Nematrian Function Library
Reference Manual

Version dated: 30 May 2023

The Nematrian online software toolkit

[Nematrian website page: IntroductionSoftware]

To complement its other business activities, Nematrian makes available through its website a wide range of accurate, efficient and well-documented online analytical tools. Our software toolkit has the following characteristics:

Breadth and Depth

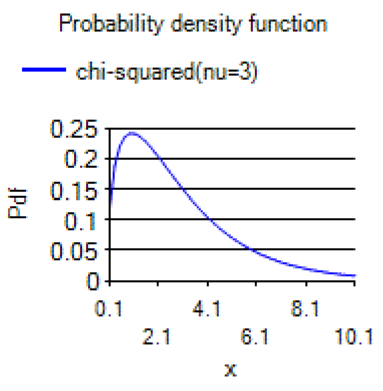
There are more than 700 functions already available in the Nematrian website toolkit. More are being added regularly. They include many financially orientated functions as well as a wide range of more generally applicable mathematical and statistical functions and tools, including tools for plotting results.

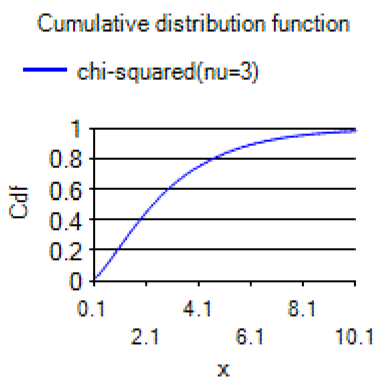
Easy accessibility

You can access the tools interactively, if this suits your needs best

Each function has an associated page on this website documenting what the function does and allowing you to run the function interactively, if you wish. You can also access all the functions using our online calculator (our "expression evaluation" tool), nesting function calls if you so wish.

As explained in the website introduction, interactive results are typically provided as hyperlinks that have embedded within them details of the underlying parameters used. For example the following graphical results obtained interactively using the Nematrian toolkit illustrate the chi-squared distribution. If you click on either chart it will take you to the page used to create that chart, pre-populated with the parameters used to do so.





Or, you can access the tools through spreadsheets (or more sophisticated programming environments), if you want to carry out more advanced manipulations.

To make it as easy as possible to use our toolkit in more advanced settings we provide example spreadsheets which illustrate sets of related tools, see here. The VBA code that these spreadsheets use to access the Nematrian website provides a full Application Programming Interface (API) for the toolkit. Similar approaches can be used to access the tools through many other more sophisticated programming environments.

Affordability

We currently operate a tiered approach to usage. Anyone can use most of the tools interactively in moderation without charge, although tools accessed through the free of charge route may run more slowly than those that are accessed on a paid for basis. The more sophisticated tools (or ones consuming more than a certain amount of processor power) are available on a subscription basis. Please contact us for more details of our subscription arrangements including pricing.

Web services and Web functions

[Nematrian website page: WebServicesIntro]

The Nematrian website gives you access to a wide range of on-line ('cloud-based') tools using the medium of *web services*. To see a complete list of tools it makes available please go to the list of types of Nematrian web services or complete list of functions. Making tools available in this way makes them accessible to any PC with internet access.

To maximise the usefulness of these tools, they can be accessed in multiple ways, including:

Interactively, focusing on a specific function

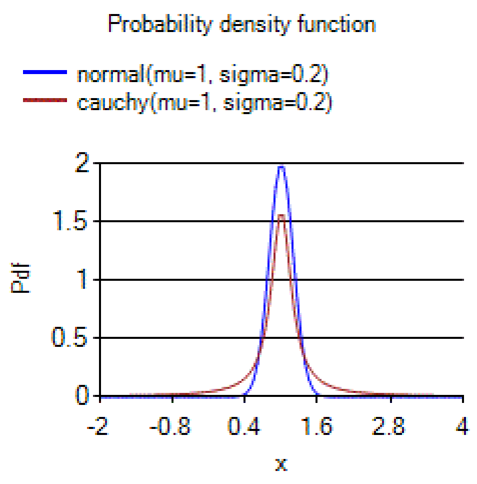
If you follow links to a specific function then one of the options available to you is to '*interactively run the function*'. This takes you to a form where you can see the result of applying the function to inputs you have entered into the form. To help you get started the page also includes an example.

Interactively, using the Nematrian expression evaluator

The Nematrian website's expression evaluator is a general purpose on-line calculator into which you can put more complicated or compound formulae. For example, if you enter 2+3 it will recognise this as 5. Various conventions allow you to manipulate arrays and other more complicated expressions. Any of the individual functions referred to above can be accessed within the expression evaluator.

Embedded within hyperlinks

If you need to share your calculations with others then another helpful way of accessing the Nematrian function library is via hyperlinks that show the end answer of the calculation but embed within the hyperlink the inputs that generated that answer. Typically the results of using either individual function pages or the expression evaluator are returned to the user in the form of such hyperlinks such as the following:



Via spreadsheets

The Nematrian function library can also be accessed via spreadsheet systems. For those familiar with Microsoft Office, the way in which the library is accessed through this route is conceptually akin to use of built-in functions such as SUM or COUNT that can be entered directly into Microsoft Excel worksheet cells, or akin to the more sophisticated functions or subroutines that you can create using the ‘macro’ language that comes packaged with Excel, i.e. Microsoft Visual Basic for Applications. Names of Nematrian web functions accessed in this manner begin with “Mn”.

For example, the Nematrian function library includes a function Abs which returns the absolute value of a real number (i.e. $-x$ if $x < 0$ and x if $x \geq 0$). To avoid confusion with Microsoft Excel’s own ABS function (which in this case Abs replicates), the Nematrian Abs function should be accessed using MnAbs(x) in the spreadsheet API. The page on the Nematrian website giving further details of this function is also called MnAbs. Note: when using the tool interactively or via an embedded hyperlink as above the Nematrian website will recognise either Abs(x) or MnAbs(x) interchangeably.

To make it easier for you to use of the library in this manner, each Nematrian function page contains a link to an example spreadsheet containing an example use of the function as well as example uses of other related functions. Any other function within the library can be accessed through the same spreadsheet, but the examples themselves are subdivided into smaller groups to make the process of illustrating related functions more manageable.

If you don’t like downloading spreadsheets with embedded macros then a step-by-step guide to creating a VBA module from scratch that permits access to the entire Nematrian function library is available here. Instructions on how to access the functions using the WEBSERVICE function built into Excel 2013 and later are available here.

Via more sophisticated programming environments

Information for software developers wanting to understand how to access Nematrian web functions through more sophisticated programming environments such as Visual Studio is set out here.

Formatting Charts produced by the Nematrian website

[ChartFormat]

In many situations presentation of results is important. A common way of presenting mathematical or numerical work is to plot the results in chart form.

The Nematrian charting algorithms allow the user to adjust some of the presentational characteristics of the chart using a *ChartFormat* parameter. The *ChartFormat* parameter is a string of the form "xxx=aaa yyy=bbb …" etc., where e.g. xxx is the sub-parameter being set and aaa is the value to be assigned to it. Where the value is in the form of a string variable then it needs to be enclosed in quotes (i.e. as e.g. xxx="aaa"). A quote sign within such a string should be referred to by two consecutive quote signs. This is particularly important if the string contains a space as otherwise the space will be interpreted by the website as delimiting different sub-parameters.

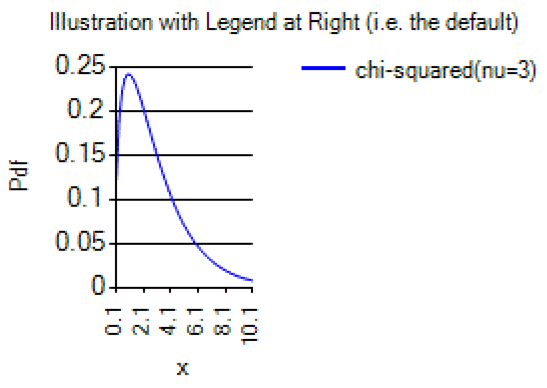
Valid sub-parameters currently include the following:

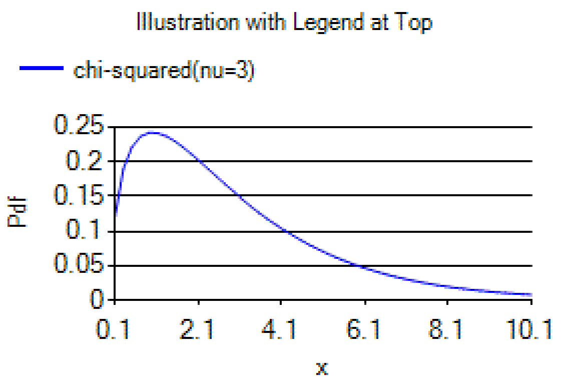
Sub-Parameter	**What it relates to**
LegendDocking	Indicates where relative to the plot area the legend appears (default is to the right)
Width	Width (in pixels) of overall chart (default width is 600)
Height	Height (in pixels) of overall chart
SeriesType0, SeriesType1, …	Chart type used for series 0, series 1 etc. Series 0 is the first series plotted, series 1 is the second etc. (default is usually a line chart)
SeriesName0, SeriesName1, …	Name ascribed to the series when it is plotted (default is usually the expression being plotted)
SeriesColor0, SeriesColor1, …	Colour of line ascribed to the series when it is plotted (default uses blue for first series, brown for next series etc.)
SeriesLineStyle0, SeriesLineStyle1, …	Style of line ascribed to the series when it is plotted (the default is a solid line)
xMajorGrid	Indicates if major grid lines for x-axis are to be included (default is false)
yMajorGrid	Indicates if major grid lines for y-axis are to be included (default is true)
xMinimum	Minimum x value (overriding the default selected by website)
xMaximum	Maximum x value (overriding the default selected by website)
yMinimum	Minimum y value (overriding the default selected by website)
yMaximum	Maximum y value (overriding the default selected by website)
xLabel	Label to describe x axis (default is usually "x"
yLabel	Label to describe y axis (various defaults depending on function)

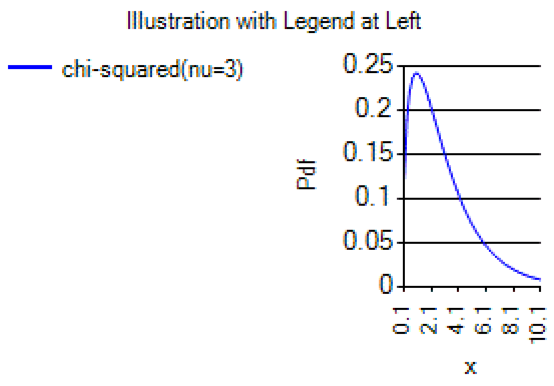
Sub-parameter names are not case sensitive.

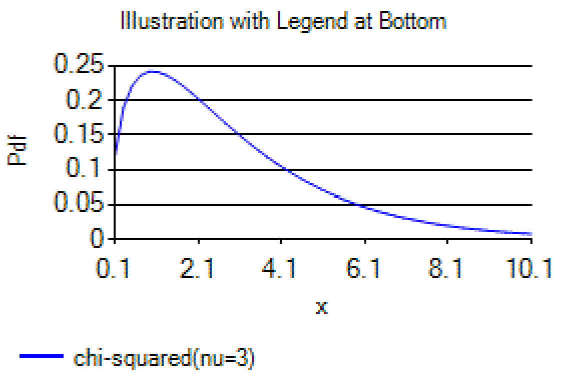
Examples

The following four charts illustrate the use of the LegendDocking sub-parameter. Recognised values are returned by MnRecognisedChartLegendDockings. They include Right, Top, Left and Bottom and are not case sensitive.

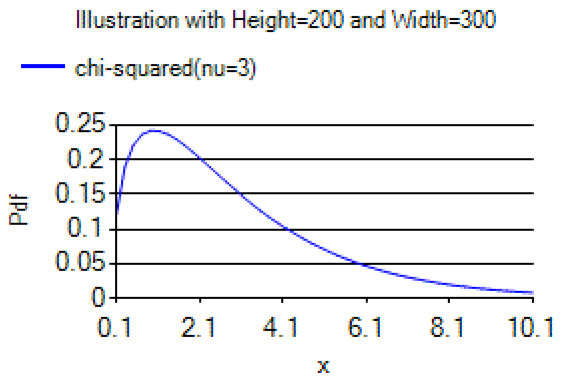


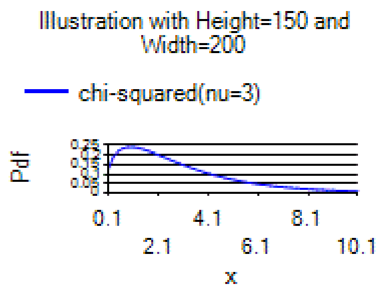




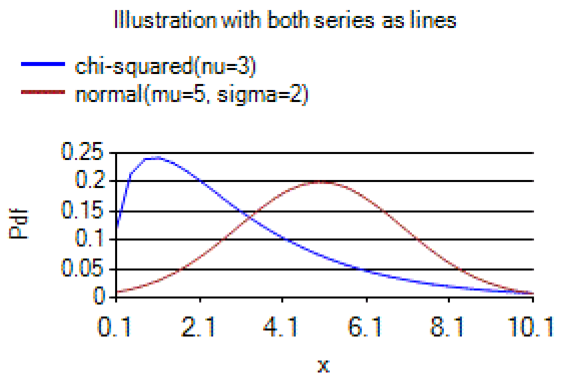


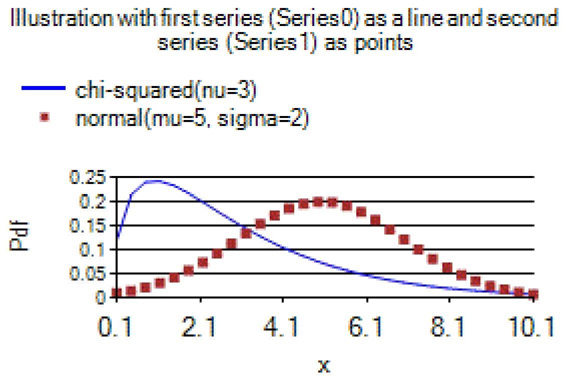
The following two charts illustrate the use of the Height and Width sub-parameters.



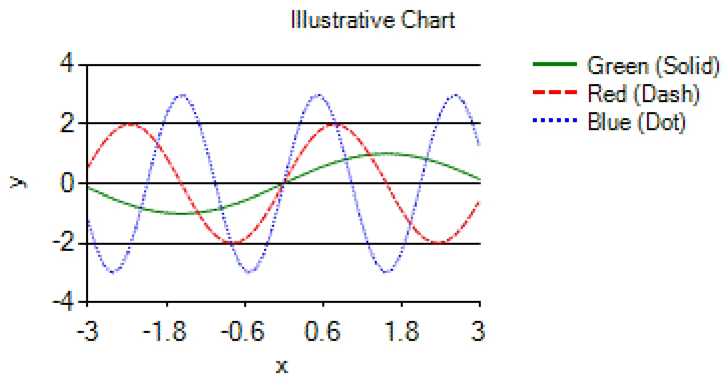


The following two charts illustrate the use of SeriesType. Recognised values are returned by MnRecognisedChartSeriesTypes. They include “Line” and “Point” and are not case sensitive. Only some values returned by MnRecognisedChartSeriesTypes will be meaningful for a specific type of chart, such the *xy* plots used in the charts below.





The following chart illustrates the use of SeriesColor and SeriesLineStyle. Recognised values are returned by MnRecognisedChartSeriesColors and MnRecognisedChartSeriesLineStyles. They include e.g. Black, Blue, Green, Pink, Purple, Red, Yellow (Color) and Solid, Dash, Dot (LineStyle), and are not case sensitive.



All charts produced by the Nematrian website are technically also old-style SmartCharts. Sub-parameters relevant to this aspect of the chart are:

Sub-Parameter	**What it relates to**
SmartText	Text characterising the appearance of the SmartChart
Creator	The creator of the SmartChart
Comment	A comment added by the creator in relation to the chart
Referencewebsite	A website that is associated with the SmartChart
LimitOtherUsers	Flag indicating if SmartChart is only viewable by certain users
OtherAllowedUsers	A SmartUList indicating which Users can view the SmartChart, if *LimitOtherUsers* is set to true

The default for each of these sub-parameters is blank, except for *LimitOtherUsers* which is false.

Session IDs

[SessionIdIntro]

As explained in the Software Introduction pages, you can currently access large parts of the Nematrian website free of charge, including many of its analytical tools. However, access to some parts or tools, particularly ones accessed through spreadsheets, may require you to obtain a temporary session-specific *Session Id* (which you can currently do by pressing a button at the top of the page used to access spreadsheets in the Nematrian spreadsheet library), or to log in and/or to enter into a subscription agreement. *Session IDs* allow the Nematrian website to keep track of your utilisation of its facilities.

Accessing the website via a browser

If you are only going to access the web-site directly via a browser then in practice you currently typically do not need to worry much about *Session IDs*. This is because the website currently typically automatically creates one for you in the background and then 'remembers' your *Session ID* as you navigate around its pages (if your browser has cookies enabled).

Accessing the website via its associated web services

Session IDs become more important if you are planning to make use of the website's web services. This is typically how you access the Nematrian function library if you are doing so through a spreadsheet or a more complicated programming environment. Each time you programatically call one of the Nematrian web services you need to provide a current *Session ID* as well as the parameter values that the web service will use in its computations. If you are logged in to the site or you have obtained a temporary Session ID then your current *Session ID* will be shown in a box towards the top of the page referred to above, along with the number of CPU computation units still remaining for it and the number of minutes remaining before it will time out.

Running functions interactively

[WebServiceInteractiveInstructions]

To run a function interactively, please enter suitable inputs into the boxes in the interactive table and then press one of the *Calculate* buttons. For some functions and/or parameters a delimiter may be requested (e.g. a comma, to separate array elements), see also here.

Press the *View an example* button to see example inputs.

If an input is an expression that you want the website to evaluate before using it as an input to this function then check the box in the column "An expression input?". For example, check this box if you want to insert as input 2+3 and you want the website to recognise this as 5. Expression construction is explained further here.

