchester, Vermont, 2001,
- "Numerical Methods that usually work", F. S. Acton, The Mathematica Association of America, 1990
- "Analysis Numerical Methods", E. Isaacson, H. B. Keller, Wiles & Sons, 1966
- "Calculo Numérico", Neide M. B. Franco, 2002
- "Metodos Numericos" Sergio R. De Freitas, 2000
- "Numerical Mathematics in Scientific Computation", G. Dahlquist, Å. Björck, vol II.
- "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
- "Computation of Special Functions", by Shanjie Zhang and Jianming Jin - John Wiley and Sons, Inc
- "Lanczos Implementation of the Gamma Function" by Paul Godfrey, Intersil Corp, 2001
- "clsMathParser - A Class for Math Expressions Evaluation in Visual Basic", Leonardo Volpi , Michael Ruder, Thomas Zeutschler, 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)
- "A Method for Finding Real and Complex Roots of Real Polynomials With Multiple Roots", by C. Bond, July 10, 2003
- "A Robust Strategy for Finding All Real and Complex Roots of Real Polynomials", by C. Bond, April 24, 2003
- "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>)
- "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>)
- "*DE-Quadrature (Numerical Automatic Integrator) Package*", by Takuya OOURA, Copyright(C) 1996
- "*Advanced Excel for scientific data analysis*", Robert de Levie, 2004, Oxford University Press
- "*Microsoft Excel 2000 and 2003 Faults, Problems, Workarounds and Fixes*", David A. Heiser, web site <http://www.daheiser.info/excel/frontpage.html>
- "*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/~ooura>)
- "*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
- "*Gli algoritmi della crittografia a chiave pubblica*", Giovanni Fraterno, settembre 2000. <http://digilander.libero.it/crittazione>
- Bailey, D. H.; Borwein, J. M.; Calkin, N. J.; Girgensohn, R.; Luke, D. R.; and Moll, V. H. "Integer Relation Detection." §2.2 in *Experimental Mathematics in Action*. Wellesley, MA: A K Peters, pp. 29-31, 2006.
- Bailey, D. H. and Broadhurst, D. J. "Parallel Integer Relation Detection: Techniques and Applications." *Math. Comput.* 70, 1719-1736, 2001.
- Bailey, D. H. and Ferguson, H. R. P. "Numerical Results on Relations Between Numerical Constants Using a New Algorithm." *Math. Comput.* 53, 649-656, 1989.
- Bailey, D. and Plouffe, S. "Recognizing Numerical Constants." <http://www.cecm.sfu.ca/organics/papers/bailey/>.
- Borwein, J. M. and Lisonek, P. "Applications of Integer Relation Algorithms." To appear in *Disc. Math.* <http://www.cecm.sfu.ca/preprints/1997pp.html>.
- Weisstein, Eric W. "Integer Relation." From *MathWorld*--A Wolfram Web Resource. <http://mathworld.wolfram.com/IntegerRelation.html>
- Helaman R. P. Ferguson and David H. Bailey, "A Polynomial Time, Numerically Stable Integer Relation Algorithm" RNR Technical Report RNR-91-032, July 14, 1992

Credits

Software developed

Xnumbers contains code developed by the following authors that kindly contributed to this collection.

Luis Isaac Ramos Garcia	Orthogonal Polynomials and Downhill Simplex Nelder-Mead routine
Olgiard Zieba	Cubic Spline and documentation
Alfredo Álvarez Valdivia	Robust Method fitting routines
Michael J. Kozluk	Log Relative Error function, Linear regression debugging, and documentation (StRD benchmark)
Ton Jeursen	Format function, Xnumber debugging; Xnumber for Excel95, xBaseChange - Multiprecision BaseChange xCalc - multiprecision conversion function
Richard Huxtable	ChangeBase and Prime functions
Michael Ruder	MathParser improvement and debugging
Thomas Zeuschler	MathParser improvement and debugging
Lieven Dossche	Class MathParser development
Arnaud de Grammont	Complex MathParser developement
Rodrigo Farinha	MathParser improvement and debugging
Vladimir Zakharov	Installation and initialization improvement
Hans Günter	Extension of exponential integral for negative x
David A. Heiser	Weilford's algorithm for statistic standard deviation
John Jones	Modular arithmetic and Xnumber converter parser

Software translated

Xnumbers contains VB code translated from the following packages:

