Xnumbers 6.0.5.2

The Excel Add-In XN.xlam (or XN.xla) has been modified from the original xnumbers.xla v.5.6 still available from the website of Leonardo Volpi, the author of Xnumbers, at:

http://digilander.libero.it/foxes/SoftwareDownload.htm

The user is expected to be familiarized with the original Xnumbers, well described in the included helpfile, or the "Xnumbers Tutorial vol.1, Oct 2007" (still available in the documents section of Leonardo's website).

Major changes to Xnumbers include:

- Support for Excel 97-2003/2007/2010/64-bit Office and Vista/Windows 7
- Can calculate to 32760 significant digits
- All exponents to ±2147000000
- Configurable "double precision" interface with Excel
- Core functions (±, ×, ÷, & the Taylor loops) are now more than 4 times faster than v.5.6 in VBA
- More than 100 additional functions (most of them supported by X-Converter)

The Add-Ins file name has been shortened to "XN" simply to reduce the number of characters stored in a cell (the Add-Ins name, along with the address of its location on your computer, is stored each time you enter a function).

Since 6.0.5.2 only one version of XN.xla and one version of XN.xlam is now necessary, working for both 32-bit and 64-bit Office. Maximum Digits_Limit can be configured up to 32760 from the Configuration Screen in the Xnumbers toolbar under X-Edit>>Configuration. It is distributed with Digits_Limit of 816.
32-bit installations will use PktSize 7 up to Digits_Limit of 630. When configuring Digits_Limit greater than 630 it will use PktSize 6 (since 6.0.5.2 only *slightly* slower than packet size 7 in 32-bit).
64-bit installations will use PktSize 8 up to Digits_Limit of 7376. When configuring Digits_Limit greater than 7376 it will use PktSize 7 (since 6.0.5.2 only *slightly* slower than packet size 8 in 64-bit).
For a complete list of changes to all versions see the included file XN60_review.txt
The VBA code is still constructed to easily compile with other packet sizes (=digits per packet) and digits limits.

Excel 97-2003: "nesting" functions with strings greater than 255 characters will not work:
xAdd(1,xDiv(1,3,254)) = #VALUE!, as the nested xDiv result is 256 characters, including the 0 & the decimal point. Each function needs to be calculated in a separate cell. Excel 2007 and 2010 work just fine for "nesting".

It also has additional code to facilitate access to all of the functions from a Visual Basic Project, so that it can easily be used in lieu of xnumbers.dll.

It also includes a much improved .chm helpfile, but can still use the original Xnumbers.hlp helpfile. Simply put it in the same folder where the Add-In is installed, and remove XN.chm (these are links for Vista and Windows7 users to download and install the .hlp help viewer from Microsoft). Excel 2007 and 2010 open XN.chm with the .chm reader, but previous versions don't, and the helpfile can be difficult to navigate, as the "Home" and "Font" buttons are missing, and the "Index" and "Search" tabs aren't valid. To open it with the .chm reader use the "Help" button from the Xnumbers Toolbar icon. If you forget to "Unblock" the downloaded .zip file before extracting, XN.chm won't work at all: right-click it, choose "Properties", then in the "General" tab click "Unblock", "Apply", and "OK".

The VBA code has been changed to comply with any file name you choose: simply right-click on the Add-In and choose "Rename". The helpfile and the .csv file will now work with or without renaming them.

When opening a spreadsheet already linked to "xnumbers.xla", you will need to "Edit Links" and "Change Source" in order to link it to XN.xla*. It is very important to note that there is currently a "bug" in some versions of Excel that does not allow you to "Change Source" of an Add-In (when you have *alot* of data), under "Edit Links", unless "Workbook Calculation" is set to "Automatic": from the 2007 Office Button (in the upper left) or 2010 File tab under "Excel Options", "Formulas", "Calculation options", or from the toolbar in "Tools", "Options", in the "Calculation" Tab. You may then need to initially use "CalculateFull" (Ctrl+Alt+F9) to update the values on your spreadsheet. If you "Change Source" on "Manual" and Excel is "(not responding)" you will need to "End Task" with Task Manager.

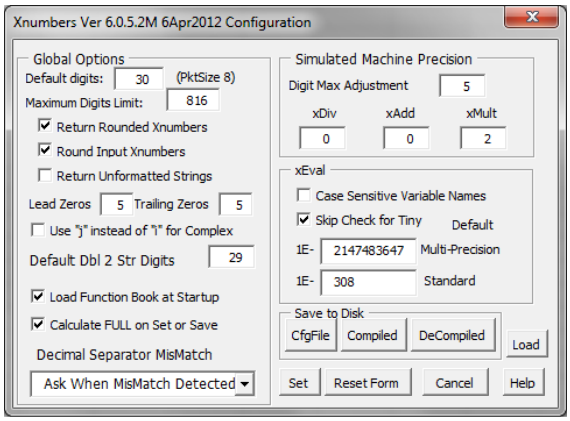
The changes to the VBA code are quite extensive, so that it is impossible to document every change. However, the code is fully accessible, so that it is easy to inspect the changes in Visual Basic or VBA.

These 4 functions are re-named, as they are in conflict with existing Excel functions (Excel since 2007 does not allow a user-defined function to have the same name as one of its own built-in functions):

BesselI	→	BesselIx
BesselJ	→	BesselJx
BesselK	→	BesselKx
BesselY	→	BesselYx

Array functions can now be entered in either direction (across or down).

The Menu for the Xnumbers icon in the toolbar has been changed. "X-Edit", "Default Digits" is now under "X-Edit", "Configuration", as there are a few other configuration options:



Default digits: 20

Extended number calculation can be easily compiled for packet size 5 thru 14 (digits per packet). Default Digit_Max is currently set to 20, as rounding to 20 digits is the best way to create the best double with xCDbl (see also x2Dbl). 32760 is the max. Excel can only display 1023 characters in a cell, but can store 32768. To see more than 1023 you can select the cell containing a formula result and "copy", then "paste values" into a blank cell, and then view the result in the formula bar by scrolling down (the button to the right of the formula bar). Alternately you can "paste" into a Microsoft Word document, a *.txt file, etc.

☑ Return Rounded Xnumbers

The internal calculation of xFunctions contain digits beyond the requested DgtMax. Check the box to return a "rounded" result: xDiv(2,3,15) = 0.666666666666667. Unchecking the box will perform the same as the original Xnumbers, in which the returned value is "truncated": xDiv(2,3,15) = 0.666666666666666

☑ Round Input Xnumbers

Extended numbers (numeric text strings) larger than the requested DgtMax will be rounded to DgtMax + the specified number of digits in the "Digit Max Adjustment" box ("5" in the example above) before being sent to a function. Unchecking this box will truncate to DgtMax + the specified number of digits. It also applies to internal calculations.

☑ Return Unformatted Strings

Will return extended numbers as unformatted full length internal calculated results (see example below)

Lead Zeros 5 Trailing Zeros 5

User can specify the maximum number of zeros to display before functions return values in scientific notation.

☑ Use "j" instead of "i" for Complex

The complex number functions can now return the entire complex number into one cell, if the function is entered into one cell, so that either "i" or "j" can be used for the suffix of the imaginary coefficient. When two cells are entered as an array, the 1st cell returns the real coefficient, and the 2nd returns the imaginary coefficient (without the suffix).

☑ Load Function Book at Startup

The "Function Book" has previously been "Loaded" upon the first use of the Xnumbers "Function manager" (under "Help" in the Xnumbers Toolbar). Loading it at startup will enable "Automatic Capitalization" of the functions according to their capitalizations in XN.csv. For example, entering a function such as =xadd(a,b) will automatically turn into =xAdd(a,b). Unchecking this box will perform as before, where during each session of Excel the capitalization of function names are "remembered" by Excel according to the way they are first entered during that session . The Add-In is distributed with the Function Book unloaded, so that other .csv's that the user might have in

their installation folder will also get loaded. Other .csv's associated with other Add-Ins will also perform "Automatic Capitalization", as long as the other Add-In is initialized by Excel *before* the Xnumbers Add-In (Excel remembers the order in which you install Add-Ins). To change the capitalization of a function change it in the 2nd column of the .csv file. To unload the Function Book use the Xnumbers Toolbar under Help>>Version, then click the "Clear" button. Since 5.0.6.2 there is also a "Load" or "Clear" button to the right of the "Save to Disk" section on the config screen.

☑ Calculate Full on Set or Save

When the configuration settings are changed with "Set" or a Save, it is probable that all of the Xnumbers functions on the spreadsheet need to be recalculated, yet Excel will not know this, so that a standard F9 (Calculate) will not suffice. If this box is unchecked it is highly recommended to manually Calculate Full (Ctrl+Alt+F9) after changing settings.