If you want to include a description of the calculation and/or a time stamp embedded within the answer then fill in the calculation description box and/or check the time stamp box. If you copy and paste the answer into a document then clicking on it will take you back to this page, pre-populated with the inputs you have entered and the calculation description / time stamp.

Computation units

[WebServiceComputationUnits]

[Explains number of computation units deducted from a SessionId whenever you run a function in a particular way.]

The Nematrian online software toolkit: Other topics

[IntroductionSoftwareOther]

Other topics covered in the Software section of the Nematrian website include:

(a) Expression evaluation: further information on Nematrian's online calculator and how it can be used to access the Nematrian function library

(b) Web services (Introduction): an introduction to web services

(c) Web services (Using in Excel/VBA): how to use web services (particularly ones accessing the Nematrian function library) through Excel/VBA

(d) Web service code generator: how to create VBA code (or equivalent) that can be inserted into Excel (or other spreadsheet packages) to access the Nematrian function library through a spreadsheet front end

(e) Web services (Using in more sophisticated programming environments): information as per (d) but for more sophisticated programming environments, such as Visual Studio

(f) Nematrian web service functions (Frequently Asked Questions): further help for problems you may come across when using web services

(g) Smart utilities, including an explanation of how charting is carried out by the Nematrian function library

APPENDIX A: USING THE NEMATRIAN FUNCTION LIBRARY INTERACTIVELY

Expression Evaluator

[ExpressionEvaluator]

To use the expression evaluator enter the expression you wish to evaluate into the Input text box (and optionally a description of the calculation in the Calculation description text box) and then press the *Calculate* button. You can also enter the expression into the Search / Calculate text box at the top of the page and then press the + / button next to it. Expressions can include brackets, operators and calls to nearly all components of the Nematrian online function library. For further details, see here.

Important notice

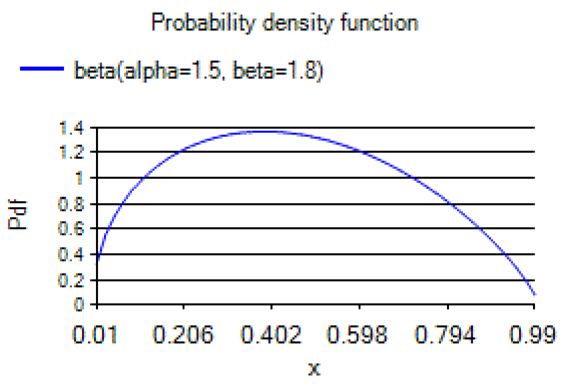
If you use any Nematrian web service either programmatically or interactively then you will be deemed to have agreed to the Nematrian website License Agreement.

The Nematrian website: Expression Evaluation

[ExpressionEvaluation]

One of the tools available on the Nematrian website is its Expression Evaluator. This tool allows users to use formulae rather than just numbers as inputs to online computations.

As with interactive use of individual Nematrian function webpages, the results are typically returned in the form of a hyperlink that embeds the input parameters used to carry out the calculation, to make it easier to check, reproduce or modify calculations at a later date. If results are charts then the same hyperlink embedding is also typically used. For example, if you wanted to move the legend in the following chart from above the figure to below it you could do so by clicking on the chart, altering the relevant *ChartFormat* parameter and then pressing the recalculate button on the page you landed on when you clicked on the chart.



There are four main ways of using the Expression Evaluator:

(a) Insert an expression into the text box on the expression evaluator page and then press *Calculate*. This will then evaluate the expression within the text box.

(b) Insert an expression into the Search / Calculate textbox at the top of any page on the website and press *Calc* button next to it. This replicates (a) but without needing to go to expression evaluator page beforehand.

(c) Enter an expression into an input text box in the interactive input table on any page summarising a specific Nematrian web function having checked the corresponding box under the column headed "An expression input?" and then press the *Calculate* button. The relevant input element of the Nematrian function is then taken as the output of the relevant expression. The page relating to any particular Nematrian function can be found from the list of Nematrian web functions or by searching the Nematrian website).

(d) Use an expression as an input to the (Mn)Evaluate function (and set its *Language* input parameter to blank) as this effectively replicates the output of the expression evaluator.

Formulae input into the expression evaluator are interpreted using a relatively simple syntax in line with typical algebraic conventions, i.e.:

- Recognised algebraic operators are + (plus), - (minus), * (multiply), / (divide) and ^ (power).

- Recognised logical operators are TRUE and FALSE, ">" (greater than), "<" (less than), ">=" (greater than or equal), "<=" (less than or equal), "=" (equals), "And" (logical and), "Or" (logical or), "XOr" (logical exclusive or, i.e. true if either x or y is true but not if both of them are true) and "Not"

- Order of evaluation is (i) Brackets (i.e. "(" and ")") and function evaluation, see below), (ii) power, (iii) mutiply and divide, then (iv) add and subtract

- All Nematrian Web functions are recognised, either including their leading "Mn" or without this prefix. Parameters follow immediately afterwards in brackets, with consecutive input parameters (if any) separated by commas. If an input parameter is an array then it should be enclosed in brackets with consecutive elements separated by commas.

- Any dates and times should be entered either using the format #YYYY-MM-DD# or #YYYY-MM-DDTHH:MM:SS# or using Nematrian functions that return values that are Dates, e.g. (Mn)DateSerial, (Mn)TimeSerial and (Mn)DateAndTimeSerial.

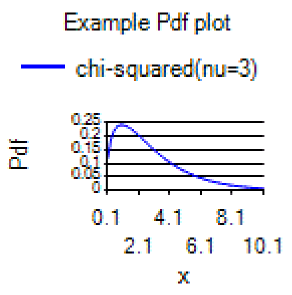
- Integer values are automatically converted to Floating Point (i.e. "Double" aka Double Precision) values if necessary.

Note

(1) Websites / browsers can be cautious if you enter "<" and ">" into text boxes (because it is possible to make malicious use of inputs that include these characters). If you come across this problem with this website then you can work around this issue by using the GT, GTE, LT and LTE functions, which return equivalent logical results to the >, >=, < and <= operators.

(2) It is not currently possible to use the expression evaluator recursively within itself. Functions such as (Mn)Evaluate, (Mn)EvaluateUsing, (Mn)EvaluateMultiValues, (Mn)EvaluateExpressionsUsing and (Mn)EvaluateExpressionsMultiValues will not therefore be evaluated by the expression evaluator. Neither will plotting functions that make use of the expression evaluator such as (Mn)PlotExpression, (Mn)PlotExpressions and

(Mn)PlotXYExpressionGeneral. Other more specialist plotting functions such as (Mn)ProbDistPlotPdf can be called via the expression evaluator, as illustrated below:



Important notice

If you use any Nematrian web service either programmatically or interactively then you will be deemed to have agreed to the Nematrian website License Agreement.

Help for using Nematrian web services interactively

[WebServicesInteractiveHelp]

If an input is an array then enter value in delimited form and indicate the delimiter. A delimiter is a piece of text such as a "," or a ";" that is placed between different entries in an input array.

If you leave the delimiter blank the website will assume that you are delimiting entries using tabs and/or carriage returns (or if no such characters are present with a comma, see MnParseStringIntoStringArray). If you paste one or more columns from Microsoft Excel into the relevant text box then this is the type of delimiter that is effectively contained in the text that you will be copying into the interactive input table.

If the output is an array then enter a piece of text that will be used to delimit individual output elements. If you leave this delimiter blank then the website will space out entries onto separate rows. If the output is two dimensional then it will usually provide a tabular output.

Please note that in numerical input commas may be ignored (so e.g. "1,2" may be recognised as 12 rather than throwing an error). Blank entries for Boolean, Double or Date input variables are by default converted to FALSE, 0 or Now.

If an input is an expression that you want the website to evaluate before using it as an input to this function then either check the box in the column "An expression input?". This allows you e.g. to insert 2+3 and for the website to recognise this as 5. Expression construction is explained further here.

Important notice

If you use any Nematrian web service either programmatically or interactively then you will be deemed to have agreed to the Nematrian website License Agreement.

APPENDIX B: FUNCTION TYPES

Types of functions included in the Nematrian Function Library

[Nematrian website page: FunctionLists]

The Nematrian function library currently includes functions covering the following topics. To view a list of functions in each category please follow the relevant hyperlink.

- Calculator-type computations (e.g. ones you might find on a pocket calculator)
- Cash-flow projections (including ones for carrying out pension fund cash flow projections)
- Charting (e.g. plotting functions or statistical distributions)
- Code generation (to help users manipulate the Nematrian function library)
- Complex numbers (akin to Calculator-type functions but accepting complex number inputs)
- Dates (to manipulate dates and times)
- Derivative pricing (e.g. to calculate Black-Scholes option pricing greeks)
- Expression evaluation (i.e. allowing calculation of complicated expressions)
- Finance (akin to those available through spreadsheets)
- General / miscellaneous (functions not currently categorised elsewhere)
- Life insurance (e.g. valuation of insurance exposures dependent on mortality rates)
- Markov processes (e.g. estimating them, simulating them or analysing their ergodic properties)
- Mathematical puzzles (e.g. solving Sudoku puzzles)
- Matrices (e.g. manipulating matrices and solving associated sets of linear equations)
- Performance measurement (including functions for manipulating peer group performance data)
- Portfolio optimisation (including efficient frontier derivation and reverse optimisation)
- Probability distributions (a comprehensive suite of functions for estimating and manipulating a wide range of statistical probability distributions)
- Risk management functions (some more specific risk management functions not otherwise classified elsewhere)
- Set manipulation (e.g. counting and deriving unions and intersections of sets)
- Solvency II (e.g. for calculating elements of its Standard Formula Solvency Capital Requirement)
- Sorting (sorting or identifying the order of different types of array)
- Special functions (i.e. more sophisticated mathematical functions than are included in Calculator-type functions)
- Statistics (a wide range of functions relevant to statistics not otherwise covered elsewhere, e.g. under Probability distributions functions)
- Strings (for manipulating strings, e.g. replacing strings in other strings)
- Tri-segmented Monte Carlo demonstrations (N.B. tri-segmented Monte Carlo is an approach for speeding up runtimes of simulation exercises)
- Unit conversions (from e.g. SI units to other units used for physical measurement)

To see a complete list of functions currently available in the Nematrian Online Toolkit please go to Complete Function List.

The Nematrian function library also includes a range of bespoke tools designed to meet specific client needs. These functions are not publicly accessible.

Calculator-type Functions

[Nematrian website page: CalculatorFunctions]

The Nematrian function library includes a range of tools that provide common functions as per an electronic calculator.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Cash-Flow Projection Functions

[Nematrian website page: CashFlowProjectionFunctions]

The Nematrian function library includes a range of tools that are useful for carrying out cash-flow projection calculations.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

Disclaimer

Please note that no web functions currently *publicly* available through the Nematrian website are classified by Nematrian as being explicitly 'actuarial' in nature. For example, they do not necessarily adhere to relevant model standards imposed on actuarial work by the UK's Board for Actuarial Standards. If you are seeking access to models that do explicitly adhere to these standards then please speak to your usual contact at Nematrian. Usage of the Nematrian website is subject to the terms of the Nematrian License Agreement.

[See table on website for latest list of these functions]

Charting Functions

[Nematrian website page: ChartingFunctions]

The Nematrian function library includes a range of tools that are useful for creating charts.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Code Generation Functions

[Nematrian website page: CodeGenerationFunctions]

The Nematrian function library includes a range of tools that are useful for creating programs that access the Nematrian function library.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Complex Numbers Functions

[Nematrian website page: ComplexNumbersFunctions]

The Nematrian function library includes a range of tools that are useful for carrying out complex number manipulations.

See also ComplexNumbersIntro.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Date Manipulation Functions

[Nematrian website page: DateFunctions]

The Nematrian function library includes a range of tools that provide common date manipulation functions as per a spreadsheet or programming language environment.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Derivative Pricing Web Service Functions

[Nematrian website page: DerivativePricingFunctions]

The Nematrian function library includes a range of tools for calculating prices and hedging parameters of different types of derivatives, given suitable user-supplied inputs to the relevant pricing algorithms.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Expression Evaluation Functions

[Nematrian website page: ExpressionEvaluationFunctions]

The Nematrian function library includes a range of tools that provide ways of evaluating and querying its own in-built function library using expressions (which can e.g. contain nested calls to different function library components within a given input).

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Finance Functions

[Nematrian website page: FinanceFunctions]

The Nematrian function library includes a range of tools that mimic finance orientated functions available in typical spreadsheet systems, e.g. DDB(), FV(), IPmt(), IRR(), MIRR(), NPer(), NPV(), Pmt(), PPmt(), PV(), Rate(), SLN() and SYD().

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

General / Miscellaneous Functions

[Nematrian website page: GeneralFunctions]

The Nematrian function library includes a range of tools. Most are specifically classified into one of a range of core topics. Those that are not otherwise classified include ones set out in the table below.

To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Life Insurance Functions

[Nematrian website page: LifeInsuranceFunctions]

The Nematrian function library includes a range of tools that are useful for carrying out life insurance orientated calculations.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

Disclaimer

Please note that no web functions currently *publicly* available through the Nematrian website are classified by Nematrian as being explicitly 'actuarial' in nature. For example, they do not necessarily adhere to relevant model standards imposed on actuarial work by the UK FRC's Actuarial Council. If you are seeking access to models that do explicitly adhere to these standards then please speak to your usual contact at Nematrian. Usage of the Nematrian website is subject to the terms of the Nematrian License Agreement.

[See table on website for latest list of these functions]

Markov Processes Functions

[Nematrian website page: MarkovProcessesFunctions]

The Nematrian function library includes a range of tools that can carry out analysis or simulation of Markov processes.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Mathematical Puzzles Functions

[Nematrian website page: MathematicalPuzzlesFunctions]

The Nematrian function library includes a range of tools that are useful for solving or analysing mathematical puzzles.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Matrix Manipulation Functions

[Nematrian website page: MatrixManipulationFunctions]

The Nematrian function library includes a range of tools that are useful for manipulating matrices.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Performance Measurement Functions

[Nematrian website page: PerformanceMeasurementFunctions]

The Nematrian function library includes a range of tools for carrying out performance measurement and attribution computations.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Portfolio Optimisation Functions

[Nematrian website page: PortfolioOptimisationFunctions]

The Nematrian function library includes a range of tools that are useful for portfolio optimisation.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Probability Distribution Functions

[Nematrian website page: ProbabilityDistributionsFunctions]

The Nematrian function library includes a range of tools that are useful for manipulating, analysing and fitting probability distributions, including calculating their moments and other characteristics and creating random variates drawn from them.

See here for a general introduction to the website's current probability distribution coverage. For details of the specific *DistributionName* input expected by these web functions (including those that involve shift and scale adjustments to the textbook formulation of a given probability distribution then please refer either to those pages or to MnRecognisedProbDists and MnProbDistParamNames.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Risk Management Functions

[Nematrian website page: RiskManagementFunctions]

The Nematrian function library includes a range of tools that are useful for carrying out risk management activities.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

As risk management draws on many different mathematical disciplines, many tools classified under other headings are also relevant to risk management. For example, the Nematrian function library has a comprehensive suite of probability distribution fitting algorithms which are classified under probability distribution functions, rather than under this heading.

[See table on website for latest list of these functions]

Set Manipulation Functions

[Nematrian website page: SetManipulationFunctions]

The Nematrian function library includes a range of tools that are useful for manipulating sets. For further details on set manipulation see SetManipulationIntro.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Solvency II orientated Functions

[Nematrian website page: SolvencyIIFunctions]

The Nematrian function library includes a range of tools that are useful for carrying out Solvency II orientated calculations.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

Disclaimer

Please note that no web functions currently *publicly* available through the Nematrian website are classified by Nematrian as being explicitly 'actuarial' in nature. For example, they do not necessarily adhere to relevant model standards imposed on actuarial work by the UK FRC's Actuarial Council. If you are seeking access to models that do explicitly adhere to these standards then please speak to your usual contact at Nematrian. Usage of the Nematrian website is subject to the terms of the Nematrian License Agreement.

[See table on website for latest list of these functions]

Sorting Functions

[Nematrian website page: SortingFunctions]

The Nematrian function library includes a range of tools that can be used to sort data.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Special Mathematical Functions

[Nematrian website page: SpecialFunctionsFunctions]

The Nematrian function library includes a range of tools that are useful for evaluating special mathematical functions.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Statistical Functions

[Nematrian website page: StatisticsFunctions]

The Nematrian function library includes a range of tools that are useful for carrying out statistical calculations.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

String Manipulation Functions

[Nematrian website page: StringFunctions]

The Nematrian function library includes a range of tools that provide common string manipulation functions as per a spreadsheet or programming language environment.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Tri-segmented Monte Carlo Demonstration Functions

[Nematrian website page: TSMCDemoFunctions]

The Nematrian function library includes a range of tools that are useful for demonstrating concepts underlying tri-segmented Monte Carlo.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

Unit Conversion Functions

[Nematrian website page: UnitConversionFunctions]

The Nematrian function library includes a range of tools that are useful for e.g. converting values from one set of units used for physical measurement (such as SI units) into another set of units.

The web functions available include ones set out in the table below. To see a complete list of functions currently available in the Nematrian online toolkit please go to list of types of available functions or complete list of individual functions.

[See table on website for latest list of these functions]

APPENDIX C: Individual Nematrian Web Service Functions (alphabetical)

Complete list of functions included in the Nematrian Function Library

[FunctionCompleteList]

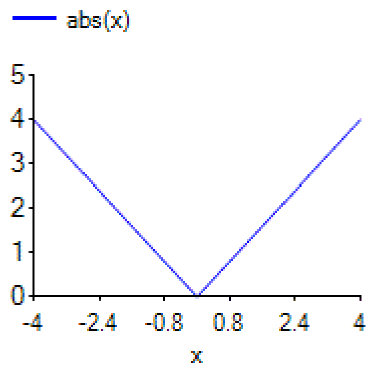
Set out below is a list of all of the functions currently included in the Nematrian Function Library.

To see functions grouped by type of activity to which they relate, please go to List of Nematrian Online Function Types and choose the function type in which you are interested.