Takuya OOURA	DE-Quadrature (Numerical Automatic Integrator) Package
Shanjie Zhang - Jianming Jin	FORTTRAN routines for computation of Special Functions

Software used

This documentation was developed thanks to:

Microsoft Help Workshop 4	Compiler for Help-on line
Microsoft Word and Equation Editor	RTF main document and formulas
Winplot (Peanut Software by Richard Parris)	Math plots and graphical elaboration
Derive 5 (Texas Instrument)	Math graph and symbolic elaboration
MathGV 3.1 (by Greg VanMullem)	Math graphs and bitmap elaboration
Msieve 1.17	by Jason Papadopoulos. Large number factoring

Many thanks to everybody that have contributed to this document and made it more intelligible. I am especially grateful for invaluable help on many occasions to [Robert de Levie](#).

Analytical Index

A

Absolute; 28
 Adams; 211; 212
 Adams-Bashfort-Moulton; 211
 Addition; 23
 Airy functions; 252
 AiryA; 252
 AiryAD; 252
 AiryB; 252
 AiryBD; 252
 Aitken; 158; 183
 Arccos; 62
 Arcsine; 62
 Arctan; 62
 Arithmetic Mean; 40

B

Base conversion; 240
 baseChange; 240
 Bessel functions; 249
 BesselSphJ; 255
 Beta function; 247
 Bisection; 231
 Bivariate Polynomial; 88
 Brouncker-Pell Equation; 120

C

Cebychev; 78
 Change sign; 28
 Check digits; 32
 Check Integer; 112
 Check odd/even; 112
 Check Prime; 109
 Cholesky; 131
 Coefficients of Orthogonal Polynomials; 96
 Combinations; 39
 Combinations function; 248
 Complement of right angle; 63
 Complex absolute; 99
 Complex Addition; 98
 Complex ArcCos; 102
 Complex ArcSin; 102
 Complex ArcTan; 102
 Complex Complementary Error Function; 104
 Complex conjugate; 101
 Complex Cos; 102
 Complex digamma; 103
 Complex Division; 98
 Complex Error Function; 104
 Complex Exp; 101
 Complex Exponential Integral; 103
 Complex Expression Evaluation; 260

Complex Function Integration (Romberg method); 156
 Complex Gamma Function; 104
 Complex Hyperbolic Cosine; 103
 Complex Hyperbolic Sine; 102
 Complex Hyperbolic Tan; 103
 Complex inverse; 101
 Complex Inverse Hyperbolic Cos; 103
 Complex Inverse Hyperbolic Sin; 103
 Complex Inverse Hyperbolic Tan; 103
 Complex Log; 101
 Complex Logarithm Gamma Function; 104
 Complex Multiplication; 98
 Complex negative; 101
 Complex power; 100
 Complex Quadratic Equation; 105
 Complex Roots; 100
 Complex Series Evaluation; 184
 Complex Sin; 102
 Complex Subtraction; 98
 Complex Tangent; 102
 Complex Zeta Function; 104
 Constant e; 59
 Constant Ln(10); 59
 Constant Ln(2); 59
 Constant pi; 62
 Continued Fraction; 108
 Continued Fraction of Square Root; 109
 Convert extended into double; 34
 Convol; 189
 corrector; 211
 Corrector; 212; 214
 Cos; 61
 Cosine Integral Ci(x); 249
 CosIntegral; 249
 cplxabs; 99
 cplxacos; 102
 cplxacosh; 103
 cplxadd; 98
 cplxasin; 102
 cplxasinh; 103
 cplxatan; 102
 cplxatanh; 103
 cplxconj; 101
 cplxcos; 102
 cplxcosh; 103
 cplxdigamma; 103
 cplxdiv; 98
 cplxdi; 103
 cplxEquation2; 105
 cplxerf; 104
 cplxerfc; 104
 cplxeval; 260
 cplxExp; 101
 cplxgamma; 104
 cplxgammaLn; 104