Decimal Separator MisMatch

Drop-down box provides some options for International Settings (using the comma for the decimal separator).

—— Simulated Machine Precision ——

Previously unknown to many users of Xnumbers who didn't read enough of the help documentation (including myself) is that "DgtMax" has had a dual meaning: 1) number of digits to return, and 2) number of digits to limit internal machine precision. Without the ability to increase internal machine precision, there has not been an option to return the requested number of digits (DgtMax) accurately without using xRoundR or xTruncR:

=xRoundR(xFunction(x,DgtMax+6),DgtMax) was required in all previous versions to return all of the requested digits accurately, for many functions.

The four adjustment boxes allow the user to regulate the internal machine precision, so that simply entering:

=xFunction(x,DgtMax) can return an accurate result without the need for xRoundR or xTruncR.

The first box is labeled "Digit Max Adjustment" and adds this number ("5" in the example above) to DgtMax for internal calculations. It is not recommended to use 0 or 1. 2: internally at least 2 extra digits of accuracy for the core functions, and the more difficult internally "nested" functions will never be off by more than 2 significant digits, which could be 10 to 99 units of the least significant digit. Probability of being off by a single unit is ~1 per 400, analyzed with random numbers and random Digit_Max requests for truncated results. The probability generally decreases by a factor of at least 5 for each increase in SMPadj. 3: at least 3 extra digits. The more difficult functions always accurate to 1 unit of least significance. Probability of 1 unit "error": ~1/2000. 4: at least 4 extra digits. The more difficult functions ~1/10,000 probability of the 1 unit "error". 5: at least 5 extra digits. The more difficult functions ~1/50,000 ... 14: at least 14 extra. The more difficult functions ~1/100,000,000,000 ... Consequentially, "nested" functions can now be calculated at increased DgtMax: =xFunc1(xFunc2(x,DgtMax+SMPadj),DgtMax), as xFunc1 now can use the entire input string from xFunc2 with significant digits up to "Digit Max + SMPadj".

For "xAdd" a number can be added to the requested DigitMax for all internal functions that use xAdd (or xSub). Setting this larger ensures that numbers with larger differences in exponents can get included in calculations.

For "xMult" the number refers to packets. 2 extra packets are required to retain accurate precision when returning rounded results. For truncated results 2 is also a great setting, but NumOfPackets-1 can be used to *absolutely* ensure correct truncation (in *extremely* rare circumstances involving a slew of 9's).

For "xDiv" using "0" still ensures at least 1 extra digit for rounding, as "1" is now added when "Round" is selected.

There is also the function xEval with the option to use ☑ Case Sensitive Variable Names: check this box to distinguish between variables "a" and "A", otherwise they are the same. Also with the option to ☑ Skip Check for Tiny, or to specify your own default for "Tiny". Two more [optional parameters] have been added to the function, after [Angle], so that it now reads: xEval(Formula, [Var], [DgtMax], [Angle], [Tiny] , [IntSwapFix])

If the default is set to 1E-[3000] for multi-precision, this means that if the resulting answer is less than 1E−3000, the function will return "0" (zero). The boxes allow you to change the default separately for [ ] Multi-Precision and [ ] Standard (just as before, xEval calculates in standard double precision when DgtMax is entered as 0).

Entering [Tiny] into the function the same as [DgtMax] performs the same as the original Xnumbers.

Also in the configuration screen is the option to "Set" your parameters (temporarily until you close Excel), or to Save them by one of the three methods in the "Save to Disk" section, so that your choices remain the defaults when Excel is opened the next time. Using the "CfgFile" button does not overwrite the add-in, but instead creates a file XN_Cfg.ini in the installation folder for the configuration settings, as some installations of Office 97 and 2000 might not properly overwrite it. "Compiled" overwrites the Add-In, and is the recommended way of saving your settings, as it is saved on *your* machine in *your* environment. It also allows the Function Book to remain loaded, so that it is faster to install each time Excel is opened. The Add-Ins description "Drive:\Installation Folder\XN.xla*" will also be updated in Excels list of Add-Ins. "Reset Form" resets all options to the original distributed defaults.

The "Decompiled" button is a quick method of creating a decompiled version for use on a different platform - for VBA programmers making changes to the add-in. It also replaces the "Create .XLA" button in versions previous to 6.0.5.2. It will properly create an .xla Add-In (for Excel 97-2003) from Excel 2007 or 2010, if you want to change some code in the .xlam and also save an .xla to use in an earlier version. This is provided because some of the code since 2007 is not recognized in earlier versions. The "Title" of the .xla will be appended with an "A" instead of an "M". This properties title is what appears in Excels "Add-Ins" menu. Excel isn't happy if you have two Add-Ins listed with the same title! The add-in is distributed *fully* DeCompiled from the program at:
http://cpap.com.br/orlando/VBADecompilerMore.asp?IdC=Help

Default Dbl 2 Str Digits 29

There are some new functions that convert a number into a text string. They are the reverse of the funtion xCDbl( ), which converts a string into a number. The box labeled "Default Dbl 2 Str Digits" allows you to configure the default function that gets used when you send an Excel number (a "double") to an extended numbers function. To obtain the best performance of *all* functions (including standard precision) at the intended accuracy of version 6.0, it is highly recommended to use a default Dbl 2 Str Digits of 20 or greater. Using 0, 15, or 16 does not uniquely distinguish all doubles, so does not allow the code to perform optimally. Internal functions that use VBA's CDbl (used by xCDbl) recognize digits up to and including the 20th. 1 through 14 are not valid entries.

Entering 0 (zero) uses the new function vCStr(x), which is VBA's convert to string function CStr, returning a string up to 15 digits, and is the same default used in the original Xnumbers.

Entering 15 - 28 uses the new function dCStr(x, [Digits]), which uses VBA's "Variant Decimal" 29 digit data type, rounded to a good 15 - 28 digit representation of a double. Default [Digits] for dCStr is the configured setting.

Entering 29 uses qCStr(x). The 29th Variant Decimal digit is not always correct, but it is a faster function than dCStr.

Using 15 more closely emulates Excels "standard" 15 digit interface, and is slightly different than vCStr. To *exactly* use Excels 15 digit representation, one should convert the double to a text string with Excel before sending it to an extended numbers function, and leave the configuration setting greater than 16 for optimum performance. Such as with Excels function COMPLEX(x,0), accomplishing the task in a generic manner by the use of 0 (zero) for the imaginary coefficient. Strangely enough, this 15 digit string will not always convert back into the same number with VBA's convert to double function (CDbl, which is used by xCDbl). Only about 6% of "doubles" (numbers) that can be calculated in a cell will convert to a 15 digit string and then back into the same double.

Using 16 returns a string up to 16 digits. It will not always convert back into the same double, but much more often than 15 (it takes 17 to uniquely distinguish all doubles). It will allow you to produce, from Excel, the same 16 digit results that perhaps you have seen in data files given in "double precision".

Entering 30 through 767 uses the new function xCStr(x, [Digits]). It is much slower to calculate, but can return a string as the exact value of an Excel calculated number (the "double" in a cell). 767 is the maximum number of significant digits needed to exactly represent any number in "double precision". Default [Digits] for xCStr is 767.

Check: xDgtS(xCStr(xCDbl(xMult("9007199254740991",xPow2(-1074,1000),1000)))) = 767

For example, Excel allows you to type in 15 significant digits.
4503599627370510 is a 16 digit integer with 15 significant digits.
4503599627370520 is the next number greater in value that you can type into a cell. However, Excel is able to store each of the 9 integers between the two, as calculated values. This applies to doubles between 2^{52} and 2^{53}, where the increment between doubles is 1 (or 2^{0}). Between 2^{51} and 2^{52} the increment is ½ (or 2^{-1}); between 2^{50} and 2^{51} the increment is ¼ (or 2^{-2}); etc. For instance,

enter into cell $A$1: =4503599627370510+1

vCStr($A$1) = 4503599627370510, the same as dCStr($A$1, 15), but
xCStr($A$1) = 4503599627370511, the same as dCStr($A$1, 16 or greater)

Also, strange as it seems, Excel does not store the exact value 1.1
If 1.1 is entered into cell $A$2, the value of the number in that cell can be expressed as

$$\frac{4953959590107546}{2^{52}} \quad \text{, or } \quad 4953959590107546 \times 2^{-52}$$

Its exact value is obtained by xCStr:

xCStr($A$2) = 1.1000000000000000888178419700125232338905334472656 25
dCStr($A$2, 24) = 1.100000000000000008881784
dCStr($A$2, 21) = 1.10000000000000008882
dCStr($A$2, 17) = 1.1000000000000001
dCStr($A$2, 16) = 1.1
dCStr($A$2, 15) = 1.1

The next number greater in value that Excel can store can be obtained by typing into cell $A$3: =$A$2+1/2^52
This applies to doubles between 1 (or 2^{0}) and 2 (or 2^{1}), where the increment between doubles is $\frac{1}{2^{52}}$ (or 2^{-52})
The exact decimal value in that cell is obtained by xCStr:

xCStr ($A$3) = 1.1000000000000003108624468950438313186168670654296875
dCStr($A$3, 28) = 1.1000000000000003108624468 95
dCStr($A$3, 17) = 1.1000000000000003
dCStr($A$3, 16) = 1.1
dCStr($A$3, 15) = 1.1

The convert to string functions are an important difference in this modified version of Xnumbers. Depending on your default setting, you could be quite surprised when using numbers (doubles), as depicted in the examples above.
Note: if you have existing spreadsheets that pass doubles to xFunctions, and need them to perform the same as the original v.5.6 *without* first converting the doubles into strings with the Xnumbers function vCStr(x) or the faster Excel function COMPLEX(x,0), use the inadvisable default setting of "0".