Abs

[MnAbs]

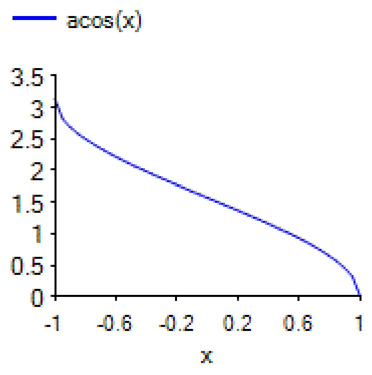
Returns the absolute value of a real number (i.e. $-x$ if $x < 0$ and x if $x \geq 0$)



Acos

[MnAcos]

Returns the (principal) arccosine of a real number, x ($-1 \leq x \leq 1$), i.e. the value y ($0 \leq y \leq \pi$) for which $x = \cos y$



Add
[MnAdd]

Returns the sum of two numbers (i.e. equivalent to the + operator), e.g. 4.1 + 5.1 = 9.2.

Provided for compatibility with the ADD function in certain spreadsheet systems.

AddConstantToSeries
[MnAddConstantToSeries]

Returns a series, y_i, calculated by adding to each element of an input series, x_i, a scalar constant, k, i.e.:

$$y_i = x_i + k$$

AppendArray
[MnAppendArray]

Appends one array onto (i.e. after) another, so if the first array consists of $x_1, x_2, \dots, x_n$ and the second array consists of $y_1, y_2, \dots, y_m$ then the output array has $n + m$ elements consisting of $x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_m$

ApplyGeometricSpreadToSeries
[MnApplyGeometricSpreadToSeries]

Returns an array $y(i)$ given an input array $x(i)$ and a spread s calculated as follows:

$$y(i) = \big(1 + x(i)\big)(1 + s) - 1$$

ApplyPriorityWaterfall
[MnApplyPriorityWaterfall]

Returns an array, $y(i)$, that corresponds to the fraction of particular priority bands ('tranches') that will be satisfied if the debt (liability) values, $x(i)$, to 'waterfall' through the priority ladder are in *ValuesToWaterfall*, the overall asset value available, A, is *AssetValue* (and if there is a surplus of *AssetValue* over the sum of the entries in *ValuesToWaterfall* then this is allocated pro-rata in line with $s(i)$, the *SurplusDistributionFractions*, to the different priorities in the order implied by *ValuesToWaterfall*).

Earlier entries are given higher priority in this cascading, i.e. all debts/liability values applicable to the earlier entries are honoured in full (to the extent that this is possible) before any later (i.e. lower priority) debts/liability values are honoured at all.

Thus (ignoring error trapping for division by zero):

$$y(i) = \frac{1}{x(i)}\left(\min\left(x(i), A - \sum_{j<i}^{n} x(j)\right) + s(i)\max\left(0, \left(A - \sum_{j=1}^{n} x(j)\right)\right)\right)$$

The function does not trap for negative values of $x(i)$ or $s(i)$.

Application of this type of computation can be important for financial structures where liabilities are tranched or otherwise honoured differently in the event of insufficient assets being present to honour all the liabilities in full.

ApplyPriorityWaterfall2

[MnApplyPriorityWaterfall2]

Returns an array, $y(i,j)$, that corresponds to the fraction of particular priority bands ('tranches') (j) owned by different investors (i) that will be satisfied if the debt (liability) values, $x(i,j)$, to 'waterfall' through the priority ladder are in *ValuesToWaterfall*, the overall asset value available, A, is *AssetValue* (and if there is a surplus of *AssetValue* over the sum of the entries in *ValuesToWaterfall* then this is allocated pro-rata in line with:

(a) $s(j)$, the *SurplusDistributionFractions*, to the different priorities in the order implied by *ValuesToWaterfall*, if *AllocateSurplusByPriorityLevel* is true
(b) $s(i)$, the *SurplusDistributionFractions*, to the different investor categories in the order implied by *ValuesToWaterfall*, if AllocateSurplusByPriorityLevel is false

Earlier entries are given higher priority in this cascading, i.e. all debts/liability values applicable to the earlier entries are honoured in full (to the extent that this is possible) before any later (i.e. lower priority) debts/liability values are honoured at all.

The function does not trap for negative values of $x(i)$ or $s(i)$.

Application of this type of computation can be important for financial structures where liabilities are tranched or otherwise honoured differently in the event of insufficient assets being present to honour all the liabilities in full.

ArithmeticallySpacedArray

[MnArithmeticallySpacedArray]

Returns a series whose elements are equally spaced (arithmetically), i.e. if size is n then returns a series whose elements x_i $(i = 1, \dots, n)$ are:

$$x_i = S + iI$$

where S is the starting value of the series and I is the increment between consecutive series elements.

ArrayAdd

[MnArrayAdd]

Returns an array z_i defined from two arrays, *Array1* and *Array2* where:

$$z_i = Array1_i + Array2_i$$

ArrayDivide

[MnArrayDivide]

Returns an array z_i defined from two arrays, *Array1* and *Array2* where:

$$z_i = \frac{Array1_i}{Array2_i}$$

ArrayMultiply

[MnArrayMultiply]

Returns an array z_i defined from two arrays, *Array1* and *Array2* where:

$$z_i = Array1_i \times Array2_i$$

ArraySubtract

[MnArraySubtract]

Returns an array z_i defined from two arrays, *Array1* and *Array2* where:

$$z_i = Array1_i - Array2_i$$

Asc

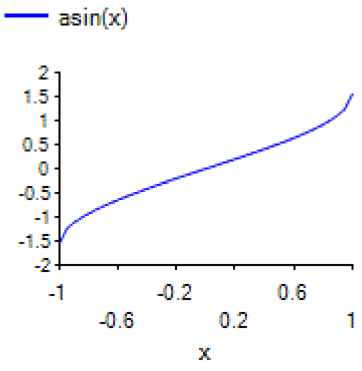
[MnAsc]

Returns the (Ascii) character code for the first character in *InputString*

Asin

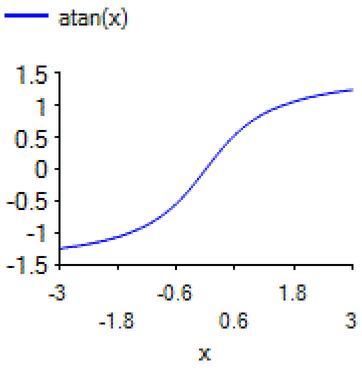
[MnAsin]

Returns the (principal) arcsine of a real number, x $(-1 \leq x \leq 1)$, i.e. the value y $(\pi/2 < y \leq \pi/2)$ for which $x = \sin y$



Atan
[MnAtan]

Returns the (principal) arctangent of a real number, x, i.e. the value y $(\pi/2 < y \leq \pi/2)$ for which $x = \tan y$



Atan2
[MnAtan2]

Returns the arctangent, or inverse tangent, θ, of the specified x- and y-coordinates. The arctangent is the angle from the x-axis to a line containing the origin (0, 0) and a point with coordinates (x, y), Differs from MnAtan as the latter returns an answer between $-\pi/2$ and $\pi/2$ so does not differentiate between (x, y) and $(-x, -y)$.

Note: the ordering of the input variables (first parameter: x, second parameter: y) is as per the ATAN2 function in Excel but the ATAN2 function in .NET has the parameter order reversed.

BaseNArrayToNonNegativeInteger
[MnBaseNArrayToNonNegativeInteger]

Converts into Integer form a number represented in digit form as per output of MnNonNegativeIntegerToBaseNArray, i.e. if input array is $x_1, \ldots, x_n$ and number base is n then calculates $\sum_{i=1}^{n} x_i n^{i-1}$.

BaseNArrayToString

[MnBaseNArrayToString]

Returns a string digit representation of an integer represented as an array, *BaseNArray*, with number base *nBase*, with order of elements of *BaseNArray* being in order of increasing power of *nBase* but with the characters of the string output being in order of decreasing power of *nBase* (in line with e.g. normal Arabic decimal and binary representation conventions).

BaseNString2Dec

[MnBaseNString2Dec]

Converts a string representing a number with a given radix between 2 and 36 to an (unsigned) integer, consecutive digits in the input string being represented by the characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y up to (for radix = 36) Z (lower case alphabetical characters also accepted). Similar to the inverse of Excel's BASE function.

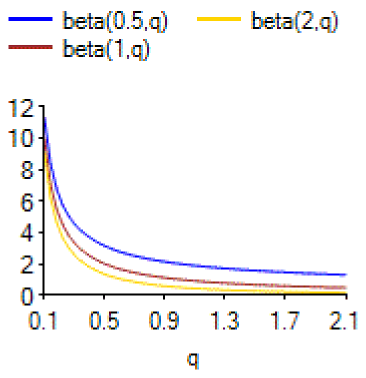
Beta

[MnBeta]

The beta function $B(p,q)$ is: defined as

$$B(p,q) = \int_0^1 t^{p-1}(1-t)^{q-1}dt \qquad (p,q>0)$$

Examples of some values of $B(p,q)$ are shown below:



Beta4DistCdf

[MnBeta4DistCdf]

Returns the cumulative distribution function of the Beta Distribution. Provided for compatibility with BETADIST and BETA.DIST functions in Microsoft Excel 2010 and later (with cumulative parameter set to true and with the *A* and *B* parameters supplied).

Equivalent to calling MnProbDistCdf using as the distribution parameter "beta4" but with A and B replaced by *A* and *B-A* respectively.

Beta4DistPdf

[MnBeta4DistPdf]

Returns the probability density function of the beta distribution. Provided for compatibility with BETA.DIST function in Microsoft Excel 2010 and later (with cumulative parameter set to false and with the *A* and *B* parameters supplied).

Equivalent to calling MnProbDistPdf using as the distribution parameter "beta4" but with *A* and *B* replaced by *A* and *B-A* respectively.

Beta4Inv

[MnBeta4Inv]

Returns the inverse distribution function (i.e. quantile function) of the beta distribution. Provided for compatibility with BETAINV and BETA.INV functions in Microsoft Excel 2010 and later (with cumulative parameter set to false and with the *A* and *B* parameters supplied).

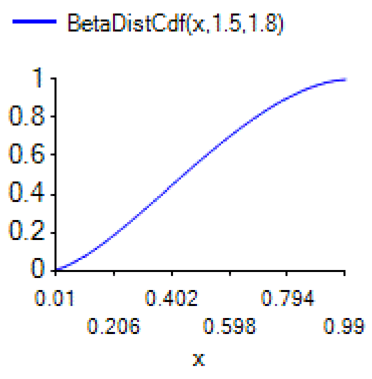
Equivalent to calling MnProbDistQuantile using as the distribution parameter "beta4" but with *A* and *B* replaced by *A* and *B-A* respectively.

BetaDistCdf

[MnBetaDistCdf]

Returns the cumulative distribution function of the (standard) beta distribution. Provided for compatibility with BETADIST and BETA.DIST functions in Microsoft Excel 2010 and later (with cumulative parameter set to true and with the *A* and *B* parameters not supplied or *A* = 0 and *B* = 1).

Equivalent to calling MnProbDistCdf using as the distribution parameter "beta".

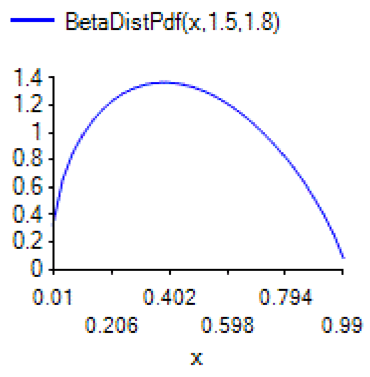


BetaDistPdf

[MnBetaDistPdf]

Returns the probability density function of the (standard) beta distribution. Provided for compatibility with BETA.DIST in Microsoft Excel 2010 and later (with cumulative parameter set to false and with the *A* and *B* parameters not supplied or *A* = 0 and *B* = 1).

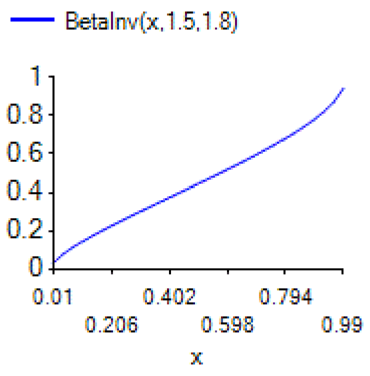
Equivalent to calling MnProbDistPdf using as the distribution parameter “beta”.



BetaInv
[MnBetaInv]

Returns the inverse distribution function (i.e. quantile function) of the (standard) beta distribution. Provided for compatibility with BETAINV and BETA.INV functions in Microsoft Excel 2010 and later (with cumulative parameter set to false and with the *A* and *B* parameters not supplied or *A* = 0 and *B* = 1).

Equivalent to calling MnProbDistQuantile.aspx using as the distribution parameter “beta”.



Bin2Dec
[MnBin2Dec]

Converts a string representing a number in binary form (i.e. base 2 with digits being 0 and 1) to an (unsigned) integer. Similar to Excel’s BIN2DEC function but accepting higher input values and rejecting negative input values.

BinaryNumerals

[MnBinaryNumerals]

Returns a string array containing a list of the binary numerals recognised by the site in ascending order.

BinomialCdf

[MnBinomialCdf]

Returns the cumulative distribution function for a binomial distribution $B(n,p)$, i.e. a distribution with $Pr(K = k)$ where

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

Provided for compatibility with BINOM.DIST and BINOMDIST in Microsoft Excel 2010 and later (with cumulative parameter set to true).

Equivalent to calling MnProbDistCdf using as the distribution parameter “binomial”.

BinomialPmf

[MnBinomialPmf]

Returns the probability mass function for $k \in \{0,1,\dots,n\}$ for a binomial distribution $B(n,p)$, i.e. $Pr(K = k)$ where

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

Provided for compatibility with BINOM.DIST and BINOMDIST in Microsoft Excel 2010 and later (with cumulative parameter set to false).

Equivalent to calling MnProbDistPmf using as the distribution parameter “binomial”.

BitAnd

[MnBitAnd]

Carries out a bit-wise AND of two numbers in string binary form, i.e.: for each corresponding digit separately carry out the following::

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

Similar to Microsoft Excel's BITAND function.

BitLShift

[MnBitLShift]

Shifts an (unsigned) integer represented in string binary format to the left (i.e. adds zeros to the least significant end if *BitsToShift* is positive or removes digits from that end if *BitsToShift* is negative).

Similar to Microsoft Excel's BITLSHIFT function.

BitOr

[MnBitOr]

Carries out a bit-wise OR of two numbers in string binary form, i.e.: for each corresponding digit separately carry out the following:

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

Similar to Microsoft Excel's BITOR function

BitRShift

[MnBitRShift]

Shifts an (unsigned) integer represented in string binary format to the right (i.e. adds zeros to the least significant end if *BitsToShift* is negative or removes them if *BitsToShift* is positive).

Similar to Microsoft Excel's BITRSHIFT function.

BitXor

[MnBitXor]

Carries out a bit-wise XOR of two numbers in string binary form, , i.e.: for each corresponding digit separately carry out the following:

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0

Similar to Microsoft Excel's BITXOR function

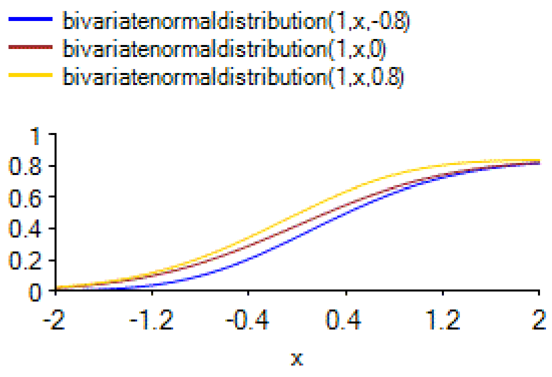
BivariateNormalDistribution

[MnBivariateNormalDistribution]

Returns $B(z_1, z_2)$, the cumulative distribution function of a standard bivariate normal distribution with correlation ρ, calculated using quadrature of a univariate integral as per Drezner and Wesolowsky (1990).

The algorithm uses an accelerated convergence approach to improve the accuracy of the calculation of the above integral.

Examples of some values of $B(z_1, z_2, \rho)$ are shown below:



BlendedPCAICA

[MnBlendedPCAICA]

Returns an array of mn entries corresponding to the the (zero mean) series corresponding to the m blended principal components/independent components of m different data series of n observations, ordered so that the signals with the largest importance criteria are first. The first n entries of the array returns correspond to the elements of the signal with the largest importance criterion etc.

The blended approach uses the following importance criterion:

$$y = a\kappa + (1-a)s(1+c\kappa)$$

where s is the standard deviation of the potential signal, κ is its kurtosis, and where the signals are normalised so that the squares of the betas of the different input series to the relevant signal add to unity.

See IndependentComponentsAnalysis for further details.

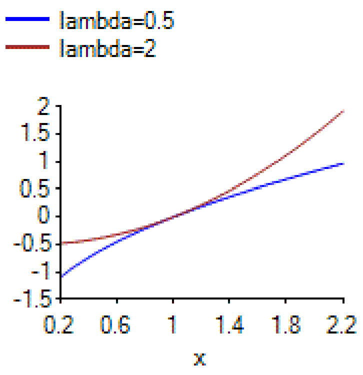
BoxCoxSeriesTransform

[MnBoxCoxSeriesTransform]

Returns a series, y_i, derived by applying the Box-Cox transform to an input series, x_i, i.e.:

$$y_i = \frac{{x_i}^{\lambda} - 1}{\lambda}$$

Example values of the transform are shown below:



BrowserFeatureDescriptions

[MnBrowserFeatureDescriptions]

Returns a list of descriptions of each recognised browser feature (such as browser type or screen width) that can be returned by MnMyBrowserFeature or that is returned by MnMyBrowserFeatures. MnMyBrowserFeature returns the value of a specific feature whilst MnMyBrowserFeatures returns a list of all available features recognised by the Nematrian website.

A list of recognised features (without descriptions) is also available using MnRecognisedBrowserFeatures.