cplxintegr; 156

cplxinv; 101

cplxLn; 101

cplxmult; 98

cplxneg; 101

cplxpolar; 99

cplxpow; 100

cplxrect; 99

cplxroot; 100

cplxserie; 184

cplxsin; 102

cplxsinh; 102

cplxsub; 98

cplxtan; 102

cplxatanh; 103

cplxzeta; 104

Crout; 131

cspline_coeff; 199

cspline_eval; 197

cspline_interp; 197

cspline_pre; 197

Cubic Spline 2nd derivatives; 197

Cubic Spline Coefficients; 199

cvBaseDec; 240

cvBinDec; 240

cvDecBase; 240

cvDecBin; 240

D

Data Conditioning; 53

Data Integration (Newton Cotes); 157

dBel; 240

Decibel; 240

Decimal part; 29

DFSP; 140

DFSP_INV; 140

DFT; 137; 142

DFT_INV; 139

Diff1; 234

Diff2; 235

digamma; 247

Digamma function; 247

Digit_Max; 22; 35

Digits count; 31

Digits sum; 33

DigitsAllDiff; 32

Diophantine; 119

Diophantine Equation; 119

DiophEqu; 119

Discrete 2D Fourier Transform; 141

Discrete Convolution; 189

Discrete Fourier Inverse Transform; 139

Discrete Fourier Spectrum; 140

Discrete Fourier Transform; 137

Division; 24

Double Data integration; 179

Double Exponential; 152

Double Integral; 168

Double integration function; 174

Double Series; 185

Downhill; 268

DPOLYN; 74

E

Elliptic Integrals; 252

erfun; 244

Error Function Erf(x); 244

Euler; 211

Euler constant γ ; 60

Euler-Mascheroni Constant; 245

Euler's Totient function; 121

exp_integr; 244

expn_integr; 244

Exponential; 58

Exponential any base; 58

Exponential integral Ei(x); 244

Exponential integral En(x); 244

Extended Number Check; 32

F

Factor; 116

Factorial; 39

Factorial with double-step; 39

Factorize; 113

Factorize function; 116

FFT; 137; 142

FFT_INV; 139

FFT2D; 141

FFT2D_INV; 141

Fibonacci numbers; 250

First Derivative; 234

Flip; 33

Format Extended Number; 32

Fourier; 165

Fourier_cos; 164

Fourier_cos; 166

Fourier_cos; 166

Fourier_sin; 164

fract; 107

Fract_Interp; 195

Fract_Interp_Coef; 195

FractCont; 108

FractContSqr; 109

Fresnel cosine Integral; 250

Fresnel sine Integral; 250

Fresnel_cos; 250

Fresnel_sin; 250

Function Integration (Double Exponential method); 152

Function Integration (mixed method); 154

Function Integration (Newton-Cotes); 159

Function Integration (Romberg method); 151

G

Gamma F-factor; 247

Gamma function; 245

Gamma quotient; 246

GCD; 106

Geometric Mean; 40

Grad; 235

Gradient; 235

Greatest Common Divisor; 106

H

Hermite; 78

Hessian; 236

Hessian matrix; 236

Hyperbolic Arc Cosine; 60

Hyperbolic Arc Sine; 59

Hyperbolic Arc Tangent; 60
Hyperbolic Cosine; 60
Hyperbolic Sine; 59
Hyperbolic Tangent; 60
Hypergeometric function; 251
Hypgeom; 251

I

I BesselSphY; 255
IElliptic1; 252
IElliptic2; 252
Incomplete Beta function; 248
Incomplete Gamma function; 248
Infinite integral; 177
Infinite Integration of oscillating functions; 165
Integer Division; 24
Integer part; 29
Integer polynomial; 86
Integer relation; 122
Integer Remainder; 25
Integer roots; 70
integr; 166
Integr; 154
Integr_2D; 174
Integr_fcos; 162
Integr_fsin; 162
Integr_nc; 159
Integr_ro; 151
Integr2D; 168
Integr3D; 171
Integral function; 147
Integral of sine-cosine power; 254
Integral_Inf; 177
Integration of oscillating functions (Filon formulas); 162
Integration of oscillating functions (Fourier transform); 164
IntegrData; 147
IntegrData2D; 179
IntegrDataC; 157
Interp_Mesh; 200
Interpolation with continue fraction; 195
Interpolation with Cubic Spline; 197
IntPowCos; 254
IntPowSin; 254
Inverse; 24
Inverse 2D Discrete Fourier Transform; 141
Inverse Discrete Fourier Spectrum; 140

J

Jacobian; 236
Jacobian matrix; 236

K

Kummer confluent hypergeometric functions; 253
Kummer1; 253
Kummer2; 253

L

LCM; 106
Least Common Multiple; 106
Legendre; 78

Linear Regression - Coefficients; 50
Linear Regression - Standard Deviation of Estimate; 45
Linear Regression Coefficients; 43
Linear Regression Covariance Matrix; 47
Linear Regression Evaluation; 49
Linear Regression Formulas; 46
Linear Regression Min-Max; 56
Linear Regression Statistics; 48
Linear Regression with Robust Method; 55
Log Gamma function; 246
Log Relative Error; 242
Logarithm in any base; 58
Logarithm natural (Napier's); 58