There is also a new macro in the X-Edit menu of the Xnumbers toolbar, "Double 2 Xnum".
It is the reverse of "Xnum 2 Double" (previously named "Double conv."), and works in the same manner.

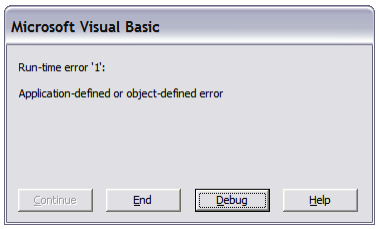
The Add-In stores constants to Digits_Limit (× 2 for $\pi/2$ and $\pi/4$) + xBASE (xBASE = PacketSize)
Note: xDgt( ) cannot recognize more than Digits_Limit number of significant digits on the spreadsheet. To confirm that you have a string longer than Digits_Limit use xDgtS( ), or the Excel function LEN( ).

Constants changed to return signif digits rather than decimal places (the same as all other functions use of DgtMax).

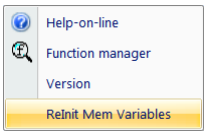
Constants are now stored in memory, and they get "loaded up" into memory variables when the Add-In is initialized:

xPi = π , x2Pi = 2π , xPi2 = $\pi/2$, xPi4 = $\pi/4$, xe = e , xEu = γ , xLn2 = Ln(2) , xLn10 = Ln(10) , xRad5 = $\sqrt{5}$, xRad12 = $\sqrt{12}$

Prior to v.6.0.4.4, if you got an "end, debug" error message (from one of the macros) and chose "End",



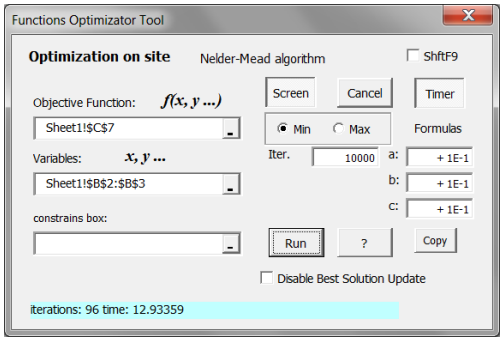
or you "reset" in VBA, the memory variables needed manual re-loading. One way to do this is to uncheck the Add-In (unload it), and then re-check it. A quicker method is provided from the "Help" menu in the Xnumbers tool-bar:



Click on "ReInit Mem Variables" to re-initialize the memory variables.

The status of the memory variables are now checked after each "calculate" (F9), or when the Toolbar icon is clicked.

The Nelder-Mead macro (Xnumbers toolbar under "Macros", "Function max/min", "Downhill-Simplex") has been changed. It now allows 128 variables, rather than only 9. It also now works with "manual calculation" turned on (original version 5.6 needs calculation set to "automatic"):



The user-form has been changed significantly. The checkbox labeled "ShftF9" will direct the macro to calculate with Shift+F9 (calculates the active worksheet only) instead of a regular F9 (calculates all open workbooks).

The "Screen" button toggles between screen updating on and off. Quicker calculating with screen updating off.

"Cancel" button allows you to stop the macro before the specified number of iterations ("Iter." box) is completed.

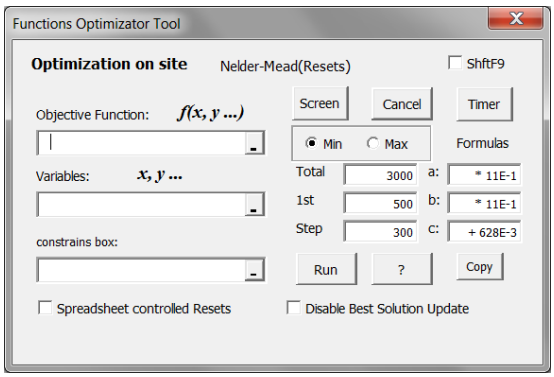
"Timer" button will record the length of time (in seconds) the macro takes to complete its task. Number of iterations is also recorded, and a running total is displayed during operation.

Boxes labeled "a:", "b:", and "c:" contain the formula that the macro uses to find better values. +0.1 is the standard. For example, if you don't want it to search too wide a range you might change the formula to +0.000001. Any formula can be entered.

The 3 different boxes allow you to have 3 different variable types, which need to be put into 3 separate rows or columns. a, b, and c are positioned on the spreadsheet left-to-right for separate columns, and top-to-bottom for rows. For 1 variable type any matrix of variables is fine, but it will still use all 3 formulae, so all three should be the same.

The "Copy" button copies the formula in box "a:", to "b:" and "c:", for convenience.

For more difficult problems, “Downhill-Simplex/Resets” restarts the algorithm (in the example below) after an initial 500 iterations (the box labeled “1st”), then restarts after 500 + 300 (from the box labeled “Step”) = 800 iterations, then again after 800 + 300 = 1100 iterations, then after 1100 + 300 = 1400 iterations, etc., until the total number of all iterations is 15000 or greater (from the box labeled “Total”).



When the “Spreadsheet controlled Resets” checkbox is checked, the macro reads the value in the cell to the left of the "Objective Function" cell, and puts the iteration count in the cell to the right. The cell to the left needs to contain an IF statement that returns 1 to request a reset. It needs to return 0 (or False) to continue optimizing. Most importantly the IF statement needs to detect that the the iteration count (in the cell to the right) has a value of 0 (which it *will* after each reset), and the IF needs to return 0 (or False), to continue the optimization. A value of -1 (in the IF to the left) means STOP optimizing, and returns the best solution that it has found to the spreadsheet. This same "best" solution will now be returned when the "Cancel" button is clicked, or when it reaches the specified total iterations. ALL previous versions of Xnumbers simply left the last attempt on the spreadsheet when the total iteration count was reached. Checking the box "Disable Best Solution Update" performs as before, returning the current solution. The "Total" box means the total of ALL iterations.
Note: to change default settings of boxes open Forms>UserFormOptim(Code)>Private Sub UserForm_Activate()

“Macros”, “Regression”, “Polynomial” is slightly changed. Some of the previous versions of Xnumbers worked with some parameters and not with others. The last version (5.6) did not work when requesting a degree 7 polynomial from 8 data points (whereas v.5.4 did!). Bug fix to one of the subroutines since v.6.0.4 should now enable all possible parameters.

xMin() & xMax() changed to analyze multiple ranges (matrices) rather than only a vector (single row or column).

xExpa(x, [a], [DgtMax]) = a^x has been changed to *allow* a result if 'a' is negative and 'x' is not an integer. It will return a leading space, alerting the user that it is only valid if the denominator of 'x' is odd: xExpa(1/3,−8)= −2. The Excel function IF(LEFT(result,1)=" ",TRUE) can test if the 1st character of the result is a space. The new function xOddDen(x, [DgtMax]) can test if the denominator of 'x' is odd (TRUE) or even (FALSE).

New function xTruncR(x, [SigDgts]) truncates an extended number to the optional parameter “significant digits”.