BSBinaryCallCharm

[MnBSBinaryCallCharm]

Returns the Charm, i.e. sensitivity of delta to time, of a European binary call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Charm = \frac{\partial^2 V}{\partial S \partial t} = \frac{e^{-r(T-t)} N'(d_2)}{S\sigma\sqrt{T-t}} \left(r + \frac{1 - d_1 d_2}{2(T-t)} + \frac{d_2 (r-q)}{\sigma\sqrt{T-t}} \right)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryCallColour

[MnBSBinaryCallColour]

Returns the Colour, i.e. sensitivity of gamma to time, of a European binary call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Colour = \frac{\partial^3 V}{\partial S^2 \partial t} = -\frac{e^{-r(T-t)} N'(d_2)}{S^2 \sigma^2 (T-t)} \left(r d_1 + \frac{2d_1 + d_2}{2(T-t)} - \frac{(r-q)}{\sigma\sqrt{T-t}} - d_1 d_2 \left(\frac{d_1}{2(T-t)} - \frac{r-q}{\sigma\sqrt{T-t}} \right) \right)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryCallDelta

[MnBSBinaryCallDelta]

Returns the Delta, i.e. sensitivity of price to underlying, of a European binary call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Delta = \frac{\partial V}{\partial S} = \frac{e^{-r(T-t)}N'(d_2)}{S\sigma\sqrt{T-t}}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryCallGamma

[MnBSBinaryCallGamma]

Returns the Gamma, i.e. sensitivity of delta to underlying, of a European binary call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Gamma = \frac{\partial^2 V}{\partial S^2} = -\frac{e^{-r(T-t)}d_1 N'(d_2)}{S^2\sigma^2(T-t)}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryCallPayoff

[MnBSBinaryCallPayoff]

Returns the Payoff of a European binary call option, i.e.:

$$Payoff = \begin{cases} 1 & \text{if } S > K \\ 0 & \text{if } S \leq K \end{cases}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryCallPrice

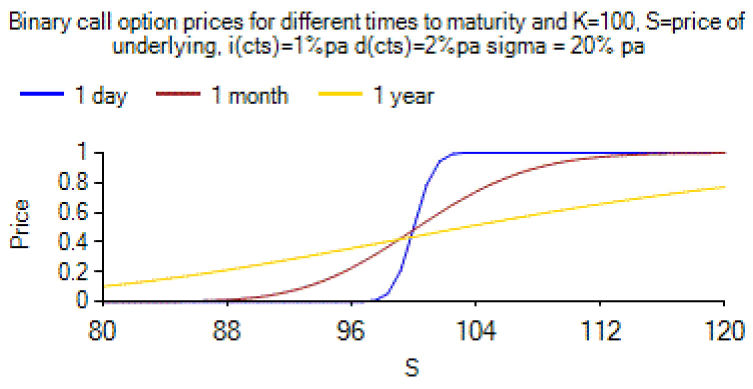
[MnBSBinaryCallPrice]

Returns the Price (i.e. value) of a European binary call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Price = V = e^{-r(T-t)}N(d_2)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

Example values for a range of times to maturity are:



BSBinaryCallRhoDividend

[MnBSBinaryCallRhoDividend]

Returns the Rho[Dividend], i.e. sensitivity of price to dividend yield, of a European binary call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Rho(Dividend) = \frac{\partial V}{\partial q} = -(T-t)e^{-r(T-t)}\, N(d_2)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryCallRhoInterest

[MnBSBinaryCallRhoInterest]

Returns the Rho[Interest], i.e. sensitivity of price to interest rate, of a European binary call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Rho(Interest) = \frac{\partial V}{\partial r} = -(T-t)e^{-r(T-t)}\, N(d_2) + \frac{\sqrt{T-t}}{\sigma} e^{-r(T-t)} N'(d_2)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryCallSpeed

[MnBSBinaryCallSpeed]

Returns the Speed, i.e. sensitivity of gamma to underlying, of a European binary call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Speed = \frac{\partial^3 V}{\partial S^3} = -\frac{e^{-r(T-t)} N'(d_2)}{\sigma^2 S^3 (T-t)} \left(-2d_1 + \frac{1-d_1 d_2}{\sigma\sqrt{T-t}}\right)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryCallTheta

[MnBSBinaryCallTheta]

Returns the Theta, i.e. sensitivity of price to time, of a European binary call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Theta = \frac{\partial V}{\partial t} = re^{-r(T-t)}N(d_2) + e^{-r(T-t)}N'(d_2)\left(\frac{d_1}{2(T-t)} - \frac{r-q}{\sigma\sqrt{T-t}}\right)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryCallVanna

[MnBSBinaryCallVanna]

Returns the Vanna, i.e. sensitivity of delta to volatility, of a European binary call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Vanna = \frac{\partial^2 V}{\partial S \partial \sigma} = -\frac{e^{-r(T-t)}N'(d_2)(1-d_1 d_2)}{S\sigma^2\sqrt{T-t}}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryCallVega

[MnBSBinaryCallVega]

Returns the Vega, i.e. sensitivity of price to volatility, of a European binary call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Vega = \frac{\partial V}{\partial \sigma} = e^{-r(T-t)}N'(d_2)\frac{d_1}{\sigma}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryPutCharm

[MnBSBinaryPutCharm]

Returns the Charm, i.e. sensitivity of delta to time, of a European binary put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Charm = \frac{\partial^2 V}{\partial S \partial t} = -\frac{e^{-r(T-t)}N'(-d_2)}{S\sigma\sqrt{T-t}}\left(r + \frac{1-d_1 d_2}{2(T-t)} + \frac{d_2(r-q)}{\sigma\sqrt{T-t}}\right)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryPutColour

[MnBSBinaryPutColour]

Returns the Colour, i.e. sensitivity of gamma to time, of a European binary put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Colour = \frac{\partial^3 V}{\partial S^2 \partial t} = \frac{e^{-r(T-t)}N'(-d_2)}{\sigma^2 S^2 (T-t)}\left(rd_1 + \frac{2d_1 + d_2}{2(T-t)} - \frac{(r-q)}{\sigma\sqrt{T-t}} - d_1 d_2 \left(\frac{d_1}{2(T-t)} - \frac{r-q}{\sigma\sqrt{T-t}}\right)\right)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryPutDelta

[MnBSBinaryPutDelta]

Returns the Delta, i.e. sensitivity of price to underlying, of a European binary put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Delta = \frac{\partial V}{\partial S} = -\frac{e^{-r(T-t)}N'(-d_2)}{S\sigma\sqrt{T-t}}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryPutGamma

[MnBSBinaryPutGamma]

Returns the Gamma, i.e. sensitivity of delta to underlying, of a European binary put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Gamma = \frac{\partial^2 V}{\partial S^2} = \frac{e^{-r(T-t)}d_1 N'(-d_2)}{S^2\sigma^2(T-t)}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryPutPayoff

[MnBSBinaryPutPayoff]

Returns the Payoff of a European binary put option, i.e.:

$$Payoff = \begin{cases} 1 & \text{if } S < K \\ 0 & \text{if } S \geq K \end{cases}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryPutPrice

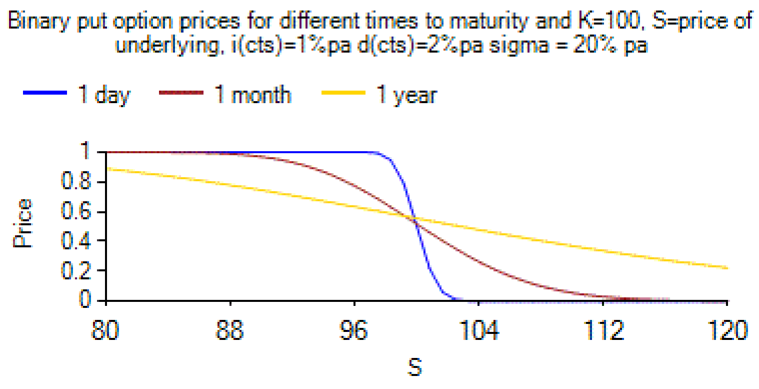
[MnBSBinaryPutPrice]

Returns the Price (i.e. value) of a European binary put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Price = V = e^{-r(T-t)} N(-d_2)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

Example values for a range of times to maturity are:



BSBinaryPutRhoDividend

[MnBSBinaryPutRhoDividend]

Returns the Rho[Dividend], i.e. sensitivity of price to dividend yield, of a European binary put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Rho(Dividend) = \frac{\partial V}{\partial q} = (T-t)e^{-r(T-t)}\, N(-d_2)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryPutRhoInterest

[MnBSBinaryPutRhoInterest]

Returns the Rho[Interest], i.e. sensitivity of price to interest rate, of a European binary put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Rho(Interest) = \frac{\partial V}{\partial r} = -(T-t)e^{-r(T-t)}\, N(-d_2) - \frac{\sqrt{T-t}}{\sigma} e^{-r(T-t)} N'(-d_2)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryPutSpeed

[MnBSBinaryPutSpeed]

Returns the Speed, i.e. sensitivity of gamma to underlying, of a European binary put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Speed = \frac{\partial^3 V}{\partial S^3} = \frac{e^{-r(T-t)}N'(-d_2)}{\sigma^2 S^3 (T-t)}\left(-2d_1 + \frac{1-d_1 d_2}{\sigma\sqrt{T-t}}\right)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryPutTheta

[MnBSBinaryPutTheta]

Returns the Theta, i.e. sensitivity of price to time, of a European binary put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Theta = \frac{\partial V}{\partial t} = re^{-r(T-t)}N(-d_2) - e^{-r(T-t)}N'(-d_2)\left(\frac{d_1}{2(T-t)} - \frac{r-q}{\sigma\sqrt{T-t}}\right)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryPutVanna

[MnBSBinaryPutVanna]

Returns the Vanna, i.e. sensitivity of delta to volatility, of a European binary put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Vanna = \frac{\partial^2 V}{\partial S \partial \sigma} = \frac{e^{-r(T-t)}N'(-d_2)(1-d_1 d_2)}{S\sigma^2\sqrt{T-t}}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryPutVega

[MnBSBinaryPutVega]

Returns the Vega, i.e. sensitivity of price to volatility, of a European binary put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Vega = \frac{\partial V}{\partial \sigma} = e^{-r(T-t)}N'(-d_2)\frac{d_1}{\sigma}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSBinaryPutVolga

[MnBSBinaryPutVolga]

Returns the Volga (or Vomma), i.e. sensitivity of vega to volatility, of a European binary put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Volga = \frac{\partial^2 V}{\partial \sigma^2} = \frac{e^{-r(T-t)} N'(-d_2)(d_1^2 d_2 - d_1 - d_2)}{\sigma^2}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSCall

[MnBSCall]

Returns the price C of a European call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies.

If the current price of the underlying is S, the strike (i.e. exercise) price of the option is K, the continuously compounded rate of interest is r, the continuously compounded annualised dividend yield (assumed continuous not discrete) is q, time now (in years) is t, time at expiry is T and the (annualised) implied volatility is σ then the price of such an option is given by:

$$C = Se^{-q(T-t)} N(d_1) - Ke^{-r(T-t)} N(d_2)$$

where $N(z)$ is the cumulative unit normal distribution function (see MnCumulativeNormal), i.e.

$$N(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-t^2/2} dt = \frac{1}{2} + \frac{1}{2} \operatorname{erf}\left(\frac{z}{\sqrt{2}}\right)$$

and

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r - q + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$$

$$d_2 = d_1 - \sigma\sqrt{T-t} = \frac{\ln\left(\frac{S}{K}\right) + \left(r - q - \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$$

BSCallCharm

[MnBSCallCharm]

Returns the Charm, i.e. sensitivity of delta to time, of a European call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Charm = \frac{\partial^2 V}{\partial S \partial t} = qe^{-q(T-t)} N(d_1) + e^{-q(T-t)} N'(d_1) \left(\frac{d_2}{2(T-t)} - \frac{r-q}{\sigma\sqrt{T-t}}\right)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSCallColour

[MnBSCallColour]

Returns the Colour, i.e. sensitivity of gamma to time, of a European call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Colour = \frac{\partial^3 V}{\partial S^2 \partial t} = \frac{e^{-q(T-t)}N'(d_1)}{S\sigma\sqrt{T-t}}\left(q + \frac{1-d_1 d_2}{2(T-t)} + \frac{d_1(r-q)}{\sigma\sqrt{T-t}}\right)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSCallDelta

[MnBSCallDelta]

Returns the Delta, i.e. sensitivity of price to underlying, of a European call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Delta = \frac{\partial V}{\partial S} = e^{-q(T-t)}N(d_1)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSCallGamma

[MnBSCallGamma]

Returns the Gamma, i.e. sensitivity of delta to underlying, of a European call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Gamma = \frac{\partial^2 V}{\partial S^2} = \frac{e^{-q(T-t)}N'(d_1)}{S\sigma\sqrt{T-t}}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSCallPayoff

[MnBSCallPayoff]

Returns the Payoff of a European call option, i.e.:

$$Payoff = \max(S-K, 0)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSCallPrice

[MnBSCallPrice]

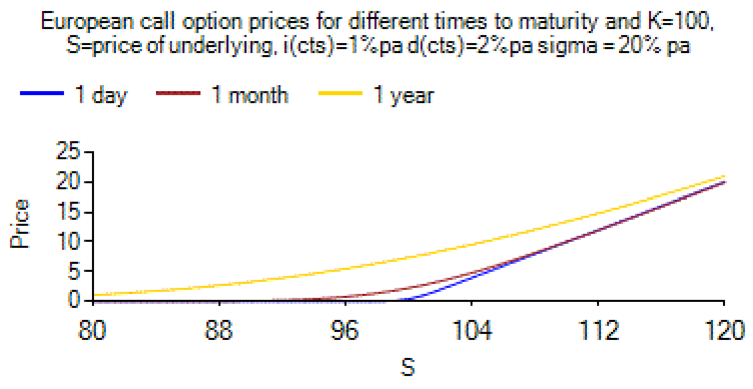
Returns the Price (i.e. value) of a European call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Price = V = Se^{-q(T-t)}N(d_1) - Ke^{-r(T-t)}N(d_2)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

N.B. Returns the same as MnBSCall

Example values for a range of times to maturity are:



BSCallRhoDividend

[MnBSCallRhoDividend]

Returns the Rho[Dividend], i.e. sensitivity of price to dividend yield, of a European call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Rho(Dividend) = \frac{\partial V}{\partial q} = -S(T-t)e^{-q(T-t)}N(d_1)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSCallRhoInterest

[MnBSCallRhoInterest]

Returns the Rho[Interest], i.e. sensitivity of price to interest rate, of a European call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Rho(Interest) = \frac{\partial V}{\partial r} = K(T-t)e^{-r(T-t)}\,N(d_2)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSCallSpeed

[MnBSCallSpeed]

Returns the Speed, i.e. sensitivity of gamma to underlying, of a European call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Speed = \frac{\partial^3 V}{\partial S^3} = -\frac{e^{-q(T-t)}N'(d_1)\left(d_1 + \sigma\sqrt{T-t}\right)}{\sigma^2 S^2 (T-t)}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSCallTheta

[MnBSCallTheta]

Returns the Theta, i.e. sensitivity of price to time, of a European call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Theta = \frac{\partial V}{\partial t} = -\frac{\sigma S e^{-q(T-t)}N'(d_1)}{2\sqrt{T-t}} + qSe^{-q(T-t)}N(d_1) - rKe^{-r(T-t)}N(d_2)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSCallVanna

[MnBSCallVanna]

Returns the Vanna, i.e. sensitivity of delta to volatility, of a European call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Vanna = \frac{\partial^2 V}{\partial S \partial \sigma} = -\frac{d_2 e^{-q(T-t)}N'(d_1)}{\sigma}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSCallVega

[MnBSCallVega]

Returns the Vega, i.e. sensitivity of price to volatility, of a European call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Vega = \frac{\partial V}{\partial \sigma} = Se^{-q(T-t)}N'(d_1)\sqrt{T-t}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSCallVolga

[MnBSCallVolga]

Returns the Volga (or Vomma), i.e. sensitivity of vega to volatility, of a European call option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Volga = \frac{\partial^2 V}{\partial \sigma^2} = \frac{d_1 d_2 S e^{-q(T-t)}N'(d_1)\sqrt{T-t}}{\sigma}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSPut

[MnBSPut]

Returns the price P of a European put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies.

If the current price of the underlying is S, the strike (i.e. exercise) price of the option is K, the continuously compounded rate of interest is r, the continuously compounded annualised dividend yield (assumed continuous not discrete) is q, time now (in years) is t, time at expiry is T and the (annualised) implied volatility is σ then the price of such an option is given by:

$$P = -Se^{-q(T-t)}N(-d_1) + Ke^{-r(T-t)}N(-d_2)$$

where $N(z)$ is the cumulative unit normal distribution function (see MnCumulativeNormal), i.e.

$$N(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-t^2/2} dt = \frac{1}{2} + \frac{1}{2}\operatorname{erf}\left(\frac{z}{\sqrt{2}}\right)$$

and:

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r - q + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$$

$$d_2 = d_1 - \sigma\sqrt{T-t} = \frac{\ln\left(\frac{S}{K}\right) + \left(r - q - \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$$

BSPutCharm

[MnBSPutCharm]

Returns the Charm, i.e. sensitivity of delta to time, of a European put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Charm = \frac{\partial^2 V}{\partial S \partial t} = -qe^{-q(T-t)}N(-d_1) + e^{-q(T-t)}N'(-d_1)\left(\frac{d_2}{2(T-t)} - \frac{r-q}{\sigma\sqrt{T-t}}\right)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSPutColour

[MnBSPutColour]

Returns the Colour, i.e. sensitivity of gamma to time, of a European put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Colour = \frac{\partial^3 V}{\partial S^2 \partial t} = \frac{e^{-q(T-t)}N'(-d_1)}{S\sigma\sqrt{T-t}}\left(q + \frac{1-d_1 d_2}{2(T-t)} + \frac{d_1(r-q)}{\sigma\sqrt{T-t}}\right)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSPutDelta

[MnBSPutDelta]

Returns the Delta, i.e. sensitivity of price to underlying, of a European put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Delta = \frac{\partial V}{\partial S} = -e^{-q(T-t)}N(-d_1)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSPutGamma

[MnBSPutGamma]

Returns the Gamma, i.e. sensitivity of delta to underlying, of a European put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Gamma = \frac{\partial^2 V}{\partial S^2} = \frac{e^{-q(T-t)}N'(-d_1)}{S\sigma\sqrt{T-t}}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSPutPayoff

[MnBSPutPayoff]

Returns the Payoff of a European put option, i.e.:

$$Payoff = \max(K-S, 0)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSPutPrice

[MnBSPutPrice]

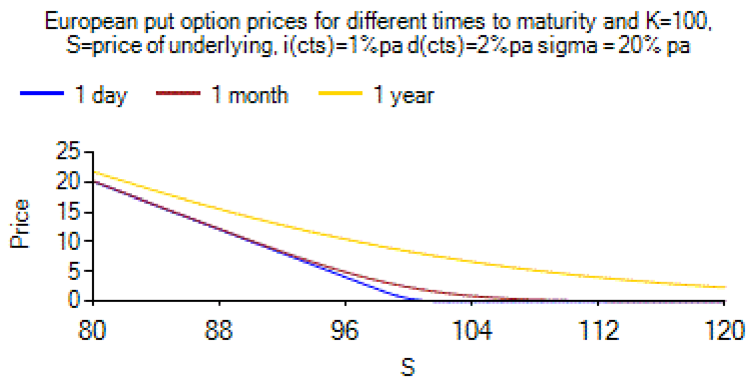
Returns the Price (i.e. value) of a European put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Price = V = -Se^{-q(T-t)}N(-d_1) + Ke^{-r(T-t)}N(-d_2)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

N.B. Returns the same as MnBSPut.