M

Macro for Double Integration; 168
Macro for Multiprecision Matrix operations; 135
Macro for Triple Integration; 171
Macro Integer relation finder; 126
Macro Mesh Fill; 201
Macro Poly Regression; 51
Macro Regression; 51
Macro Sampler; 145
Macros for optimization on site; 267
Macros X-Edit; 35
Math expression strings; 263
MathParser; 265
matrix; 135
Matrix Addition; 129
Matrix Determinant; 129
Matrix Inverse; 129
Matrix LL^T decomposition; 131
Matrix LU decomposition; 131
Matrix Modulus; 130
Matrix Multiplication; 129
Matrix Power; 130
Matrix Subtraction; 129
Maximum Common Divisor; 106
MCD; 106
MCM; 106
Mesh Interpolation 2D; 200
Minimum Common Multiple; 106
mjkLRE; 242
Modular Addition; 110
Modular Division; 110
Modular Multiplication; 110
modular power; 110
Modular Power; 110
Modular Subtraction; 110
Multiple roots; 72
Multiplication; 24
Multiprecision Base Conversion; 241
Multiprecision Excel Formula Evaluation; 261
Multiprecision Expression Evaluation; 256

N

Next Prime; 109
NextPrime; 109
Non Linear Equation Solving; 238
N-Root; 28
Numbers comparison; 31

O

Objective function; 267
 OD Linear System; 222
 ODE Implicit Predictor-Corrector; 219
 ODE Multi-Steps; 211
 ODE Predictor-Corrector 4; 217
 ODE Runge-Kutta 4; 207
 ODE_COR; 214
 ODE_PC2I; 219
 ODE_PC4; 217
 ODE_PRE; 214
 ODE_RK4; 207
 ODE_SYSL; 222
 Orthogonal polynomials; 92
 Orthogonal Polynomials; 92
 Orthogonal Polynomials evaluation; 93
 oscillating; 165

P

Partial; 89
 PECE; 214; 216
 Perfect Square; 112
 permutation; 40
 Permutations; 40
 Polar Conversion; 99
 Poly_ChebyshevT; 93
 Poly_ChebyshevU; 93
 Poly_Gegenbauer; 93
 Poly_Hermite; 93
 Poly_Jacobi; 93
 Poly_Laguerre; 93
 Poly_Legendre; 93
 Poly_Weight_ChebyshevT; 95
 Poly_Weight_ChebyshevU; 95
 Poly_Weight_Gegenbauer; 95
 Poly_Weight_Hermite; 95
 Poly_Weight_Jacobi; 95
 Poly_Weight_Laguerre; 95
 Poly_Weight_Legendre; 95
 PolyAdd; 76
 PolyBuild; 83
 PolyBuildCfx; 85
 PolyCenter; 81
 PolyDiv; 77
 PolyInt; 86
 PolyInterp; 191
 PolyInterpCoef; 191
 PolyMult; 76
 POLYN; 73
 POLYN2; 88
 Polynomial addition; 76
 Polynomial building from roots; 83
 Polynomial building with multi-precision; 85
 Polynomial center; 81
 Polynomial coefficients; 75
 Polynomial derivatives; 74
 Polynomial division quotient; 77
 Polynomial division remainder; 77
 Polynomial evaluation; 73
 Polynomial interpolation; 191
 Polynomial multiplication; 76
 Polynomial Regression - Standard Deviation of Estimates; 50
 Polynomial Regression Statistics; 50
 Polynomial roots radius; 82

Polynomial shift; 81
 Polynomial solving; 86
 Polynomial subtraction; 77
 Polynomial System of 2nd degree; 87
 Polynomial writing; 76
 PolyRadius; 82
 PolyRem; 77
 PolyShift; 81
 PolySolve; 86
 PolySub; 77
 Polyterms; 80
 PolyTerms; 75
 predictor; 211
 Predictor; 212
 Predictor; 214
 Prime; 109
 Prime Numbers Generator; 117
 Prime Test; 117
 PrimeGenerator; 117
 Product; 27
 PSLQ; 122