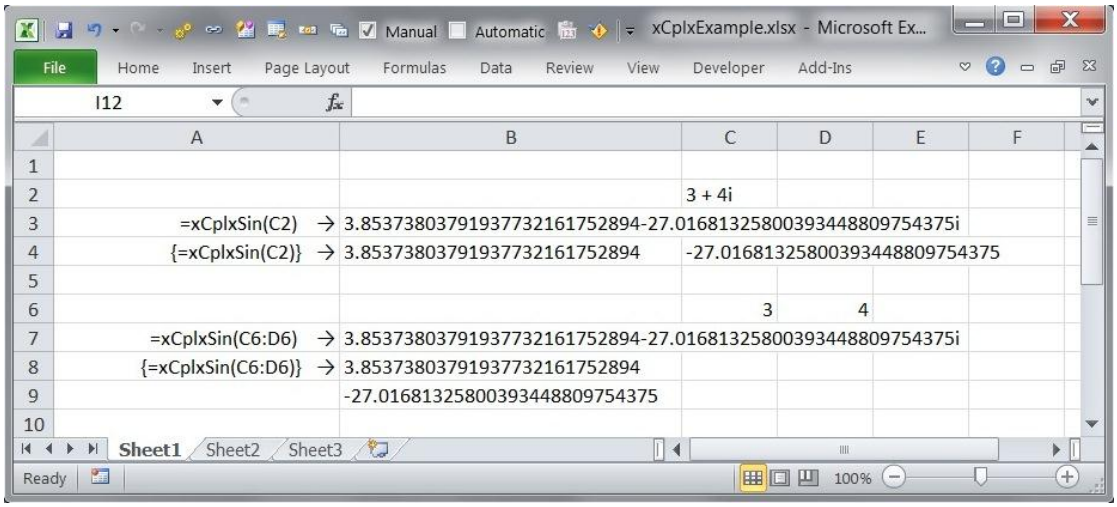
New function vRoundR(x, [SigDgts]), similar to xRoundR but uses VBA’s round function, which performs differently when you are rounding to one less digit than the number of digits passed, and the last digit is a 5. It rounds *up* if the second to last is odd, and rounds *down* if the second to last is even. For example, rounding 1.635 to 3 significant digits returns 1.64 (rounds up because the 3 is odd). But rounding 1.625 to 3 significant digits returns 1.62 (rounds down because the 2 is even). vRound(x, [DecDgts]) rounds to number of DecDgts decimal places for doubles

New function xAtan2(y,x,[DgtMax]) providing the extended accuracy of Xnumbers. It is similar to Excels ATAN2(x,y). Caution: x and y are reversed.

All of the complex number functions can now return the entire complex number into one cell, the same as Excels complex number functions, so that Xnumbers can read Excels results and vice versa. Previous versions of Xnumbers returned only the real coefficient when entered into a single cell. The complex functions can also still be entered as a two cell array, returning the real coefficient into the first cell and the imaginary coefficient into the second.
To return the real coefficient of a complex extended number there is a new function xReal( ), which performs the same as Excels function IMREAL( ). xImag( ) returns the imaginary coefficient, as does Excels IMAGINARY( ), and xCplx( ) builds a complex number from two cells, the same as Excels function COMPLEX( ).

Many complex trig functions have been added to enable calculation with extended numbers, and can be entered in a single cell, returning a complex number, or as a two-cell array in either direction, across or down. For example, xCplxSin(z,[DgtMax]) is shown below entered:

In the single cell B3 with its parameter "z" the single cell C2
In cells B4:C4 as an array going across, "z" the single cell C2
In the single cell B7 with its parameter "z" the two cells C6:D6
In cells B8:B9 as an array going down, "z" the two cells C6:D6



There are the new functions xDiff1( ), xDiff2( ), xJacobian( ), and xGrad( ) that return extended numbers (strings), and simply add an optional parameter [DgtMax] to the existing functions Diff1( ), Diff2( ), Jacobian( ), and Grad( ), which return numbers (doubles). Since v.6.0.4 these again use 5 point equations for the derivative estimates.

Adds ability of xBaseChange( ) to do a maximum of 55 decimal places rather than 50.

Fract( ) and xFract( ) will return "numerator/denominator" when entered in one cell (readable by xCalc or xEval).

FractCont( ) and xFractCont( ) will return the continued fraction in brackets $[a_0;a_1,a_2,\ldots,a_n]$ when entered into one cell. Enter optional parameter [n] to truncate. When one row or column is selected, the first cell will return the continued fraction in brackets and the subsequent cells will return "numerator/denominator". When entered in two rows or columns, the first will display "a" values, and the second "numerator/denominator". As before, when entering 3 rows (or now columns), the first will display "a" values, the second "numerator" and the third "denominator". xFractCont can use optional parameters [StopMethod] and [StopValue] to truncate. StopMethod "0" is the default, and an improved algorithm since v.6.0.4 accurately returns the *entire* continued fraction of a number *x* with digits < Digits_Limit. "1" uses *x* – num/den for StopValue. "2" uses rDgMat(*x*, num/den). "3" uses length of numerator + length of denominator. "4" uses *x* × den − num.

There are two new macros in the X-Edit menu of the Xnumbers toolbar:

Paste Special — 'Paste Values': into each cell with a formula in selected range, and pastes each cell's formula into a comment. 'UnPasteValues' puts the comment formulas back in cells. 'RePasteValues' performs UnPasteValues, calculates formula, then re-performs PasteValues. 'RePaste + Calc' performs UnPasteValues, calculates workbooks (F9), then re-performs PasteValues.

'Paste Exact': Pastes the exact cell contents of the Source Range to the Destination Range. Cell references in formulas remain exactly the same. When one entire array is selected can change size of the array. Selecting upper left cell of array also indicates the entire array

Trace Range — Traces 'Dependent' Arrows or Traces 'Precedent' Arrows from selected range of cells. 'Remove Both' or remove either type arrow from selected range of cells.

This is the list of additional functions thru ver. 6.0.4.4, including existing functions with new additional parameters. Newer functions are listed in XN60_reveiw.txt. These will not appear in Xnumbers.hlp, but a short description exists in Excels function manager (the *fx* button on the left of the formula bar).
Hint: for faster use of a function manager enter the NON-optional parameters *last* (they are displayed in **Bold** type)

cplxFact(z)	Factorial of the complex number z
Cstr2logD(x)	Returns Log of extended number x (base 10), as a double
ClogD2str(x)	Reverse of Cstr2logD. Returns 10^x (x is a double) as a 15 digit string
CheckFormula(Formula, [Digit_Max], [AngleSet])	1 × 3 array returns formula as used by Multiprecision MathParser. 2nd cell returns ordered list of variable names. 3rd cell returns any error message
dCStr(x, [Digits])	Convert to string function uses VBA's "Variant Decimal" data type to convert a double (Excel number) into a numeric text string up to 28 digits. Default [Digits] for dCStr is the configured Default Dbl 2 Str setting. [Digits] of 29 uses qCStr. [Digits] greater than 29 uses xCStr
DgMat(x, y, [MatchValue])	returns the number of matching significant digits of x and y. If x and y are an exact match, by default returns 20000, but can be specified by [MatchValue]
Dpoly(x, Coef, ord)	existing private function made available. Calculates derivative of a polynomial
fDgMat(x, y, [M])	similar to rDgMat, also returns fractional (decimal) portion of match. If x and y are an exact match, default returns xDgtS(x), but can be specified by [M]. If x and y differ by more than an exponent returns -ABS(Log(x)-Log(y)), unless M is negative it returns zero, and uses Abs(M) for an exact match
FractCont(x, [n])	adds an additional optional parameter n to the existing function FractCont(x), allowing you to enter the function into one cell resulting as $[a_0;a_1,a_2,\ldots,a_n]$ (see text above)
GetDecr(x)	returns approximate 29 digit string of (Decreasing) increment for the previous (smaller) double. To return as double use vGetDecr
GetDerivCoefs([Order], [NumofPoints], [Offset])	Returns the coefficients for the derivative functions as a 2 × (2+NumofPoints) array, with LCD and Divisor as the first 2 returns
GetDerivD([Var_Value], [Digit_Max], [Order], [NumOfPoints], [Offset],[MaxPrec])	Returns the initial internal machine precision that will be used for the given parameters in the derivative functions. If [MaxPrecision] is missing in the derivative functions it is allowed up to 3 × GetDerivD
GetDerivExpectedDigits ([Var_Value], [Digit_Max], [Order], [NumOfPoints], [Offset],[MaxPrec])	The reverse of GetDerivD. Returns the expected number of correct digits for the derivative functions with the given parameters. If [MaxPrecision] is missing in the derivative functions it is allowed up to 3 × GetDerivD
GetDigitsLimit( )	returns Digits_Limit, the largest allowed DgtMax for your compiled version
GetIncr(x)	returns approximate 29 digit string of (Increasing) increment for the next (larger) double. To return as double use vGetIncr
GetNextDouble(x)	returns the next double (further from zero than x)
GetPrevDouble(x)	returns the previous double (closer to zero than x)
GetxBASE( )	returns xBASE, the number of digits per packet, or PacketSize (6 for v.6.0-6, 7 for v.6.0-7, 8 for v.6.0-8, 13 for v.6.0-3)
GetXnArgSep( )	Returns the argument separator being used in VBA
GetXnCaseSen( )	Returns True/False, the Configured default for Case Sensitive Variable Names
GetXnConfigStatus( )	2 × 19 array function returns all 18 Configuration settings, with labels as their Public variable names (listed in the XnumbPrivate module), plus Digits_Limit
GetXnDecSep( )	Returns the decimal separator being used in VBA
GetXnDefaultDigits( )	returns the configured default DigitMax
GetXnDefCStr( )	Returns the current Configuration setting for Default Dbl 2 Str Digits. 0=vCStr, 15-28=dCStr, 29-Digits_Limit=xCStr