Example values for a range of times to maturity are:



BSPutRhoDividend

[MnBSPutRhoDividend]

Returns the Rho[Dividend], i.e. sensitivity of price to dividend yield, of a European put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Rho(Dividend) = \frac{\partial V}{\partial q} = -S(T-t)e^{-q(T-t)}N(-d_1)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSPutRhoInterest

[MnBSPutRhoInterest]

Returns the Rho[Interest], i.e. sensitivity of price to interest rate, of a European put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Rho(Interest) = \frac{\partial V}{\partial r} = -K(T-t)e^{-r(T-t)}\,N(-d_2)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSPutSpeed

[MnBSPutSpeed]

Returns the Speed, i.e. sensitivity of gamma to underlying, of a European put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Speed = \frac{\partial^3 V}{\partial S^3} = -\frac{e^{-q(T-t)}N'(-d_1)\left(d_1 + \sigma\sqrt{T-t}\right)}{\sigma^2 S^2 (T-t)}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSPutTheta

[MnBSPutTheta]

Returns the Theta, i.e. sensitivity of price to time, of a European put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Theta = \frac{\partial V}{\partial t} = -\frac{\sigma S e^{-q(T-t)} N'(-d_1)}{2\sqrt{T-t}} - qSe^{-q(T-t)}N(-d_1) + rKe^{-r(T-t)}N(-d_2)$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSPutVanna

[MnBSPutVanna]

Returns the Vanna, i.e. sensitivity of delta to volatility, of a European put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Vanna = \frac{\partial^2 V}{\partial S \partial \sigma} = -\frac{d_2 e^{-q(T-t)} N'(-d_1)}{\sigma}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSPutVega

[MnBSPutVega]

Returns the Vega, i.e. sensitivity of price to volatility, of a European put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Vega = \frac{\partial V}{\partial \sigma} = Se^{-q(T-t)}N'(-d_1)\sqrt{T-t}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

BSPutVolga

[MnBSPutVolga]

Returns the Volga (or Vomma), i.e. sensitivity of vega to volatility, of a European put option assuming that the (generalised) Black-Scholes (i.e. Garman-Kohlhagen) pricing formula applies, i.e.:

$$Volga = \frac{\partial^2 V}{\partial \sigma^2} = \frac{d_1 d_2 S e^{-q(T-t)} N'(-d_1)\sqrt{T-t}}{\sigma}$$

See Black-Scholes option pricing greeks for further details of notation and (other) option greeks.

CAbs

[MnCAbs]

Returns $|z|$, the modulus or absolute value of a complex number, z. If z has the form $x + iy$ where x and y are real then:

$$|z| = \sqrt{x^2 + y^2}$$

For details of how to pass complex numbers to and from the Nematrian website please see Complex Numbers Introduction.

CAcos

[MnCAcos]

Returns $\arccos(z)$, the principal value of the inverse cosine of a complex number z, i.e. the principal value of q for which $\cos(q) = z$. The Nematrian website uses the following calculation, see Bronshtein and Semendyayev (1985):

$$\arccos(z) = -i \log\left(z + \sqrt{z^2 - 1}\right)$$

For details of how to pass complex numbers to and from the Nematrian website and on principal values of complex-valued functions please see Complex Numbers Introduction.

CAcosh

[MnCAcosh]

Returns $\operatorname{arcosh}(z)$, the principal value of the inverse hyperbolic cosine of a complex number z, i.e. the principal value of q for which $\cosh(q) = z$. The Nematrian website uses the following calculation, see Bronshtein and Semendyayev (1985):

$$\operatorname{arcosh}(z) = \log\left(z + \sqrt{z^2 - 1}\right)$$

For details of how to pass complex numbers to and from the Nematrian website and on principal values of complex-valued functions please see Complex Numbers Introduction.

CAcot

[MnCAcot]

Returns $\operatorname{arccot}(z)$, the principal value of the inverse cotangent of a complex number z, i.e. the principal value of q for which $\cot(q) = z$. The Nematrian website uses the following calculation, see Bronshtein and Semendyayev (1985):

$$\operatorname{arccot}(z) = -\frac{1}{2i} \log\left(\frac{iz + 1}{iz - 1}\right)$$

For details of how to pass complex numbers to and from the Nematrian website and on principal values of complex-valued functions please see Complex Numbers Introduction.

CAcoth

[MnCAcoth]

Returns $\mathrm{arcoth}(z)$, the principal value of the inverse hyperbolic cotangent of a complex number z, i.e. the principal value of q for which $\coth(q) = z$. The Nematrian website uses the following calculation, see Bronshtein and Semendyayev (1985):

$$\mathrm{arcot}(z) = \frac{1}{2}\log\left(\frac{z+1}{z-1}\right)$$

For details of how to pass complex numbers to and from the Nematrian website and on principal values of complex-valued functions please see Complex Numbers Introduction.

CAdd

[MnCAdd]

Returns $z_1 + z_2$, the sum of two complex numbers. If z_1 and z_2 have the form $x_1 + iy_1$ and $x_2 + iy_2$ respectively where $x_1,\ x_2,\ y_1,\ y_2,$ are real then:

$$z_1 + z_2 = (x_1 + x_2) + i(y_1 + y_2)$$

For details of how to pass complex numbers to and from the Nematrian website please see Complex Numbers Introduction.

CArg

[MnCArg]

Returns $\arg(z)$ the principal value of the argument of a complex number, z. If the complex number has the form $x + iy$ where x and y are real then:

$$\arg(z) = \arctan(y/x)$$

(N.B. the corresponding arctan function in Microsoft Excel or Visual Studio is ATAN2, rather than ATAN)

For details of how to pass complex numbers to and from the Nematrian website and on principal values of complex-valued functions please see Complex Numbers Introduction.

CartesianToPolar

[MnCartesianToPolar]

Returns an array (r,θ) which in polar coordinates corresponds to the point which has Cartesian coordinates as defined by the input array (x,y), i.e. so that $x = r\cos\theta$, $y = r\sin\theta$ and $-\pi < \theta \leq \pi$. This means that $r = \sqrt{x^2+y^2}$ and θ are as identified by MnAtan2.

CartesianToPolar2

[MnCartesianToPolar2]

Returns an array (r,θ) which in polar coordinates corresponds to the point which has Cartesian coordinates, x and y, i.e. so that $x = r\cos\theta$, $y = r\sin\theta$ and $-\pi < \theta \leq \pi$. This means that $r = \sqrt{x^2+y^2}$ and θ are as identified by MnAtan2.

CAsin

[MnCAsin]

Returns $\arcsin(z)$, the principal value of the inverse sine of a complex number z, i.e. the principal value of q for which $\sin(q) = z$. The Nematrian website uses the following calculation, see Bronshtein and Semendyayev (1985):

$$\arcsin(z) = -i\log\left(iz + \sqrt{1-z^2}\right)$$

For details of how to pass complex numbers to and from the Nematrian website and on principal values of complex-valued functions please see Complex Numbers Introduction.

CAsinh

[MnCAsinh]

Returns $\operatorname{arsinh}(z)$, the principal value of the inverse hyperbolic sine of a complex number z, i.e. the principal value of q for which $\sinh(q) = z$. The Nematrian website uses the following calculation, see Bronshtein and Semendyayev (1985):

$$\operatorname{arsinh}(z) = \log\left(z + \sqrt{z^2+1}\right)$$

For details of how to pass complex numbers to and from the Nematrian website and on principal values of complex-valued functions please see Complex Numbers Introduction.

CAtan

[MnCAtan]

Returns $\operatorname{artan}(z)$, the principal value of the inverse tangent of a complex number z, i.e. the principal value of q for which $\tan(q) = z$. The Nematrian website uses the following calculation, see Bronshtein and Semendyayev (1985):

$$\arcsin(z) = \frac{1}{2i}\log\left(\frac{1+iz}{1-iz}\right)$$

For details of how to pass complex numbers to and from the Nematrian website and on principal values of complex-valued functions please see Complex Numbers Introduction.

CAtanh

[MnCAtanh]

Returns $\operatorname{artanh}(z)$, the principal value of the inverse hyperbolic tangent of a complex number z, i.e. the principal value of q for which $\tanh(q) = z$. The Nematrian website uses the following calculation, see Bronshtein and Semendyayev (1985):

$$\operatorname{artanh}(z) = \frac{1}{2}\log\left(\frac{1+z}{1-z}\right)$$

For details of how to pass complex numbers to and from the Nematrian website and on principal values of complex-valued functions please see Complex Numbers Introduction.

CCCall

[MnCCCall]

Returns the price, *C*, of a European-style call option under the 'Cost of Capital' Pricing Model, i.e.:

$$C = S\left(e^{-q(T-t)} - e^{-q_a(T-t)}\right) + Se^{-q_a(T-t)}N(d_1) - Ke^{-r_a(T-t)}N(d_2)$$

where

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r_a - q_a + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$$
$$d_2 = d_1 - \sigma\sqrt{T-t} = \frac{\ln\left(\frac{S}{K}\right) + \left(r_a - q_a - \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$$

CConjugate

[MnCConjugate]

Returns $\bar{z}$ the complex conjugate of a complex number, z. If z has the form $x + iy$ where x and y are real then:

$$\bar{z} = x - iy$$

For details of how to pass complex numbers to and from the Nematrian website please see Complex Numbers Introduction.

CCos

[MnCCos]

Returns $\cos(z)$, the cosine of a complex number, z. If z has the form $x + iy$ where x and y are real then:

$$\cos(z) = (\cos x \cosh y) - i(\sin x \sinh y)$$

For details of how to pass complex numbers to and from the Nematrian website please see Complex Numbers Introduction.

CCosh

[MnCCosh]

Returns $\cosh(z)$, the hyperbolic cosine of a complex number, z. If z has the form $x + iy$ where x and y are real then:

$$\cosh(z) \equiv \frac{e^z + e^{-z}}{2} = (\cosh x \cos y) - i(\sinh x \sin y)$$

For details of how to pass complex numbers to and from the Nematrian website please see Complex Numbers Introduction.

CCot

[MnCCot]

Returns $\cot(z)$, the cotangent of a complex number, z.

$$\cot(z) = \frac{1}{\tan z} = \frac{\cos z}{\sin z}$$

For details of how to pass complex numbers to and from the Nematrian website please see Complex Numbers Introduction.

CCoth

[MnCCoth]

Returns $\coth(z)$, the hyperbolic cotangent of a complex number, z.

$$\coth(z) = \frac{1}{\tanh z} = \frac{e^z + e^{-z}}{e^z - e^{-z}}$$

For details of how to pass complex numbers to and from the Nematrian website please see Complex Numbers Introduction.

CCPut

[MnCCPut]

Returns the price, *P*, of a European-style call option under the 'Cost of Capital' Pricing Model, i.e.:

$$P = K\left(e^{-r(T-t)} - e^{-r_a(T-t)}\right) - Se^{-q_a(T-t)}N(-d_1) + Ke^{-r_a(T-t)}N(-d_2)$$

where

$$d_1 = \frac{\ln\left(\frac{S}{K}\right) + \left(r_a - q_a + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$$
$$d_2 = d_1 - \sigma\sqrt{T-t} = \frac{\ln\left(\frac{S}{K}\right) + \left(r_a - q_a - \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$$

CCsc

[MnCCsc]

Returns $\mathrm{cosec}(z)$, the cosecant of a complex number, z. The cosecant can be calculated as follows. Given this simple relationship, it is used rather infrequently (e.g. in astronomy), so no equivalent inverse function is provided by the Nematrian website:

$$\sec(z) = \frac{1}{\sin z}$$

For details of how to pass complex numbers to and from the Nematrian website please see Complex Numbers Introduction.

CDivide

[MnCDivide]

Returns z_1/z_2, the result of dividing one complex number, z_1 by another one, z_2 (as long as $|z_2| > 0$). If z_1 and z_2 have the form $x_1 + iy_1$ and $x_2 + iy_2$ respectively where $x_1,\ x_2,\ y_1,\ y_2,$ are real then:

$$\frac{z_1}{z_2} = \frac{x_1x_2 + y_1y_2 + i(x_2y_1 - x_1y_2)}{x_2^2 + y_2^2}$$

For details of how to pass complex numbers to and from the Nematrian website please see Complex Numbers Introduction.

Ceiling

[MnCeiling]

Rounds a number towards positive infinity, i.e. 7.1 goes to 8, -7.1 to -7

CellFormulaA1StyleRangeConcatenate

[MnCellFormulaA1StyleRangeConcatenate]

Returns text to create a formula that will concatenate the values in a range of spreadsheet cells, including a delimiter (i.e. a specific piece of text) between each entry.

StartCellRow and *StartCellCol* (both integers) define the top left hand corner of the range, the size of which is defined by *NoRows* and *NoCols* (also both integers). If *TransposeRange* is true then the resulting formula will cycle through the cells starting at the top left hand corner working downwards and then across, whereas if *TransposeRange* is false then the resulting formula works across then down. The string inserted between each cell value is *DelimiterString*. If *FrozenRows* is true then the formua created by this function includes absolute row cell references, e.g. A$1, whereas if *FrozenRows* is false then it creates relative row cell references, e.g. A1. *FrozenCols* does the same but for column cell references, e.g. $A1 versus A1.

This function is particularly useful for creating formulae that can be used within a WEBSERVICE call or equivalent, see e.g. Introduction to the Nematrian Spreadsheet Library.

CellReferenceA1Style

[MnCellReferenceA1Style]

Returns text corresponding to a reference to an individual cell, in A1 style.

CellRow and *CellCol* define the cell itself (e.g. *CellRow* = 3 and *CellCol* = 2 corresponds to B3). If *FrozenRows* is true then the formula created by this function includes absolute row cell references, e.g. B$3, whereas if *FrozenRows* is false then it creates relative row cell references, e.g. B3. *FrozenCols* does the same but for column cell references, e.g. $B3 versus B3.

CExp

[MnCExp]

Returns $\exp z$, the exponential of a complex number, z, also written e^z. If $z = x + iy$ where x and y are real then:

$$e^z = e^{x+iy} = e^x(\cos y + i \sin y)$$

For details of how to pass complex numbers to and from the Nematrian website please see Complex Numbers Introduction.

CGamma

[MnCGamma]

Returns $\Gamma(z)$, the value of the gamma function evaluated for a complex number, z. The gamma function, $\Gamma(z)$ is defined as:

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt$$

It can thought of as an extension of the factorial function, but offset by one, since it satisfies the following recurrence relationship: $\Gamma(z+1) = z\Gamma(z)$.

The Nematrian website calculates this value using a Lanczos approximation and a reflection formula, see MnGamma.

For details of how to pass complex numbers to and from the Nematrian website and on principal values of complex-valued functions please see Complex Numbers Introduction.

ChiDist

[MnChiDist]

Returns 1 minus the cumulative distribution function of the chi-squared distribution. Provided for compatibility with CHIDIST in Microsoft Excel 2010 and later.

Note CHIDIST = CHISQ.DIST.RT = 1 - CHISQ.DIST(cumulative=true) and CHISQ.DIST(cumulative=true) = MnChiSqDistCdf

Please also bear in mind that the Nematrian website functions relating to the chi-squared distribution accept non-integral values of *nu* whereas the Excel functions are limited to integral values.

Equivalent to calling 1 - MnProbDistCdf using as the distribution parameter “chi-squared”.

ChiInv

[MnChiInv]

Returns the inverse of MnChiDist (i.e. the quantile function of 1 - q) of the chi-squared distribution. Provided for compatibility with CHIINV in Microsoft Excel 2010 and later.

Note MnChiInv(q) = CHIINV(q) = CHISQ.INV.RT(q) = CHISQ.INV(1-q) and CHISQ.INV(q) = MnChiSqInv(q)

Please also bear in mind that the Nematrian website functions relating to the chi-squared distribution accept non-integral values of *nu* whereas the Excel functions are limited to integral values.

Equivalent to calling MnProbDistQuantile using as the distribution parameter “chi-squared” and entering 1-q for the input value.

ChiSqDistCdf

[MnChiSqDistCdf]

Returns the cumulative distribution function of the chi-squared distribution. Provided for compatibility with CHISQ.DIST (with cumulative parameter set to true) in Microsoft Excel 2010 and later.

Note CHIDIST = CHISQ.DIST.RT = 1 - CHISQ.DIST(cumulative=true) and CHISQ.DIST(cumulative=true) = MnChiSqDistCdf

Please also bear in mind that the Nematrian website functions relating to the chi-squared distribution accept non-integral values of *nu* whereas the Excel functions are limited to integral values.

Equivalent to calling MnProbDistCdf using as the distribution parameter “chi-squared”.

ChiSqDistPdf

[MnChiSqDistPdf]

Returns the probability density function of the chi-squared distribution. Provided for compatibility with CHISQ.DIST (with cumulative parameter set to true) in Microsoft Excel 2010 and later.

Please also bear in mind that the Nematrian website functions relating to the chi-squared distribution accept non-integral values of *nu* whereas the Excel functions are limited to integral values.