Q

Quadratic Mean; 40

R

Raise to power; 27
 Rational Fraction approximation; 107
 Rectangular Conversion; 99
 RegLinMM; 56
 RegLinRM; 55
 regression; 272
 Relative Rounding; 30
 Root Error Estimation; 68
 Rounding; 29

S

Scalar Product; 130
 Scientific Format; 34
 Secant; 232
 Second Derivative; 235
 Serie_trig; 186
 Serie2D_trig; 188
 Series acceleration with Δ^2 ; 183
 Series Evaluation; 182
 sign; 31
 Significant Digits count; 31
 Similarity Transform; 130
 Simplex; 268
 Sin; 61
 Sine Integral Si(x); 249
 SinIntegral; 249
 Solve Linear Equation System; 132
 SortRange; 33
 Spherical Bessel functions of integer order; 255
 Split scientific format; 34
 Square Delta Extrapolation; 133
 Square Root; 28
 Standard Deviation; 40
 Sub-Tabulation; 53
 Subtraction; 23
 Sum; 27
 sumDigits; 33
 SYSPOLY2; 87

T

Tan; 62
tanh-sinh transformation; 152
 Totient; 121
 Trigonometric double serie; 188
 Trigonometric series; 186
 Truncating; 29

U

Univariate Statistic; 42

V

Variance; 41
 Vector flip; 33
 Vector Product; 132

W

Weight of Orthogonal Polynomials; 95

X

x2pi; 62
 xabs; 28
 xacos; 62
 xacosh; 60
 xadd; 23
 xaddmod; 110
 xanglecompl; 63
 xasin; 62
 xasinh; 59
 xatan; 62
 xatanh; 60
 xBaseChange; 241
 xBeta; 247
 xBetai; 248
 xCalc; 261
 xcdbl; 34
 xcomb; 39
 xcomb_big; 248
 xcomp; 31
 xcos; 61
 xcosh; 60
 xcplxabs; 99
 xcplxadd; 98
 xcplxconj; 101
 xcplxdiv; 98
 xcplxExp; 101
 xcplxinv; 101
 xcplxLn; 101
 xcplxmult; 98
 xcplxneg; 101
 xcplxpolar; 99
 xcplxpow; 100
 xcplxrect; 99
 xcplxroot; 100
 xcplxsub; 98
 xcvcxp; 34
 xdec; 29
 xDgt; 31
 xdiv; 24
 xdivint; 24
 xdivmod; 110
 xdivrem; 25
 xe; 59

xeu; 60
 xeval; 26; 256
 xevall; 256
 xexp; 58
 xfact; 39
 xfact2; 39
 xFib; 250
 xFormat; 32
 xfrac; 107
 xFractCont; 108
 xGammaF; 247
 xGammaI; 248
 xGammaIn; 246
 xGammalog; 246
 xGammaQ; 246
 xGm; 245
 xgmean; 40
 xint; 29
 xinv; 24
 xlsInteger; 112
 xlsOdd; 112
 xlsSquare; 112
 xlsXnumber; 32
 xLn; 58
 xLn10; 59
 xLn2; 59
 xLog; 58
 xLRE; 242
 xMatAbs; 130
 xMatAdd; 129
 xMatBAB; 130
 xMatDet; 129
 xMatInv; 129
 xMatLL; 131
 xMatLU; 131
 xMatMult; 129
 xMatPow; 130
 xMatSub; 129
 xMCD; 106
 xMCM; 106
 xmean; 40
 xmult; 24
 xmultmod; 110
 xneg; 23; 28
 xpi; 62
 xpi2; 62
 xpi4; 62
 xpow; 27
 xpowmod; 110
 xProdScal; 130
 xProdVect; 132
 xqmean; 40
 xRegLinCoef; 43
 xRegLinCoef; 45
 xRegLinEval; 49
 xroot; 28
 xround; 29
 xroundr; 30
 xSerie2D; 185
 xsin; 61
 xsinh; 59
 xsplit; 34
 xsqr; 28
 xstatis; 42
 xstdev; 40
 xstdevp; 40

xsub; 23
xsubmod; 110
xsum; 27
xSYSLIN; 132
xtan; 62
xtanh; 60
xtrunc; 29
xUnformat; 32
xvar; 41

xvarp; 41

Z

Zero_bisec; 231
Zero_sec; 232
Zeros of Orthogonal Polynomials; 95
Zeta; 251
Zeta function; 251



© 2007, by Foxes Team
ITALY
Oct 2007