GetXnLeadZeros( )	Returns the Configured default max number of leading zeros returned with numbers between -1 and 1. If more exist extended numbers are returned in scientific notation. To display more zeros than DgtMax use the xFmt function
GetXnMilSep( )	Returns the thousands separator being used in VBA
GetXnTrailZeros( )	Returns the Configured default maximum number of trailing zeros that get returned with integers. If more exist extended numbers are returned in scientific notation. To display more 0's than DigitMax use the xFmt function
GetXnUseXroundIN( )	Returns TRUE or FALSE, the Configured default setting for using rounded input. (IF extended number input strings are larger than DigitMax)
GetXnUseXroundOUT( )	Returns TRUE or FALSE, the Configured default setting for using rounded output of extended number strings
GetXNxAddAdj( )	Returns the Configured default number of extra digits to use for internal calculation of xAdd (and xSub). Setting this larger ensures that numbers with larger differences in exponents get included in calculations
GetXNxDivAdj( )	Returns the Configured default extra digits for internal calculation of xDiv
GetXNxMultAdj( )	Returns the Configured default number of extra packets (multiples of xBASE) to use for internal calculation of xMult. '2' is required to ensure accuracy
GetXnSMPadj( )	Configured default number of extra digits (beyond DigitMax) used for internal calculation of most functions
gGTd(x)	Returns the closest double Greater Than x. gGTd(x) > x for all numbers, positive and negative
gLTd(x)	Returns the closest double Less Than x. gLTd(x) < x for all numbers, positive and negative
iFmt(x, [NumIntDigits])	formats an extended number, real or complex, as an integer and an exponent
LRE (Q, C, [NoSD])	the same as the existing mjkLRE( ), but with optional NoSD (default 15)
MatT(Mat)	Matrix Transpose is an existing private function made available. (Mat) is a matrix n × m. Array function transposes to m × n
pFmt(x)	Formats a numeric string, or a number as a string using DefaultCStr, with a 'plus' sign in front of a positive number, or leaves a negative number with its 'minus' sign. Useful for concatenation of formula strings
Poly(x, Coef)	existing private function. Calculates a polynomial with Coef array, at value 'x'
PrevPrime(num)	Returns the first prime number less than 'num', for all primes less than 2^53
rDgMat (x, y, [MatchValue], [DgtMax])	Returns the relative number of matching significant digits of x and y. If x and y are an exact match, by default returns 20000, but can be specified by [MatchValue]. Optional [DgtMax] specifies digits to compare
s2Dbl(x)	one of the Xnum 2 Double options. Strangely enough, VBA's CDbl (convert to double) function can sometimes produce doubles from numeric text strings in an incorrect mathematical sequence. This only occurs near the "transition" between doubles. s2DBl straightens the sequence by analyzing the 20th digit. This anomaly apparently does not exist in 64-bit Office 2010
sCStr( )	Returns current Configuration# setting for 'Default Dbl 2 Str Digits' If Config# is 0 returns 'vCStr' ; If 15 thru 28 returns 'dCStr(Config#)' If greater than 28 returns 'xCStr(Config#)'
sFmt(x)	converts an extended number, real or complex, to scientific format
v2Dbl(x)	one of the Xnum 2 Double options. Uses VBA's Val (value) function rather than its CDbl function, very rarely giving a different result
vAdd(x, y)	Uses VBA's addition for x+y, where x and y are doubles. Enables manipulation of doubles that are too close to each other for EXCEL
vCStr(x)	Convert Double (x) to String using VBA's CStr function (15 digit limit). Can sometimes differ from Excel's 15 digit representation and dCStr(x, 15)

vDiv(x, y)	uses VBA's division for x/y, where x and y are doubles. Enables manipulation of doubles smaller than 2.2251E-308
vExponent(x)	same as xExponent , but returns a number (double) instead of a string (text)
vGetDecr(x)	Returns a double: the (Decreasing) Increment for Previous (smaller) Double x + GetDecr(x) < x for positive x. x + GetDecr(x) > x for negative x. GetDecr(0) = -4.94E-324 , the smallest negative Increment: =xCDbl(xNeg(xPow2(-1074)))
vGetIncr(x)	Returns a double: the Increment for the Next (larger) Double. x + GetIncr(x) > x for positive numbers. x + GetIncr(x) < x for negative numbers. GetIncr(0) = 4.94E-324 , the smallest positive Increment: =xCDbl(xPow2(-1074))
vIntLog10(x)	where x is a double, correctly determines Int(Log(x)/Log(10))
vIntLog2(x)	where x is a double, correctly determines Int(Log(x)/Log(2)) (see text above)
vIntMod(N, D)	Returns the remainder of the division N / D in double precision
vMult(x, y)	uses VBA's multiplication for x*y, where x and y are doubles. Enables manipulation of doubles smaller than 2.2251E-308
vRoundR(x, [SigDgts])	rounds to significant digits with VBA's rounding function. Can be different results than xRoundR (see text above)
vSub(x, y)	Uses VBA's subtraction for x−y, where x and y are doubles. Enables manipulation of doubles that are too close to each other for EXCEL
x2Dbl(x)	one of the Xnum 2 Double options in X-Edit menu. Absolutely ensures that the double produced is *the* mathematically closest double to the numeric string being passed, x. Very rarely differs from other Xnum 2 Double options only near the "transitions" between doubles. Slower but more accurate than xCDbl
xAddR (x, y, [DgtMax])	internal function the same as xAdd, but 10% faster in VBA Projects. Can also be called from the spreadsheet, where it is 3% faster, but returns numbers in 'Raw' format: as an integer and an unformatted exponent
xAdjPi(x, [DgtMax])	adjusts an extended number given in radians to a value between -Pi and +Pi
xAdj2Pi(x, [DgtMax])	adjusts an extended number given in radians to a value between 0 and +2Pi
xAlog(x, [DgtMax])	Multiprecision antilogarithm in base 10. xAlog(x) = 10^x
xAtan2(y, x, [DgtMax])	the same as Excels ATAN2(x,y) function, but with extended precision. Caution: x and y are reversed
xAveDev(a, [DgtMax])	the same as Excels AVEDEV( ) function, but with extended precision
xBinomial(k,n,p,[Dtype],[DgtMax])	the same as the existing function DSBinomial( ), but with extended precision
xCat([Method], a1, [a2], [a3],…)	the same as Excels CONCATENATE( ) function, but with extended precision. Adds the additional initial parameter "Method", a number, which if is missing uses the configured Default Dbl 2 Str Digits to convert doubles to strings
xCeil(x, signif)	the same as Excels CEILING( ) function, but with extended precision
xClip(T, Floor, Ceil)	Returns T if Floor < T < Ceil; Floor if T <= Floor; Ceil if T >= Ceil
xComp1(x)	compares Abs(x) to 1. Same as xComp(xAbs(x), 1)
xCorrel(ax, ay, [DgtMax])	the same as Excels CORREL( ) function, but with extended precision
xCoVar(ax, ay, [DgtMax])	the same as Excels COVAR( ) function, but with extended precision
xCplx(r, i, [s])	the same as Excels COMPLEX( ) function. The optional parameter "s" allows you to select a character other than the configured default, "i" or "j", for the suffix of the imaginary coefficient. Don't forget the "quotes" for text argument
xCplxAcos(z, [DgtMax])	the same as the existing CplxAcos( ) function, but with extended precision
xCplxAcosh(z, [DgtMax])	the same as the existing CplxAcosh( ) function, but with extended precision
xCplxArg(z, [DgtMax])	the same as Excels IMARGUMENT( ) function, but with extended precision