Equivalent to calling MnProbDistPdf using as the distribution parameter “chi-squared”.

ChiSqInv

[MnChiSqInv]

Returns the inverse (i.e. quantile function) of the chi-squared distribution. Provided for compatibility with CHISQ.INV function in Microsoft Excel 2010 and later.

Note MnChiInv(q) = CHIINV(q) = CHISQ.INV.RT(q) = CHISQ.INV(1-q) and CHISQ.INV(q) = MnChiSqInv(q)

Please also bear in mind that the Nematrian website functions relating to the chi-squared distribution accept non-integral values of *nu* whereas the Excel functions are limited to integral values.

Equivalent to calling MnProbDistQuantile using as the distribution parameter “chi-squared”.

ChiSqTest

[MnChiSqTest]

Returns the confidence level relating to applying a chi-squared test to some data, i.e. the (right tail) cumulative distribution function (MnChiDist) with inputs *x* = test statistic as per MnChiSqTestStatistic and number of degrees of freedom as per MnChiSqTestDoF.

ChiSqTestDoF

[MnChiSqTestDoF]

Returns the number of degrees of freedom for a chi-squared test, i.e.

- If NoRows = 1 (and NoCols > 1) then returns NoCols - 1

- If NoCols = 1 and NoRows > 1 then returns NoRows - 1

- If NoRows > 1 and NoCols > 1 then returns (NoRows - 1) * (NoCols - 1)

ChiSqTestStatistic

[MnChiSqTestStatistic]

Returns the test statistic for a chi-squared test, i.e:

$$\sum \frac{(Actual_i - Expected_i)^2}{Expected_i}$$

CholeskyDecomposition

[MnCholeskyDecomposition]

Returns the Cholesky decomposition, *L*, of a square matrix, *A*.

If *A* has real entries, is symmetric and is positive definite then this decomposition involves expressing it in the form $A = LL^T$ where *L* is a lower triangular matrix with strictly positive diagonal entries and L^T is its transpose. The entries of *L* are:

$$L_{j,j} = \sqrt{A_{j,j} - \sum_{k=1}^{j-1} L_{j,k}^2} \qquad L_{i,j} = \frac{1}{L_{j,j}}\left(A_{i,j} - \sum_{k=1}^{j-1} L_{i,k}L_{j,k}\right) \quad for\ i > j$$

Cholesky decomposition has two main uses:

(a) Suppose we draw vectors of independent normal random variables, x_i then $y_i = \sum_{j \le i} L_{j,i} x_j$ are vectors drawn from a multivariate normal distribution with covariance matrix V.

(b) A covariance matrix is non-negative definite. Perhaps the easiest way to test if a symmetric matrix is non-negative definite is to see if a Cholesky decomposition can be applied to it.

Chr

[MnChr]

Returns the character corresponding to the (Ascii) character code *CharCode*

CImag

[MnCImag]

Returns $Im(z)$, the imaginary part of a complex number, z. If z has the form $x + iy$ where x and y are real then:

$$Im(z) = y$$

For details of how to pass complex numbers to and from the Nematrian website and on principal values of complex-valued functions please see Complex Numbers Introduction.

CLog

[MnCLog]

Returns $\log(z)$, the principal value of the natural logarithm of a complex number, z.

For details of how to pass complex numbers to and from the Nematrian website and on principal values of complex-valued functions please see Complex Numbers Introduction.

CMultiply

[MnCMultiply]

Returns $z_1 z_2$, the product of two complex numbers. If z_1 and z_2 have the form $x_1 + iy_1$ and $x_2 + iy_2$ respectively where $x_1,\ x_2,\ y_1,\ y_2,$ are real then:

$$z_1 z_2 = (x_1 x_2 - y_1 y_2) + i(x_1 y_2 + x_2 y_2)$$

For details of how to pass complex numbers to and from the Nematrian website please see Complex Numbers Introduction.

Concat

[MnConcat]

Returns the concatenation of two strings (i.e. equivalent to the & operator).

Provided for compatibility with the CONCAT function in Google Drive.

Concatenate

[MnConcatenate]

Returns the result of concatenating all of the entries of *InputStringArray*.

ConfidenceLevelKurtApproxIfNormal

[MnConfidenceLevelKurtApproxIfNormal]

Returns the (large sample) confidence point for the (sample) skew of a series of observations, given the null hypothesis that the observations are coming from a normal distribution. See e.g. Press et al. (2007). Is calculated from the quantile point of a normal distribution with standard deviation $\sqrt{24/n}$ where n is the (effective) number of observations.

ConfidenceLevelSkewApproxIfNormal

[MnConfidenceLevelSkewApproxIfNormal]

Returns the (large sample) confidence point for the (sample) skew of a series of observations, given the null hypothesis that the observations are coming from a normal distribution. See e.g. Press et al. (2007). Is calculated from the quantile point of a normal distribution with standard deviation $\sqrt{6/n}$ where n is the (effective) number of observations.

ConfidenceNorm

[MnConfidenceNorm]

Returns the size of a confidence interval for the mean of a sample, using a normal distribution, i.e. returns:

$$-N^{-1}\left(\frac{\alpha}{2}\right)\frac{\sigma}{\sqrt{n}}$$

where N^{-1} is the inverse cumulative distribution function for the normal distribution, σ is the standard deviation and n is the sample size.

ConfidenceT

[MnConfidenceT]

Returns the size of a confidence interval for the mean of a sample, using a Student's t distribution, i.e. returns:

$$-F^{-1}\left(\frac{\alpha}{2},\nu\right)\frac{\sigma}{\sqrt{n}}$$

where $\nu = n - 1$, F^{-1} is the inverse cumulative distribution function for the Student's t distribution with ν degrees of freedom, σ is the standard deviation and n is the sample size.

Provided for compatibility with the CONFIDENCE.T function in Microsoft Excel and some other spreadsheet packages. Please note that the Nematrian function accepts non-integral values of ν (these can arise if observations are given different weights).

ConstrainedQuadraticOptimiser

[MnConstrainedQuadraticOptimiser]

Returns a vector $x = (x_1, \dots x_n)^T$ that minimises a given penalty function $L(x) = B + Cx + x^T Dx$ subject to lower bound constraints of the form $x \geq Q$ and m further (linear) constraints of the form $Ax \leq (or = or\ \geq)P$.

Constraints are coded -1 for $\leq$, 0 for $=$ and +1 for $\geq$. For example, if $n = 3$, $m = 2$ and the additional linear constraints are $x_1 + x_2 + x_3 = 1$ and $2x_1 + 3x_2 + 4x_3 \leq 5$ then:

$$ConstraintMatrix = A = (1,1,1,2,3,4)$$
$$ConstraintLimits = P = (1,5)$$
$$ConstraintTypes = (0,-1)$$

ConstrainedQuadraticPortfolioOptimiser

[MnConstrainedQuadraticPortfolioOptimiser]

Returns a vector $x = (x_1, \dots x_n)^T$ (and three further values indicating the total asset weight, the return and the risk of the portfolio in that order) that maximises the following investor utility function subject to lower bound constraints of the form $x \geq Q$ and m further (linear) constraints of the form $x \geq Q$ and $Ax \leq (or = or \; \geq)P$.

$$U(x) = r.x - \lambda(x-b)^T V(x-b)$$

Here x are the portfolio weights (so typically we impose at least the following constraint $\sum_i x_i = 1$), b is the benchmark (or 'minimum risk' portfolio), r is a vector of assumed returns on each asset and V is the covariance matrix ($= s^T Cs$, where s is the vector of risks on each asset class, here assumed to be characterised by their volatilities, as this approach is merely a mean-variance one, and C their correlation matrix).

Constraints are coded -1 for $\leq$, 0 for $=$ and +1 for $\geq$. For example, if $n = 3$, $m = 2$ and the additional linear constraints are $x_1 + x_2 + x_3 = 1$ and $2x_1 + 3x_2 + 4x_3 \leq 5$ then:

$$ConstraintMatrix = A = (1,1,1,2,3,4)$$
$$ConstraintLimits = P = (1,5)$$
$$ConstraintTypes = (0,-1)$$

Convert2dIndexArrayToLogRelativeReturns

[MnConvert2dIndexArrayToLogRelativeReturns]

Returns an array of logged relative returns given as input an array of index values, for several different series simultaneously, i.e. if input array is $x_{i,j}$ for $i = 1, \dots, n$, $j = 1, \dots, m$ (actually passed as a unidimensional array of size $n \times m$ whose $m(i-1) + j$'th element is $x_{i,j}$) and benchmark index vector is b_j for $j = 1, \dots, m$ then returns an output array $y_{i,j}$ for $i = 1, \dots, n$, $j = 1, \dots, m-1$ (actually passed as a unidimensional array of size $n \times (m-1)$ whose $(m-1)(i-1) + j$'th element is $y_{i,j}$) calculated as:

$$y_{i,j} = \log\left(\frac{x_{i,j+1}}{x_{i,j}} \cdot \frac{b_j}{b_{j+1}}\right)$$

Convert2dIndexArrayToLogReturns

[MnConvert2dIndexArrayToLogReturns]

Returns an array of logged returns given as input an array of index values, for several different series simultaneously, i.e. if input array is $x_{i,j}$ for $i = 1, \dots, n$, $j = 1, \dots, m$ (actually passed as a unidimensional array of size $n \times m$ whose $m(i-1) + j$'th element is $x_{i,j}$) then returns an output array $y_{i,j}$ for $i = 1, \dots, n$, $j = 1, \dots, m-1$ (actually passed as a unidimensional array of size $n \times (m-1)$ whose $(m-1)(i-1) + j$'th element is $y_{i,j}$) calculated as:

$$y_{i,j} = \log\left(\frac{x_{i,j+1}}{x_{i,j}}\right)$$

Convert2dIndexArrayToRelativeReturns

[MnConvert2dIndexArrayToRelativeReturns]

Returns an array of relative returns given as input an array of index values, for several different series simultaneously, i.e. if input array is $x_{i,j}$ for $i = 1, \dots, n$, $j = 1, \dots, m$ (actually passed as a unidimensional array of size $n \times m$ whose $m(i-1)+j$'th element is $x_{i,j}$) and benchmark index vector is b_j for $j = 1, \dots, m$ then returns an output array $y_{i,j}$ for $i = 1, \dots, n$, $j = 1, \dots, m-1$ (actually passed as a unidimensional array of size $n \times (m-1)$ whose $(m-1)(i-1)+j$'th element is $y_{i,j}$) calculated as:

$$y_{i,j} = \frac{x_{i,j+1}}{x_{i,j}} \cdot \frac{b_j}{b_{j+1}} - 1$$

Convert2dIndexArrayToReturns

[MnConvert2dIndexArrayToReturns]

Returns an array of returns given as input an array of index values, for several different series simultaneously, i.e. if input array is $x_{i,j}$ for $i = 1, \dots, n$, $j = 1, \dots, m$ (actually passed as a unidimensional array of size $n \times m$ whose $m(i-1)+j$'th element is $x_{i,j}$) then returns an output array $y_{i,j}$ for $i = 1, \dots, n$, $j = 1, \dots, m-1$ (actually passed as a unidimensional array of size $n \times (m-1)$ whose $(m-1)(i-1)+j$'th element is $y_{i,j}$) calculated as:

$$y_{i,j} = \frac{x_{i,j+1}}{x_{i,j}} - 1$$

ConvertDateArrayToDoubleArray

[MnConvertDateArrayToDoubleArray]

Returns an array of Double variables corresponding to the input Date variables (using the 'Date.FromOADate' facility within Visual Basic)

ConvertDateToDouble

[MnConvertDateToDouble]

Returns a Double variable corresponding to the input Date variable (using the 'Date.FromOADate' facility within Visual Basic)

ConvertDoubleArrayToDateArray

[MnConvertDoubleArrayToDateArray]

Returns an array of Date variable corresponding to the input Double variables (using the ‘Date.ToOADate’ facility within Visual Basic)

ConvertDoubleToDate

[MnConvertDoubleToDate]

Returns a Date variable corresponding to the input Double variable (using the ‘Date.ToOADate’ facility within Visual Basic)

ConvertIntegerArrayToDoubleArray

[MnConvertIntegerArrayToDoubleArray]

Returns an array of Double variables corresponding to the input Integer variables (useful for converting values to the right sort of input in e.g. MnEvaluate)

ConvertIntegerToDouble

[MnConvertIntegerToDouble]

Returns a Double variable corresponding to the input Integer variable (useful for converting values to the right sort of input in e.g. MnEvaluate)

ConvertUnit

[MnConvertUnit]

Returns the value reexpressed from one physical unit of measurement into another physical unit of measurement. If you want to convert between arbitrary compound units of measurement (e.g. miles per hour), then see MnConvertUnitGeneral.

Please note that this function converts temperature scale units according to their absolute value rather than their size (e.g. it will convert 5 degrees Celsius into 32 + 9 = 41 degrees Fahrenheit).

ConvertUnitCompatibles

[MnConvertUnitCompatibles]

Returns an array containing all recognised physical units of measurement that have the same underlying physical dimensions as *UnitName*.

ConvertUnitFromSI

[MnConvertUnitFromSI]

Converts a unit value in the corresponding SI physical unit to a unit value in the selected physical unit (which is not necessarily a SI unit).

Please note that this function converts temperature scale units according to their absolute value rather than their size (e.g. it will convert 273.15 Kelvin into 0 degrees Celsius).

ConvertUnitGeneral

[MnConvertUnitGeneral]

Returns a general unit conversion, re-expressing a value in the *FromUnits* into a value in the *ToUnits* allowing for arbitrary (integer) powers of each unit, e.g. if re-expressing a value expressed in ms^{-2} we would input into *FromUnits* an array {metre, second} and into *FromUnitPowers* an array {1, -2}.

Please note that this function converts temperature scale units according to their size rather than their absolute value (e.g. it will convert 5 degrees Celsius into 9 degrees Fahrenheit).

ConvertUnitSIEquivalent

[MnConvertUnitSIEquivalent]

Returns the SI equivalent of a given physical unit.

ConvertUnitToSI

[MnConvertUnitToSI]

Converts a unit value in the selected physical unit (which is not necessarily a SI unit) to a unit value in the equivalent SI unit.

Please note that this function converts temperature scale units according to their absolute value rather than their size (e.g. it will convert 0 degrees Celsius into 273.15 Kelvin).

ConvertUnitToSIExplanation

[MnConvertUnitToSIExplanation]

Returns an explanation of the derivation of a unit conversion into the equivalent SI units (and whether the conversion is exact to the accuracy of the Nematrian servers).

ConvertUnitToSIFactor

[MnConvertUnitToSIFactor]

Returns the factor used to convert observations in one measurement basis to another measurement basis (as long as the factor scaling is purely multiplicative).

Please note that this function converts temperature scale units according to their size rather than their absolute value (e.g. it will convert 9 degrees Fahrenheit into 5 Kelvin).

ConvertUnitType

[MnConvertUnitType]

Returns the type of unit that a measurement unit is (e.g. if it is a SI unit)

CornishFisher4

[MnCornishFisher4]

Returns an array of n elements containing the quantiles predicted by the 4th moment Cornish-Fisher asymptotic expansion for a given input array of probability levels (with n elements), for a given mean, standard deviation, skew and (excess) kurtosis.

The Cornish-Fisher asymptotic expansion is a methodology for predicting the shape of a (univariate) distributional form merely from the moments of the distribution. The 4th moment variant uses the mean, standard deviation, skew and (excess) kurtosis of the distribution.

For standardised returns (zero mean, unit standard deviation), the 4th moment variant involves estimating the shape of the quantile-quantile plot of the actual versus (standard) Normal distribution using the following cubic equation, where γ_1 is the skew and γ_2 is the (excess) kurtosis of the distribution:

$$y(x) = x + \frac{\gamma_1(x^2-1)}{6} + \frac{3\gamma_2(x^3-3x) - 2\gamma_1^2(2x^3-5x)}{72}$$

In the more generalised case where the mean, m, is not necessarily zero and the standard deviation, σ is not necessarily unity the 4th moment variant involves estimating the shape of the quantile-quantile plot as follows:

$$y(x) = m + \sigma\left(x + \frac{\gamma_1(x^2-1)}{6} + \frac{3\gamma_2(x^3-3x) - 2\gamma_1^2(2x^3-5x)}{72}\right)$$

Correlation

[MnCorrelation]

Returns the (Pearson) correlation coefficient $c_{1,2}$ between two series, $x_{1,t}$ and $x_{2,t}$.

The correlation coefficient is calculated as:

$$c_{1,2} = \frac{\frac{1}{n-1}\sum_{t=1}^{n}\left(x_{1,t}-\bar{x}_1\right)\left(x_{2,t}-\bar{x}_2\right)}{\sqrt{\frac{1}{n-1}\sum_{t=1}^{n}\left(x_{1,t}-\bar{x}_1\right)^2 \cdot \frac{1}{n-1}\sum_{t=1}^{n}\left(x_{2,t}-\bar{x}_2\right)^2}} = \frac{V_{1,2}}{\sqrt{s_1^2 s_2^2}}$$

where $\bar{x}_i$ and s_i are the mean and (sample) standard deviations of the i-th series etc. and $V_{1,2}$ is the (sample) covariance between the two series.

Correlations

[MnCorrelations]

Returns an array of m^2 elements corresponding to the elements of the matrix, C, of (Pearson) correlation coefficients for m series that are provided to the function as an array of mn elements, where the first n elements correspond to the terms in the first series, the next n elements to the corresponding terms in the second series etc.

Correlation coefficients are calculated as:

$$c_{i,j} = \frac{\frac{1}{n-1}\sum_{t=1}^{n}(x_{i,t}-\bar{x}_i)(x_{j,t}-\bar{x}_j)}{\sqrt{\frac{1}{n-1}\sum_{t=1}^{n}(x_{i,t}-\bar{x}_i)^2 \cdot \frac{1}{n-1}\sum_{t=1}^{n}(x_{j,t}-\bar{x}_j)^2}} = \frac{V_{i,j}}{\sqrt{s_i^2 s_j^2}}$$

where $\bar{x}_i$ is the mean of the i-th series etc.

CorrespondingSpouseMortalityTableName

[MnCorrespondingSpouseMortalityTableName]

Returns the mortality table name applicable to that of an individual of the opposite sex of someone to whom the input mortality name is applicable, e.g. the mortality table name corresponding to "PCMA00" is "PCFA00".

Currently the Nematrian website supports a range of UK mortality tables using nomenclature derived from terminology adopted by the UK's Continuous Mortality Investigation Bureau.

Cos

[MnCos]

Returns the cosine of a (real) number (the input value being expressed in radians)

Cosh

[MnCosh]

Returns the hyperbolic cosine of a (real) number, i.e.:

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

Cot

[MnCot]

Returns the cotangent of an angle. This is the reciprocal of the tangent function:

$$\cot x = \frac{1}{\tan x}$$

Coth

[MnCoth]

Returns the hyperbolic cotangent of an angle. This is the reciprocal of the hyperbolic tangent function:

$$\coth x = \frac{1}{\tanh x} = \frac{e^x + e^{-x}}{e^x - e^{-x}}$$

CountArray
[MnCountArray]

Returns the number of entries in *InputArray*

CountBooleanArray
[MnCountBooleanArray]

Returns the number of entries in *InputArray*

CountDateArray
[MnCountDateArray]

Returns the number of entries in *InputArray*

CountIntegerArray
[MnCountIntegerArray]

Returns the number of entries in *InputArray*

CountSetOfMembersOfArray
[MnCountSetOfMembersOfArray]

Returns the number of distinct members of *InputArray*, e.g. if *InputArray* is {3.0, 5.0, 5.0, 4.0} then returns 3 as the distinct members are {3.0, 5.0, 4.0}

CountSetOfMembersOfDateArray
[MnCountSetOfMembersOfDateArray]

Returns the number of distinct members of *InputArray*, e.g. if *InputArray* is {3 Jan 1980, 5 Jan 1980, 5 Jan 1980, 4 Jan 1980} then returns 3 as the distinct members are {3 Jan 1980, 5 Jan 1980, 4 Jan 1980}

CountSetOfMembersOfIntegerArray
[MnCountSetOfMembersOfIntegerArray]

Returns the number of distinct members of *InputArray*, e.g. if *InputArray* is {3, 5, 5, 4} then returns 3 as the distinct members are {3, 5, 4}

CountSetOfMembersOfStringArray

[MnCountSetOfMembersOfStringArray]

Returns the number of distinct members of *InputArray*, e.g. if *InputArray* is {“3”, “5”, “5”, “4”} then returns 3 as the distinct members are {“3”, “5”, “4”}

CountsOfArray

[MnCountsOfArray]

Returns the number of occurrences of distinct values in numeric array (i.e. an array of ‘double precision’ values). The entries in the output array correspond to consecutively ordered distinct values, i.e. are the elements returned by MnSortedSetOfMembersOfArray.

CountsOfDateArray

[MnCountsOfDateArray]

Returns the number of occurrences of distinct values in an array. The entries in the output array correspond to consecutively ordered distinct values, i.e. are the elements returned by MnSortedSetOfMembersOfDateArray.

CountsOfIntegerArray

[MnCountsOfIntegerArray]

Returns the number of occurrences of distinct values in an array. The entries in the output array correspond to consecutively ordered distinct values, i.e. are the elements returned by MnSortedSetOfMembersOfIntegerArray.

CountsOfStringArray

[MnCountsOfStringArray]

Returns the number of occurrences of distinct values in an array. The entries in the output array correspond to consecutively ordered distinct values, i.e. are the elements returned by MnSortedSetOfMembersOfStringArray.

CountStringArray

[MnCountStringArray]

Returns the number of entries in *InputArray*

Covariance

[MnCovariance]

Returns the (sample) covariance $V_{1,2}$ between two series, $x_{1,t}$ and $x_{2,t}$.

The (sample) covariance is calculated as:

$$V_{1,2} = \frac{1}{n-1}\sum_{t=1}^{n}(x_{1,t} - \bar{x}_1)(x_{2,t} - \bar{x}_2)$$

where $\bar{x}_i$ is the mean of the i-th series etc.

Please note that the Nematrian website's equivalent to Microsoft Excel's COVAR is the population covariance, i.e. MnPopulationCovariance rather than this function.

Covariances

[MnCovariances]

Returns an array of m^2 elements corresponding to the elements of the matrix, V, of the (sample) covariances for m series that are provided to the function as an array of mn elements, where the first n elements correspond to the terms in the first series, the next n elements to the corresponding terms in the second series etc.

(Sample) covariances are calculated as:

$$V_{i,j} = \frac{1}{n-1}\sum_{t=1}^{n}(x_{i,t} - \bar{x}_i)(x_{j,t} - \bar{x}_j)$$

where $\bar{x}_i$ is the mean of the i-th series etc.

CPower

[MnCPower]

Returns ${z_1}^{z_2}$, i.e. (the principal value of) one complex number, z_1 raised to the power of another, z_2, The Nematrian website evaluates this as follows:

$${z_1}^{z_2} = \exp\left(z_2 \log(z_1)\right)$$

For details of how to pass complex numbers to and from the Nematrian website and on principal values of complex-valued functions please see Complex Numbers Introduction.

CReal

[MnCReal]

Returns $Re(z)$, the real part of a complex number, z. If z has the form $x + iy$ where x and y are real then:

$$Re(z) = x$$

For details of how to pass complex numbers to and from the Nematrian website and on principal values of complex-valued functions please see Complex Numbers Introduction.

CreateArrayWithRepeatedElement

[MnCreateArrayWithRepeatedElement]

Creates an array that contains the same element (*RepeatedElement*) repeated *n* times

CreateArrayWithRepeatingElements

[MnCreateArrayWithRepeatingElements]

Creates an array that contains a sequence of elements (*RepeatingElements*) repeated different numbers of times (these numbers are input in the *NumbersOfRepeats* array)

Csc

[MnCsc]

Returns the cosecant of an angle. This is the reciprocal of the sine function:

$$\csc x = \frac{1}{\sin x}$$

Csch

[MnCsch]

Returns the hyperbolic cosecant of an angle. This is the reciprocal of the hyperbolic sine function:

$$\operatorname{csch} x = \frac{1}{\sinh x} = \frac{2}{e^x - e^{-x}}$$

CSec

[MnCSec]

Returns $\sec(z)$, the secant of a complex number, z. The secant can be calculated as follows. Given this simple relationship, it is used rather infrequently (e.g. in astronomy), so no equivalent inverse function is provided by the Nematrian website:

$$\sec(z) = \frac{1}{\cos z}$$

For details of how to pass complex numbers to and from the Nematrian website please see Complex Numbers Introduction.

CSin

[MnCSin]

Returns $\sin(z)$, the sine of a complex number, z. If z has the form $x + iy$ where x and y are real then:

$$\sin(z) = (\sin x \cosh y) + i(\cos x \sinh y)$$

For details of how to pass complex numbers to and from the Nematrian website please see Complex Numbers Introduction.

CSinh

[MnCSinh]

Returns $\sinh(z)$, the hyperbolic sine of a complex number, z. If z has the form $x + iy$ where x and y are real then:

$$\sinh(z) \equiv \frac{e^z - e^{-z}}{2} = (\sinh x \cos y) + i(\cosh x \sin y)$$

For details of how to pass complex numbers to and from the Nematrian website please see Complex Numbers Introduction.

CSqrt

[MnCSqrt]

Returns $\sqrt{z}$, the principal value of the square root of a complex number.

For details of how to pass complex numbers to and from the Nematrian website and on principal values of complex-valued functions please see Complex Numbers Introduction.

CSubtract

[MnCSubtract]

Returns $z_1 - z_2$, the difference between two complex numbers. If z_1 and z_2 have the form $x_1 + iy_1$ and $x_2 + iy_2$ respectively where $x_1,\ x_2,\ y_1,\ y_2,$ are real then:

$$z_1 - z_2 = (x_1 - x_2) + i(y_1 - y_2)$$

For details of how to pass complex numbers to and from the Nematrian website please see Complex Numbers Introduction.

CTan

[MnCTan]

Returns $\tan(z)$, the tangent of a complex number, z (if $\cos z \neq 0$).

$$\tan(z) = \frac{\sin z}{\cos z}$$

For details of how to pass complex numbers to and from the Nematrian website please see Complex Numbers Introduction.

CTanh

[MnCTanh]

Returns $\tanh(z)$, the hyperbolic tangent of a complex number, z (if $\cosh(z) \neq 0$).

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

For details of how to pass complex numbers to and from the Nematrian website please see Complex Numbers Introduction.

CubeRoot

[MnCubeRoot]

Returns the cube root of a (real) number

CumulativeNormal

[MnCumulativeNormal]

MnCumulativeNormal

Returns $N(z)$, the cumulative unit Normal distribution function, i.e. for a $N(0,1)$ distribution, where:

$$N(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-t^2/2} dt = \frac{1}{2} + \frac{1}{2} \mathrm{erf}\left(\frac{z}{\sqrt{2}}\right)$$

The equivalent function in Microsoft Excel is NORMSDIST.

In practice, the website uses the following rational approximation to the above integral (as apparently do some versions of Microsoft Excel), source e.g. Abramowitz and Stegun (1970):

$$y = \frac{1}{1 + a|z|}$$
$$r = f(z)(c_1 y + c_2 y^2 + c_3 y^3 + c_4 y^4 + c_5 y^5)$$
$$N(z) \cong \begin{cases} r & if\ z \leq 0 \\ 1 - r & otherwise \end{cases}$$

where:

$a = 0.2316419$, $c_1 = 0.31938153$, $c_2 = -0.356563782$,
$c_3 = 1.781477937$, $c_4 = -1.821255978$, $c_5 = 1.330274429$

and $f(z)$ is the unit Normal density function, i.e.

$$f(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2}$$

Is a special case of MnProbDistCdf.

CumulativeSeries

[MnCumulativeSeries]

Returns the cumulative values of a series, i.e. y_i where $y_1 = x_1$ and $y_i = y_{i-1} + x_i$ for $i > 1$.

DAAnovaOneAnalysis

[MnDAAnovaOneAnalysis]

Returns the results of a one factor Analysis of Variance, given an input array, *InputStringArray* (of type string), which needs to be of length *n1* x *n2*, where *n1* is the number of groups of data and *n2* is the (maximum) number of observations available for each group. For compatibility with the Data Analysis Toolpack available via Microsoft Excel the input values are strings and non-numerical values (e.g. blanks) are ignored (so the number of observations available for each set may differ).

The output is as a numeric array with 3 rows and 6 columns as follows (akin to that supplied by the Data Analysis Toolpack available via Microsoft Excel):

Rows: Applicable ANOVA component, here:

- Row 1: between groups
- Row 2: within groups
- Row 3: total

Columns: Various relevant outputs, here:

- Col 1: *SS* = Sum of squared deviations for applicable ANOVA component
- Col 2: *df* = Degrees of freedom for applicable ANOVA component
- Col 3: *MS* = *SS*/*df* (for Rows 1 and 2 only)
- Col 4: *F* statistic, i.e. MS for applicable ANOVA component divided by *MS* for within group (Row 1 only)
- Col 5: *P*-value from *F* distribution applicable to the *F* statistic (Row 1 only)
- Col 6: Equivalent critical value of *F* distribution given degrees of freedom and confidence level *Alpha*

DAAnovaOneSummary

[MnDAAnovaOneSummary]

Returns a summary of the input data applicable to a one factor Analysis of Variance, given an input array, *InputStringArray* (of type string), which needs to be of length *n1* x *n2*, where *n1* is the number of groups of data and *n2* is the (maximum) number of observations available for each group. For compatibility with the Data Analysis Toolpack available via Microsoft Excel the input values are strings and non-numerical values (e.g. blanks) are ignored (so the number of observations available for each set may differ).

The output is a numeric array with *n1* rows and 4 columns as follows (akin to that supplied by the Data Analysis Toolpack available via Microsoft Excel).

Rows 1 to *n1* contain summary data for consecutive groups

Col 1: Count
Col 2: Sum
Col 3: Average
Col 4: (Sample) variance

See also MnDAAnovaOneAnalysis.

DAAnovaTwoWithAnalysis

[MnDAAnovaTwoWithAnalysis]

Returns the results of a two factor Analysis of Variance (with replication), given an input array, *InputArray*, which needs to be of length *n1* x *n2*, where *n1* is the number of groups of data and *n2* is the number of observations for each group, with the groups further subdivided into sub-groups, the (string) values of which are contained in the array *SubGroups*.

The output is as a numeric array with 3 rows and 6 columns, as follows (akin to that supplied by the Data Analysis Toolpack available via Microsoft Excel):

Rows: Applicable ANOVA component, here

- Row 1: sample
- Row 2: columns
- Row 3: interaction
- Row 4: within (sub)groups
- Row 5: total

Columns: Various relevant outputs:

- Col 1: *SS* = Sum of squared deviations for applicable ANOVA component
- Col 2: *df* = Degrees of freedom for applicable ANOVA component
- Col 3: *MS* = *SS*/*df* (for Rows 1 – 4 only)
- Col 4: *F* statistic, i.e. *MS* for applicable ANOVA component divided by *MS* for within group (Rows 1 – 3 only)
- Col 5: *P*-value from *F* distribution applicable to the *F* statistic (Rows 1 – 3 only)
- Col 6: Equivalent critical value of *F* distribution given degrees of freedom and confidence level *Alpha*

DAAnovaTwoWithoutAnalysis

[MnDAAnovaTwoWithoutAnalysis]

Returns the results of a two factor Analysis of Variance (without replication), given an input array, *InputArray*, which needs to be of length *n1* x *n2*, where *n1* is the number of groups of data and *n2* is the number of observations for each group.

The output is as a numeric array with 3 rows and 6 columns, as follows (akin to that supplied by the Data Analysis Toolpack available via Microsoft Excel):

Rows: Applicable ANOVA component, here

- Row 1: rows
- Row 2: columns
- Row 3: error
- Row 4: total

Columns: Various relevant outputs:

- Col 1: *SS* = Sum of squared deviations for applicable ANOVA component
- Col 2: *df* = Degrees of freedom for applicable ANOVA component
- Col 3: *MS* = *SS*/*df* (for Rows 1 – 3 only)
- Col 4: *F* statistic, i.e. *MS* for applicable ANOVA component divided by *MS* for error (Rows 1 and 2 only)
- Col 5: *P*-value from *F* distribution applicable to the *F* statistic (Rows 1 and 2 only)
- Col 6: Equivalent critical value of *F* distribution given degrees of freedom and confidence level *Alpha*

DAAnovaTwoWithoutSummary
[MnDAAnovaTwoWithoutSummary]

Returns a summary of the input data applicable to a two factor Analysis of Variance (without replication), given an input array, *InputArray*, which needs to be of length *n1* x *n2*, where *n1* is the number of groups of data and *n2* is the number of observations for each group.

The output is a numeric array with (*n1* + *n2*) rows and 4 columns as follows (akin to that supplied by the Data Analysis Toolpack available via Microsoft Excel).

Rows 1 to *n1* contain summary data for consecutive rows
Rows *n1*+1 to *n1*+*n2* contain summary data for consecutive columns

Col 1: Count
Col 2: Sum
Col 3: Average
Col 4: (Sample) variance

DAAnovaTwoWithSummary
[MnDAAnovaTwoWithSummary]

Returns a summary of the input data applicable to a two factor Analysis of Variance (with replication), given an input array, *InputArray*, which needs to be of length *n1* x *n2*, where *n1* is the number of groups of data and *n2* is the number of observations for each group, with the groups further subdivided into sub-groups, the (string) values of which are contained in the array *SubGroups*.

The output is a numeric array with *(m+1) x 4* rows and *n2* + 1 columns as follows (akin to that supplied by the Data Analysis Toolpack available via Microsoft Excel), where *m* is the number of distinct subgroups in the array *SubGroups*:

Each group of 4 rows contains summary data for a sub-group (the last 4 rows containing summary data for all subgroups combined), in order that the subgroup first appears in array *SubGroups*. Within each group of 4 rows the data is as follows:

Row 1: Count
Row 2: Sum
Row 3: Average
Row 4: Sample variance

Each column relates to a separate column within the input array (the final column relating to all columns combined.

DACorrelations

[MnDACorrelations]

Returns a correlation matrix (only lower triangular components) of series that are contained in *InputArray*, so the (lower) *i,j*'th component of the matrix is the correlation between the elements of *InputArray*(*i*,*t*) and *InputArray*(*j*,*t*).

Is the same as the result of applying MnMatLT to MnCorrelations.

DACovariances

[MnDACovariances]

Returns a covariance matrix (only lower triangular components) of series that are contained in *InputArray*, so the (lower) *i,j*'th component of the matrix is the (population) covariance between the elements of *InputArray*(*i*,*t*) and *InputArray*(*j*,*t*).

Is the same as the result of applying MnMatLT to MnCovariances.

DAExponentialSmoothing

[MnDAExponentialSmoothing]

Returns an array providing the result of applying an exponential smoothing algorithm to the *InputArray* (algorithm used is compatible with the one implicit in the Excel Data Analysis Toolpack as at 2012).

If the input array, $x(t)$, is of length n then the output consists of a two dimensional array $ans(t,q)$ of size $(n-1)\times 2$ where $q=0$ corresponds to the exponentially smoothed values, $y(t)$ and $q=1$ corresponds to the "standard error" of these values, $e(t)$, these being calculated as follows, where $s = SmoothingFactor$ and $m = StdErrorCalcWindow$:

$$y(t) = \begin{cases} x(1) \; if \; t = 1 \\ s.x(t-1) + (1-s)y(t-1) \; otherwise \end{cases}$$

$$e(t) = \begin{cases} 0 \; if \; t < m+1 \; (these \; values \; should \; be \; ignored) \\ \sqrt{\frac{\sum_{i=t-m+1}^{t} \left(y(t) - x(t-1)\right)^2}{m}} \end{cases}$$

DAFTest

[MnDAFTest]

Returns the results of an *F*-test applied to two series designed to identify if the two have different variances, given the null hypothesis that they are drawn from normal distributions with the same variance.