xCplxAsin(z, [DgtMax])	the same as the existing CplxAsin( ) function, but with extended precision
xCplxAsinh(z, [DgtMax])	the same as the existing CplxAsinh( ) function, but with extended precision
xCplxAtan(z, [DgtMax])	the same as the existing CplxAtan( ) function, but with extended precision
xCplxAtanh(z, [DgtMax])	the same as the existing CplxAtanh( ) function, but with extended precision
xCplxCos(z, [DgtMax])	the same as the existing CplxCos( ) function, but with extended precision
xCplxCosh(z, [DgtMax])	the same as the existing cplxCosh( ) function, but with extended precision
xCplxLog(z, [BASEn], [DgtMax])	similar to the existing function xLog( ), but accepts a complex number
xCplxLog10(z, [DgtMax])	the same as Excels IMLOG10( ) function, but with extended precision
xCplxLog2(z, [DgtMax])	the same as Excels IMLOG2( ) function, but with extended precision
xCplxSin(z, [DgtMax])	the same as the existing CplxSin( ) function, but with extended precision
xCplxSinh(z, [DgtMax])	the same as the existing CplxSinh( ) function, but with extended precision
xCplxSqr(z, [DgtMax])	the same as Excels IMSQRT( ) function, but with extended precision
xCplxTan(z, [DgtMax])	the same as the existing CplxTan( ) function, but with extended precision
xCplxTanh(z, [DgtMax])	the same as the existing CplxTanh( ) function, but with extended precision
xCStr(x, [DgtMax])	Converts a double 'x' into a string with absolute accuracy (see text above)
xDecr(x)	Decrements an extended number 'x' by 1; xDecr(x) = xSub(x, 1)
xDegrees(Angle, [DgtMax])	the same as Excels DEGREES( ) function, but with extended precision
xDelta(x, [y])	the same as Excels DELTA( ) function, but with extended precision
xDevSq(a, [DgtMax])	the same as Excels DEVSQ( ) function, but with extended precision
xDiff1(Var_Value, Func, [Lim], [Param], [Digit_Max], [Var_Label], [MaxPrecision])	the same as the existing Diff1( ) function, but with extended precision. Uses 5 point formula and Optim method=False of xDiffOpt
xDiff2(Var_Value, Func, [Param], [Digit_Max], [Var_Label], [MaxPrecision])	the same as the existing Diff2( ) function, but with extended precision. Uses 5 point formula and Optim method=False of xDiffOpt
xDiff (Var_Value, Func, [Param], [Digit_Max], [Order], [MaxPrecision], [Var_Label], [Delta_H])	simplest 1st to 20th Order derivitive of f(x). If missing Order = 1 'x' is default [Var_Label] [MaxPrecision] limits precision to calculate with. [DgtMax] is digits to return. [Delta_H] allows user to pass small value Delta (h), else it is calc'ed
xDiffI(Var_Value, Func, [Param], [Digit_Max], [Order], [NumOfPoints], [Offset], [MaxPrecision], [MaxDerEstCnt], [FindBestHflag],[Var_Label], [Coef])	any Order derivitive of f(x). If missing Order = 1. 'x' is default [Var_Label]. [MaxPrecision] limits precision to calculate with. [DgtMax] is digits to return. Optimizes h [Delta_H] by an Iterative method of “intersecting lines”, whereas xDiffOpt( ) has optional methods for optimization. MaxDerEstCnt limits the number of iterations. FindBestHflag is True or False (default). Setting to True finds the optimum h for the machine precision being used.
xDiffOpt(Var_Value, Func, [Param], [Digit_Max], [Order], [NumOfPoints], [Offset], [MaxPrecision], [Optim], [Var_Label], [Delta_H], [Coef])	any Order derivitive of f(x). If missing Order = 1. 'x' is default [Var_Label]. [NumOfPoints] determines formula to use. Minimum is (Order + 1) [Offset] shifts laterally left or right. 0 is right pt. NumOfPoints-1 is left pt. Offset = (NumOfPoints-1)/2 for central point (NumOfPoints must be odd). [MaxPrecision] limits precision to calculate with. [DgtMax] is digits to return. [Delta_H] allows user to pass small value Delta (h) between points [Optim] 'False' is default. 'True' optimizes Delta by method of R. de Levie [Coef] allows user to pass Coefficient array. If missing it calc's them.
xDivR (x, y, [DgtMax])	internal function the same as xDiv, but 10% faster in VBA Projects. Can also be called from the spreadsheet, where it is 3% faster, but returns numbers in 'Raw' format: as an integer and an unformatted exponent

xDpoly(x, Coef, ord, [DgtMax])	the same as the existing Dpoly( ) function, but with extended precision
xERF(x, [DgtMax])	the same as Excels ERF( ) function, but with extended precision
xERFC(x, [DgtMax])	the same as Excels ERFC( ) function, but with extended precision
xEval(Formula, [VarArray], [DgtMax], [AngleSet], [Tiny], [IntSwapFix])	adds an optional parameter "Tiny" to the existing function xEval(Formula, [VarArray], [DgtMax], [AngleSet]) (see text above). Also increases DgtMax used in internal calculating according to the number of mathematical operators in the formula string, resulting in more accuracy with many operators
xEven(x)	the same as Excels EVEN( ) function, but with extended precision
xExpBase(a, x, [DgtMax])	existing private function made available (for compatibility with custom VBA macros). Calculates a^x the same as xExpa
xExponent(x)	returns exponent of extended number, the second value returned from xSplit( )
xFisher(x, [DgtMax])	the same as Excels FISHER( ) function, but with extended precision
xFishInv(y, [DgtMax])	the same as Excels FISHERINV( ) function, but with extended precision
xFloor(x, signif)	the same as Excels FLOOR( ) function, but with extended precision
xFmt(x, [DgtMax], [Zeros])	formats a numeric string according to your configuration settings for "lead zeros" and "trailing zeros". Also formats complex numbers. Optional [Zeros] allows you to specify zeros other than the configured default
xFractCont(x, [DgtMax], [n] [StopMethod], [StopValue])	adds an additional optional parameter n to the existing function xFractCont(x, [DgtMax]), allowing you to specify the number of results, n (see text above)
xGEstep(x, [step])	the same as Excels GESTEP( ) function, but with extended precision
xGrad(Var_Values, Func, [Param], [DgtMax], [Var_Labels])	the same as the existing Grad( ) function, but with extended precision. Var_Labels allows changing the variables (default "x", "y", "z", "t")
xHmean(a, [DgtMax])	the same as Excels HARMEAN( ) function, but with extended precision
xImag(x)	returns the imaginary coefficient of a complex number, the same as Excels IMAGINARY( ) function
xIncr(x)	Increments an extended number 'x' by 1; xIncr(x) = xAdd(x, 1)
xIntercept(ay, ax, [DgtMax])	the same as Excels INTERCEPT( ) function, but with extended precision
xIntLog2(x)	returns corrected Int(Log(Abs(x))/Log(2)) for Xnumbers. Same as xInt(xLog(xAbs(x),2), with absolute accuracy near exact powers of 2
xIntLog10(x)	returns corrected Int(Log(Abs(x))/Log(10)) for Xnumbers. Same as xInt(xLog(xAbs(x),10), with absolute accuracy near exact powers of 10
xIsErr(a1, [a2], [a3],…)	inspects multiple input ranges for errors, returns 0 if none, otherwise 1
xIsErrNA(a1, [a2], [a3],…)	inspects input ranges for errors (excluding #N/A), returns 0 if none, else 1
xIsEven(a) As Boolean	the same as Excels ISEVEN( ) function, but with extended precision
xIsNumeric(x)	returns 0 for non-numeric text, 1 for a double (an Excel number), 2 for a numeric text string
xJacobian(Var_Values, Func, [Param], [DgtMax], [Var_Labels])	the same as the existing function Jacobian( ), but with extended precision. Var_Labels allows changing the variables (default "x", "y", "z", "t")
xLogistic(x, avg, dev, [Dtype], [DgtMax])	the same as the existing function DSLogistic( ), but with extended precision

xLogNorm(x, avg, dev, [Dtype], [DgtMax])	Log-normal distribution with extended precision. Optional Dtype can be entered as 1 or TRUE; 0 or FALSE If Dtype is missing, Dtype = 0 (FALSE)
xMantissa(x)	returns the mantissa of an extended number. The same as entering xSplit( ) into a single cell
xMax(a1, [a2], [a3],…)	The existing function xMax(a) is modified to analize multiple ranges (matrices) rather than a vector (single row or column)
xMaxwell(x, a, [Dtype], [DgtMax])	the same as the existing function DSMaxwell( ), but with extended precision
xMCond(Mat, [DgtMax], [ETA])	Multi-precision Condition number of a Matrix using Singular Value Decomposition. Derived from matrix.xla (Foxes Team)
xMedian(a1, [a2], [a3],…)	the same as Excels MEDIAN( ) function, but with extended precision
xMin(a1, [a2], [a3],…)	The existing function xMin(a) is modified to analize multiple ranges (matrices) rather than a vector (single row or column)
xMode(a1, [a2], [a3],…)	the same as Excels MODE( ) function, but with extended precision
xMpCond(Mat, [DgtMax] , [ETA])	Multi-precision -log10 of Condition number of a Matrix using Singular Value Decomposition. Derived from matrix.xla (Foxes Team)
xMround(x, multiple)	the same as Excels MROUND( ) function, but with extended precision
xMultinom(a, [DgtMax])	the same as Excels MULTINOMIAL( ) function, but with extended precision
xMultR (x, y, [DgtMax])	internal function the same as xMult, but 10% faster in VBA Projects. Can also be called from the spreadsheet, where it is 3% faster, but returns numbers in 'Raw' format: as an integer and an unformatted exponent
xNegR(x)	internal function the same as xNeg, but 10% faster in VBA Projects. Can also be called from the spreadsheet, where it is 3% faster, but returns numbers in 'Raw' format: as an integer and an unformatted exponent
xNormal(x, avg, dev, [Dtype], [DgtMax])	the same as Excels NORMDIST( ) function, but with extended precision
xNormalS(z, [Dtype], [DgtMax])	the same as Excel 2010 NORM.S.DIST( ), but with extended precision
xNormS(z, [DgtMax])	the same as Excels NORMSDIST( ) function, but with extended precision
xOdd(x)	the same as Excels ODD( ) function, but with extended precision
xOddDen (x, [DgtMax])	Checks if the denominator of 'x' is odd (TRUE) or even (FALSE), from the resulting denominator of 'x' as string from xFract(x, DgtMax), or 'x' as number from Fract(x)
xPearson(ax, ay, [DgtMax])	the same as Excels PEARSON( ) function, but with extended precision
xPoly(x, Coef, [DgtMax])	the same as the existing function Poly( ), but with extended precision
xPolyAdd(Poly1, Poly2, [DgtMax])	the same as the existing function PolyAdd( ), but with extended precision
xPolyDiv(Poly1, Poly2, [DgtMax])	the same as the existing function PolyDiv( ), but with extended precision
xPolyMult(Poly1, Poly2, [DgtMax])	the same as the existing function PolyMult( ), but with extended precision
xPolyRem(Poly1, Poly2, [DgtMax])	the same as the existing function PolyRem( ), but with extended precision
xPolySub(Poly1, Poly2, [DgtMax])	the same as the existing function PolySub( ), but with extended precision
xPolyTerms(Polynomial, [DgtMax])	the same as the existing function PolyTerms( ), but with extended precision
xPow2(n, [DgtMax])	n is raised to the power of 2. A faster function than xPow(n,2,[DgtMax])
xqCos(x), xqExp(x), xqExpa(x, a), xqLn(x), xqLog(x, [BASE]),	existing internal functions made available to the user. These are faster functions than the corresponding "x-functions" to return up to 29 digits

xqSin(x), xqAngleC(x)	(from VBA's "Variant Decimal" data type)
xRad5([DgtMax])	existing multiprecision 'constant' made available. Returns square root of 5
xRadians(angle, [DgtMax])	the same as Excels RADIANS( ) function, but with extended precision
xRand([DgtMax])	returns a random numeric text string between 0 and 1, the same as Excels RAND( ) function, but with a length of the requested DgtMax
xRandD(x, y, [DgtMax])	returns a random numeric text string between x and y, with a length of the requested DgtMax
xRandI(x, y, [DgtMax])	returns a random integer as a numeric text string between x and y, up to the requested DgtMax
xRank(num, ref, [order])	the same as Excels RANK( ) function. It is also an existing internal function
xRayleigh(x, dev, [Dtype], [Digit_Max])	the same as the existing function DSRayleigh( ), but with extended precision
xRdown(v, [Dec])	the same as Excels ROUNDDOWN( ) function, but with extended precision
xReal(x)	returns the real coefficient of a complex number, the same as Excels IMREAL( ) function
xRsq(ay, ax, [DgtMax])	the same as Excels RSQ( ) function, but with extended precision
xRup(v, [Dec])	the same as Excels ROUNDUP( ) function, but with extended precision
xSerSum(x, n, m, a, [DgtMax])	the same as Excels SERIESSUM( ) function, but with extended precision
xSlope(ay, ax, [DgtMax])	the same as Excels SLOPE( ) function, but with extended precision
xSqrPi(x, [DgtMax])	the same as Excels SQRTPI( ) function, but with extended precision. Returns the square root of (x*Pi). If x is omitted x=1. If x<0 returns error #VALUE!
xSumProd(ax, ay, [DgtMax])	same as Excels SUMPRODUCT( ) function, but with extended precision
xSumSq(a, [DgtMax])	the same as Excels SUMSQ( ) function, but with extended precision
xSumX2mY2(ax, ay, [DgtMax])	same as Excels SUMX2MY2( ) function, but with extended precision
xSumX2pY2(ax, ay, [DgtMax])	same as Excels SUMX2PY2( ) function, but with extended precision
xSumXmY2(ax, ay, [DgtMax])	the same as Excels SUMXMY2( ) function, but with extended precision
xSVDD(Mat, [DgtMax] , [ETA])	Singular Value Decomposition: returns D matrix in Multi-Precision. Derived from matrix.xla (Foxes Team), which return results in standard precision
xSVDU(Mat, [DgtMax] , [ETA])	Singular Value Decomposition: returns U matrix in Multi-Precision. Derived from matrix.xla (Foxes Team), which return results in standard precision
xSVDV(Mat, [DgtMax] , [ETA])	Singular Value Decomposition: returns V matrix in Multi-Precision. Derived from matrix.xla (Foxes Team), which return results in standard precision
xMPseudoinv(Mat, [DgtMax] , [ETA])	Moore-Penrose Pseudo-Inverse of SVD decomposition
xRegrL(y, x, [Digit_Max], [Intcpt] , [ETA])	Returns the coefficients of the linear regression y = a0 + a1*x1 + a2*x2 + a3*x3 + … + am*xm using SVD decomposition
xTruncR(x, [SigDgts])	truncates an extended number to optional significant digits
xWeibull(x, k, lambda, [Dtype], [DgtMax])	the same as Excels WEIBULL( ) function, but with extended precision
xZeta(x, [Digit_Max])	the same as the existing Zeta( ) function, but with extended precision
x_And(a1, [a2], [a3],…)	Multiprecision AND function returns TRUE or FALSE. Numeric string evaluates to TRUE, '0' as string evaluates to FALSE, whereas Excel's AND does not evaluate strings. Empty cells and non-numeric strings are skipped

x_If(TestVal, [TrueVal], [FalseVal])	Multiprecision IF function evaluates TRUE and FALSE conditions. A numeric string will evaluate to TRUE, '0' as string will evaluate to FALSE, whereas Excel does not evaluate numeric strings
x_Not(a)	Multiprecision NOT function reverses TRUE or FALSE. A numeric string will evaluate to TRUE, '0' as string will evaluate to FALSE, whereas Excel does not evaluate strings
x_Or(a1, [a2], [a3],…)	Multiprecision OR function returns TRUE or FALSE. Numeric string evaluates to TRUE, '0' as string evaluates to FALSE, whereas Excel's OR does not evaluate strings. Empty cells and non-numeric strings are skipped
subroutine XNsetup	this Macro can be run from XN.csv to update the function definitions from the fourth column. It won't appear on the list of Macro's: just type in 'xnsetup'. 255 character limit. Alt+Enter for a carriage return within a cell. Columns D & F cannot start with a number unless there is a comma within the cell

---

Installation note for Excel 2007 and 2010

This Add-In requires 3 files:

XN.xlam
XN.chm (or XN.hlp)
XN.csv

Make sure the files are unblocked: right-click, choose "Properties", then in the "General" tab click "Unblock".

The same as always, place these files in any folder you choose.

Follow the usual procedure for installing Excel Add-Ins:

1) open Excel
2) from the Office Button (2007) or the File tab (2010), select "Excel Options" , "Add-Ins" , "Go"
3) then "Browse" for and select "XN.xlam"

Go to "Excel Options" from the Office Button or File tab, click "Trust Center" , "Trust Center Settings" , "Macro Settings" , then check "Enable all macros" and "Trust access to the VBA project".

It is also helpful to make sure the Add-In is in a Trusted Location in the "Trust Center" under "Trusted Locations". If not: "Add new location" , and type in the address or "Browse" for the location of the appropriate folder.

You might get a "Security Warning" in the toolbar when you open a workbook: Click "Options" and "Enable this content". It might be helpful to "Enable automatic update…" in "Trust Center Settings" under "External Content".

It is recommended to Save the Add-In on your machine in your environment, using the "Compiled" button in the configuration screen, from the Xnumbers toolbar under "X-Edit", "Configuration"

Note on multiple installations:

When installing more than one Xnumbers Add-In to compare results with different versions, the *first* one installed will be the default. To link to another, enter the file name followed by an exclamation before the function:

=XN13.xlam!xAdd(a,b) or =xnumbers.xla!xadd(a,b)

In 2007 and 2010 multiple toolbars get installed so that there is access to each Add-Ins Macros. In 97/2000/XP/2003 only the first toolbar will get installed. To access Macros, use Tools>Macros (Alt+F8) and enter:

XN13.xla!Functions_Handbook or xnumbers.xla!Functions_Handbook

Click Run, and the specified Handbook appears. Choose "Sub" for Macro Type, highlight the desired Macro, and click OK. Setting_Digits_Default is the configuration screen. To access that directly from Tools>Macros enter:

XN13.xla!Setting_Digits_Default or xnumbers.xla!Setting_Digits_Default

---

Accessing Xnumbers Functions with Custom VBA Macros and Functions

Note to Steve,

Change for version 6.0.4

Most Xnumbers functions can be accessed with custom VBA Macros and Functions. Open a code module in the workbook in which you want to use the Add-In's functions then go to **Tools > References** to open the **References** dialog where you will see a list of all the libraries and other objects (like Add-Ins) to which you can set a reference. Put a tic mark in the box next to **Xnumbers60.** Then the following macro works just fine.

```
Sub test()
Dim a$
a = xAdd(1, 2)
End Sub
```

Functions that can return more than one cell to a spreadsheet will **not** work with the above example because we use Application.Caller to determine the correct direction. I created three Public variables that when used will allow access to these functions as well. You set xNumInvAppCallFlg = True at the beginning and = False at the end of the macro. You then simulate the number of return cells by setting xNumACCols and xNumACRows to the desired values. In the following macro we request a single cell return from xCplxDiv.

```
Sub test()
Dim a$
xNumInvAppCallFlg = True
xNumACCols = 1
xNumACRows = 1
a = xCplxDiv("3+i", "2+j", 50)
xNumInvAppCallFlg = False
End Sub
```

This should eliminate the need to dual load xnumbers.dll for VBA access, make all the code fully accessible, and increase the number of functions available. All options selected from the configuration form will be in effect for the function calls. Although setting xNumInvAppCallFlg = True is not always necessary, it will never do any harm. xNumACCols and xNumACRows are never used unless xNumInvAppCallFlg = True. To view a list of the available functions open a code module in the workbook in which you want to use the Add-In's functions then go to **View > Object Browser** select **Xnumbers60** in the **Project/Library** drop-down box and click on **<globals>**. Subroutines will also be listed, but I do not think very many will be useful/valid. Double clicking a function will open the Xnumbers module for that function.

Since I am not familiar with all the possible uses of xnumbers.dll, I cannot say for sure that it is now unnecessary. It would probably execute faster, but not all the functions or configurations could be supported, it would use quite a bit of memory, and the code would be hidden from the VBA programmer.

John

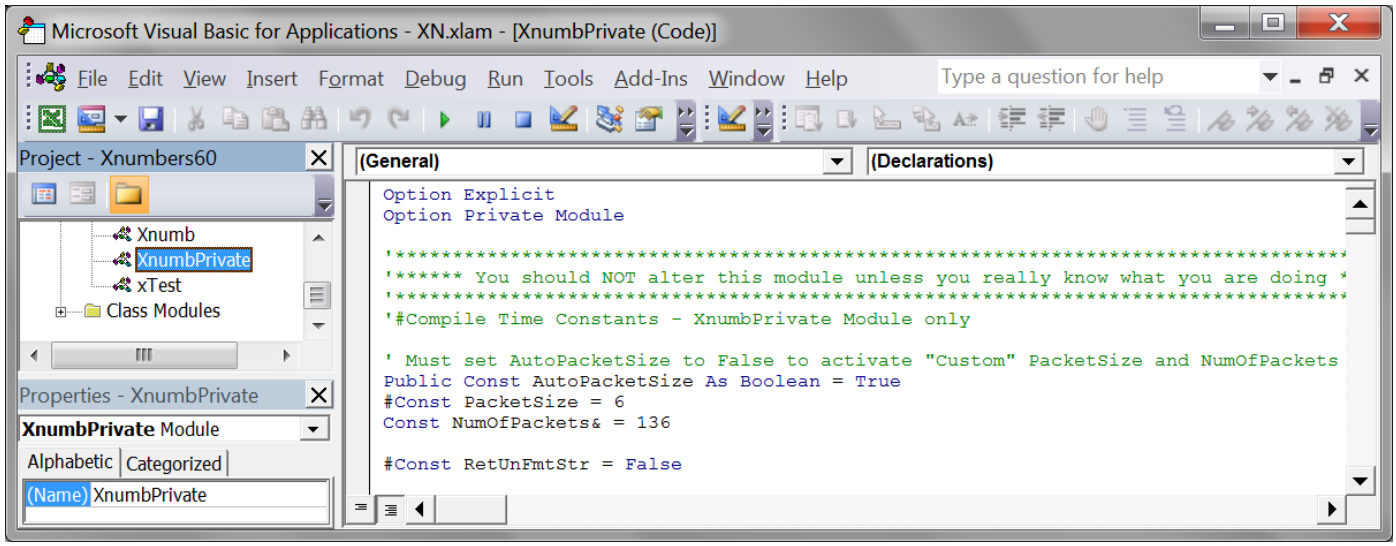
P.S. – There is another function SetXnDefaultDigits which sets Default digits in the configuration screen from VBA, and a function GetXnDefaultDigits which retrieves the currently configured value.

```
Sub test()
Dim OrigDefDgts As Long
OrigDefDgts = GetXnDefaultDigits()
SetXnDefaultDigits 30
'
'  Put custom macro code here
'
SetXnDefaultDigits OrigDefDgts
End Sub
```

This is a list of the “Get” and “Set” functions from the Xnumbers60 'Initialize' Module:

GetXnDefCStr	GetXnUseXroundIN	SetXnDefCStr	SetXnUseXroundIN
GetXnDefaultDigits	GetXnTrailZeros	SetXnDefaultDigits	SetXnTrailZeros
GetXNxAddAdj	GetXnLeadZeros	SetXNxAddAdj	SetXnLeadZeros
GetXNxDivAdj	GetXnSMPadj	SetXNxDivAdj	SetXnSMPadj
GetXNxMultAdj	GetXnCaseSen	SetXNxMultAdj	SetXnCaseSen
GetXnUseXroundOUT	GetDigitsLimit	SetXnUseXroundOUT	SetDigitsLimit

To compile with different packet sizes (xBASE) and Digits_Limits open the Xnumbers60 'XnumbPrivate' Module:



To set custom packet size change 'Public Const AutoPacketSize As Boolean = True' to = False.
The line '#Const PacketSize = 6' compiles for packet size 6; change the 6 to desired packet size. 5 thru 14 are valid.
The line 'Const NumOfPackets = 136' compiles for a Digits_Limit of 136*6=816 significant digits; change the 136 to desired number of packets, then 'Compile' and 'Save' . The table at the top of XnumbPrivate can be used as a guideline.

Version 6.0.5.2: To compile the Add-In to return unformatted strings with the full length of internal results there is now simply a checkbox in the configuration screen Examples:
xDiv(2,3,15) returns 3 packets using PktSz 7, the unformatted 21 digit string 666666666666666666666E-21
xFmt(xDiv(2,3,15),15) returns the 15 digit string 0.666666666666667 with "Return Rounded Xnumbers" option.
sFmt(xDiv(2,3,15),15) returns the 21 digit string 6.66666666666666666666E-1 in standard scientific format.

Note:For some unknown reason a spreadsheet with a reference to Xnumbers60 does not perform auto-capitalization as expected. Sub ReDoCapitalization() can be set up in ThisWorkbook to run once when the spreadsheet is opened to ensure that functions in that spreadsheet also perform auto-capitalization:

```
Private Sub Workbook_Open()
Xnumbers60.ReDoCapitalization
End Sub
```

For questions, comments or “bug reports” email to steve@thetropicalevents.com

April 2012