The output has 7 rows and 2 columns (for Rows 1 to 4, ignore second column entry for Rows 5 to 7):

- Row 1: means
- Row 2: (sample) variances
- Row 3: numbers of observations in each series
- Row 4: numbers of degrees of freedom for each series
- Row 5: *F* statistic
- Row 6: Probability of statistic being less than *F*-statistic if null hypothesis applies, for one-tail test
- Row 7: Critical value of statistic for given level of *alpha*

DAMovingAverage

[MnDAMovingAverage]

Returns an array providing the result of applying a moving average algorithm to the *InputArray* (algorithm used is compatible with the one implicit in the Excel Data Analysis Toolpack as at 2012).

DateAdd

[MnDateAdd]

Returns the result of adding a set number of time intervals to a date/time (or if *NumberOfIntervals* is negative then subtract)

DateAndTimeSerial

[MnDateAndTimeSerial]

Returns a date (with a no zero time) given separate inputs for year, month, day, hour, minute and (integral) second

DateDiff

[MnDateDiff]

Returns the number of time intervals between two dates/times

DateDiffFull
[MnDateDiffFull]

Returns the number of time intervals between two dates/times, potentially including weekly data

DatePart
[MnDatePart]

Returns the relevant time part of a date/time

DatePartFull
[MnDatePartFull]

Returns the relevant time part of a date/time, potentially including weekly data

DateSerial
[MnDateSerial]

Returns a date given separate inputs for year, month and day

DateValue
[MnDateValue]

Returns a date given a single string input

DATTestEqual
[MnDATTestEqual]

Returns the results of a Student's *t* test applied to two separate series to test if their means are different, under the null hypothesis that the data is normally distributed and the series have the same (unknown) variance.

The output has 10 rows and 2 columns (ignore second column entry for Rows 4 to 10):

- Row 1: Mean (of series)
- Row 2: Variance (of series)
- Row 3: Number of entries in series
- Row 4: Hypothesized Mean Difference
- Row 5: degrees of freedom in Student's *t* test
- Row 6: *t* statistic

- Row 7: *P*-value of this *t* statistic (one tail)
- Row 8: *t* critical level (one tail), for given confidence level
- Row 9: *P*-value of this t statistic (two tail)
- Row 10: *t* critical level (two tail), for given confidence level

DATTestPaired

[MnDATTestPaired]

Returns the results of a Student's *t* test applied to paired data designed to identify if the difference (element by element) of two series of equal lengths is normally distributed, given the null hypothesis that the differences are drawn from a normal distribution.

The output has 11 rows and 2 columns (ignore second column entry for Rows 5 to 11):

- Row 1: Mean (of series)
- Row 2: Variance (of series)
- Row 3: Number of entries in series
- Row 4: Pearson Correlation Coefficient between the two series
- Row 5: Hypothesized Mean Difference
- Row 6: degrees of freedom in Student's *t* test
- Row 7: *t* statistic
- Row 8: *P*-value of this *t* statistic (one tail)
- Row 9: *t* critical level (one tail), for given confidence level
- Row 10: *P*-value of this t statistic (two tail)
- Row 11: *t* critical level (two tail), for given confidence level

DATTestUnequal

[MnDATTestUnequal]

Returns the results of an approximate Student's *t* test applied to two separate series to test if their means are different, under the null hypothesis that the data is normally distributed and that the two series do not necessarily have the same (unknown) variance.

The output has 10 rows and 2 columns (ignore second column entry for Rows 4 to 10):

- Row 1: Mean (of series)
- Row 2: Variance (of series)
- Row 3: Number of entries in series
- Row 4: Hypothesized Mean Difference
- Row 5: degrees of freedom in Student's *t* test
- Row 6: *t* statistic
- Row 7: *P*-value of this *t* statistic (one tail)
- Row 8: *t* critical level (one tail), for given confidence level
- Row 9: *P*-value of this *t* statistic (two tail)
- Row 10: *t* critical level (two tail), for given confidence level

Day

[MnDay]

Returns the day of a given date/time

DAZTest

[MnDAZTest]

Returns the results of a z test t test applied to two separate series to test if their means are different, under the null hypothesis that the data is normally distributed and that the two series have known variances.

The output has 9 rows and 2 columns (ignore second column entry for Rows 4 to 9):

- Row 1: Mean (of series)
- Row 2: Variance (of series)
- Row 3: Number of entries in series
- Row 4: Hypothesized Mean Difference
- Row 5: *z* statistic
- Row 6: *P*-value of this *z* statistic (one tail)
- Row 7: *t* critical level (one tail), for given confidence level
- Row 8: *P*-value of this *z* statistic (two tail)
- Row 9: *t* critical level (two tail), for given confidence level

Dec2BaseNString

[MnDec2BaseNString]

Converts an (unsigned) integer to a string representing the number with a given radix between 2 and 36, consecutive digits being represented by the characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y up to (for radix = 36) Z. Similar to Excel's BASE function.

Dec2Bin

[MnDec2Bin]

Converts an (unsigned) integer to a string representing the number in binary form (i.e. base 2 with digits being 0 and 1). Similar to Excel's DEC2BIN function but accepting higher input values and rejecting negative input values.

Dec2Hex

[MnDec2Hex]

Converts an (unsigned) integer to a string representing the number in hexadecimal form (i.e. base 16 with digits being 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F). Similar to Excel's DEC2HEX function but accepting higher input values and rejecting negative input values.

Dec2Oct

[MnDec2Oct]

Converts an (unsigned) integer to a string representing the number in octal form (i.e. base 8 with digits being 0, 1, 2, 3, 4, 5, 6 and 7). Similar to Excel's DEC2OCT function but accepting higher input values and rejecting negative input values.

DecimalNumerals

[MnDecimalNumerals]

Returns a string array containing a list of the decimal numerals recognised by the site in ascending order

DegreeToRadian

[MnDegreeToRadian]

Converts a value in degrees into a value in radians

DemoTSMC

[MnDemoTSMC]

Returns a vector using tri-segmented Monte Carlo (TSMC) that contains three entries: *Portfolio Value*, *Quantile* and *Control Variate Variance Ratio*.

This function provides a simplified demonstration of the usefulness of tri-segmented Monte Carlo for valuing portfolios of assets and liabilities and for calculating associated risk measures of these portfolios

For further details of TSMC, see Efficient Monte Carlo simulation of portfolio value, value-at-risk and other portfolio metrics or speak to your contact at Nematrian. Instead of applying all simulations to a portfolio of exposures, the simulations are split into 3 subsets, (1) an "underlying", (2) an "added" and (3) an "extended" simulation set. The extended set is usually by far the largest of these three sets. Only the underlying and added sets are actually applied to the portfolio; the extended set is instead applied to only to a fast to evaluate approximation derived principally from the underlying simulation set. The added simulation set helps to correct for inaccuracies in this approximation.

The demonstration of TSMC provided by MnDemoTSMC assumes a portfolio of 40 instruments which have specified strikes, $K(i)$, and terms, $T(i)$ (see MnDemoTSMCStrikes and MnDemoTSMCTerms for details) and relate to one of two specified indices, $U(i)$ (see MnDemoTSMCUnderlyings for details). We use $S_{u,t}$ to represent the value of the u'th index at time t. Each instrument is a European-style call option so has payoff equal to $PosSizes(i) \times \max\left(S_{U(i),T(i)} - K(i), 0\right)$. Instrument terms are all 1, 2, 3, 4 or 5 years. Users can select the position sizes, i.e. *PosSizes* to apply to the portfolio.

The two indices are assumed to follow independent log-normal (Brownian) motions with (constant) annualised volatilities (cumulatively compounded) specified by the two entries of *ImpliedVolatilities*. It is assumed that interest rates are zero at all times, making it practical to value each instrument analytically using a Nematrian web function call along the line of:

PosSizes(*i*) * MnBSCall(*K*(*i*), 1, 0, 0, 0, *T*(*i*), *ImpliedVolatilities*(*U*(*i*)))

Please note:

(a) *BaseSimulationsUandA*: needs to be of size (*nUnderlying* + *nAdded*) x 10 (i.e. 10 for each simulation, these corresponding to log returns in years 1-5 for first index and to log returns in years 1-5 for the second index respectively
(b) *SimulationValuesUandA*: needs to be of size (*nUnderlying* + *nAdded*)
(c) *DistanceScaling*: this array defines the weights to give to different dimensions in the calculation of the distance between two different points in the underlying 10-dimensional space
(d) This function is designed to provide a simple demonstration of tri-segmented Monte Carlo. It therefore only considers “run-off” VaRs/quantiles and does not aim to demonstrate other elements of Nematrian’s full tri-segmented Monte Carlo engine such as 1-year / nested VaR calculations and use of importance sampling.

DemoTSMCSimGen

[MnDemoTSMCSimGen]

Returns a two-dimensional array of (*nUnderlying* + *nAdded*) x 10 terms representing unit normal random variables underlying the simulations of portfolio payoffs used by MnDemoTSMC

DemoTSMCStrike

[MnDemoTSMCStrikes]

Returns a vector of the strikes of the 40 instruments in the portfolio used by MnDemoTSMC

DemoTSMCTerms

[MnDemoTSMCTerms]

Returns a vector of the terms of the 40 instruments in the portfolio used by MnDemoTSMC

DemoTSMCUnderlyings

[MnDemoTSMCUnderlyings]

Returns a vector of the underlying indices of the 40 instruments in the portfolio used by MnDemoTSMC

DemoTSMCValueAnalytical

[MnDemoTSMCValueAnalytical]

Returns the analytic value (using MnBSCall) of the portfolio of 40 instruments used by MnDemoTSMC

DemoTSMCValueSim

[MnDemoTSMCValueSim]

Returns an array of (*nUnderlying* + *nAdded*) terms representing computed present values of payoffs for simulations of the portfolio payoffs used by MnDemoTSMC

Desmooth_AR1

[MnDesmooth_AR1]

Returns an array corresponding to the AR(1) de-smoothed (or 'de-correlated) values of a series, as described in the book Extreme Events. This involves:

(a) Postulating (for, say, a return series) that there is some underlying 'true' return series, $\tilde{r}_t$, and that the observed series, r_t, derives from it via a first order autoregressive model, $r_t = (1-\rho)\tilde{r}_t + \rho\tilde{r}_{t-1}$;

(b) Identifying the value of ρ (between 0 and 0.5) for which $\tilde{r}_t$ has zero first order correlation (which implies that the covariance between $\{r_1, \dots, r_{n-1}\}$ and $\{r_2, \dots, r_n\}$ is zero; and

(c) Using this value of ρ deriving $\tilde{r}_t$ using the following formula:

$$\tilde{r}_t = \begin{cases} r_t & \text{if } t = 1 \\ \dfrac{r_t - \rho\tilde{r}_{t-1}}{1-\rho} & \text{if } t > 1 \end{cases}$$

The value of ρ in (b) can be accessed using MnDesmooth_AR1_rho.

Desmooth_AR1_rho

[MnDesmooth_AR1_rho]

Returns the de-smoothing parameter corresponding to the AR(1) de-smoothed (or 'de-correlated) values of a series, as described in the book Extreme Events. This involves:

(a) Postulating (for, say, a return series) that there is some underlying 'true' return series, $\tilde{r}_t$, and that the observed series, r_t, derives from it via a first order autoregressive model, $r_t = (1-\rho)\tilde{r}_t + \rho\tilde{r}_{t-1}$; and

(b) Identifying the value of ρ (between 0 and 0.5) for which $\tilde{r}_t$ has zero first order correlation (which implies that the covariance between $\{r_1, \dots, r_{n-1}\}$ and $\{r_2, \dots, r_n\}$ is zero.

The de-smoothed series, $\tilde{r}_t$, can then be derived using the following formula, see MnDesmooth_AR1:

$$\tilde{r}_t = \begin{cases} r_t & \text{if } t = 1 \\ \dfrac{r_t - \rho\tilde{r}_{t-1}}{1-\rho} & \text{if } t > 1 \end{cases}$$

DevSq

[MnDevSq]

Returns the sum of the squared deviations of an array, i.e. if the elements of the array are x_i (for $i = 1, \dots, n$) then returns y where

$$y = \sum_{i=1}^{n} (x_i - \bar{x})^2$$

where

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

Provided for compatibility with the DEVSQ function in Microsoft Excel and some other spreadsheet systems.

Dec2Oct

[MnDec2Oct]

The digamma function, $\Psi(x)$ is defined as the first derivative of the log gamma function, i.e.:

$$\Psi(x) = \frac{d log \Gamma(x)}{dx} = \frac{\Gamma'(x)}{\Gamma(x)}$$

In practice, the Nematrian website uses a variant of the Lanczos approximation to compute the digamma function, see also gamma function.

Divide

[MnDivide]

Returns one number divided by another (i.e. equivalent to the / operator).

Provided for compatibility with the DIVIDE function in Google Drive.

DNSHostAddressList

[MnDNSHostAddressList]

Returns the list of IP addresses associated with a given Host Name (or IP address) within the Internet domain name system.

DNSHostName

[MnDNSHostName]

Returns the Host Name associated with a given IP address within the Internet's domain name system.

DoesArrayHaveMode

[MnDoesArrayHaveMode]

Returns true if the array has a mode, i.e. a unique value which appears in the array more than any other value, otherwise returns false

E
[MnE]

Returns the base of natural logarithms, i.e. e.

See also exponential function.

EigenvalueSpreadsForRandomMatrices
[MnEigenvalueSpreadsForRandomMatrices]

Returns the (sorted) values of $\lambda_{max}(i)$ (containing n terms) above which we would not expect eigenvalues to lie if coming from a covariance matrix derived from samples of independent, identically distributed (Gausssian) random variables, if the number of variables, n is large, given an array of actual observed eigenvalues and given the number of observations applicable to each underlying variable.

See Random Matrix Theory for further details.

EQ
[MnEQ]

Returns TRUE if two specified values are equal and FALSE otherwise. Equivalent to the == operator (in Google Drive) or the = operator (in Visual Basic etc.).

Provided for compatibility with the EQ function in Google Drive.

Erf
[MnErf]

Returns the error function

Provided for compatibility with ERF function in Microsoft Excel (although may not produce exactly the same result because of slight differences in the algorithms used). ERF(x) = MnErf(x) subject to these algorithm differences

The Microsoft Excel function includes an optional upper limit and if it is included then ERF(lower,upper) = ERF(upper) - ERF(lower).

Erfc
[MnErfc]

Returns the complementary error function, i.e. 1 minus the error function.

ErgodicProbabilities

[MnErgodicProbabilities]

Returns the ergodic (in effect, the long run) probabilities of a system characterised by a Markov chain being in its given states. The Markov chain is characterised by a matrix defining the probability of transitioning from a given state (first index) to a given state (second index).

Determination of the ergodic properties of a Markov chain is in general a complicated task as a Markov chain may not actually have ergodic properties (if e.g. it has two or more disjoint sub-chains and the system always stays within a given sub-chain because it starts out in a state within a specific sub-chain). If a Markov chain does have ergodic probabilities then these can be found by iterating the Markov chain for long enough; the long run probabilities of being in each state will then be the ergodic probabilities.

The algorithm used by the Nematrian website selects two randomly chosen sets of starting probabilities for each state, projects forward these two separate starting states up to *IterationLimit* number of times and returns an error if the two sets of states have not by then converged to the same apparent state probabilities, with convergence measured by the size of the Tolerance parameter. To allow the algorithm to produce the same answer each time it is run given the same inputs, it includes a random number seed used merely in the selection of these two initial starting probabilities.

ErrorFromValueWithBracketedError

[MnErrorFromValueWithBracketedError]

Returns the error from an input (string) expression that involves a measurement together with a bracketed term (and possibly an exponent), e.g. applying this to 1.345(34)e-12 or 1.345(34)E-12 should return 0.034 x 10^(-12), whilst 1.345(34) should return 0.034 and 1.345 should return 0 (as it assumed that if there is no term in brackets the error on the base value is zero). All space spaces in the input string are ignored (it is conventional in some contexts to write, say, 1.345674(2) as 1.345 674(2)).

Please bear in mind that for this type of function the Nematrian website only accepts decimal delimiters in the form of a full stop, i.e. '.' and only accepts integral exponents. As with all other Nematrian floating point functions, accuracy is limited to the machine accuracy of the Nematrian servers.

Evaluate

[MnEvaluate]

Evaluates an expression (*InputExpression*) using a specified computer language (*Language*) and (if applicable) given a specific entry point (*EntryPoint*) into the expression. If *Language* is blank then it uses the expression convention described here.

The number of units debited from a *SessionId* to carry out this function is driven primarily by the complexity of the expression involved and the number of calls to other Nematrian functions included in the expression.

EvaluateExpressions

[MnEvaluateExpressions]

Evaluates a set of expressions (*InputExpressions*) using a specified computer language (*Language*) and (if applicable) given a specific entry point (*EntryPoint*) into the expression. If *Language* is blank then it uses the expression convention described here. The expressions are evaluated 'all at once' in much the same way as spreadsheet formulae recalculate, i.e. the order of the expressions is irrelevant and the function will throw an error if there is a circular reference.

The number of units debited from a *SessionId* to carry out this function is driven primarily by the complexity of the expression involved and the number of calls to other Nematrian functions included in the expression.

EvaluateExpressionsMultiValues

[MnEvaluateExpressionsMultiValues]

Evaluates a set of expression (*InputExpressions*) using a specified computer language (*Language*) and (if applicable) given a specific entry point (*EntryPoint*) into the expression a multiple number of times, each time substituting a specific value for a selected variable. If *Language* is blank then it uses the expression convention described here. The expressions are evaluated 'all at once' in much the same way as spreadsheet formulae recalculate, i.e. the order of the expressions is irrelevant and the function will throw an error if there is a circular reference.

The number of units debited from a *SessionId* to carry out this function is driven primarily by the complexity of the expression involved and the number of calls to other Nematrian functions included in the expression.

EvaluateExpressionsUsing

[MnEvaluateExpressionsUsing]

Evaluates a set of expressions (*InputExpressions*) using a specified computer language (*Language*) and (if applicable) given a specific entry point (*EntryPoint*) into the expression, substituting specific values (*SubstituteValues*) for selected variables (*SubstituteNames*). The type of the variable is specified in *SubstituteTypes*. If *Language* is blank then it uses the expression convention described here. The expressions are evaluated 'all at once' in much the same way as spreadsheet formulae recalculate, i.e. the order of the expressions is irrelevant and the function will throw an error if there is a circular reference.

The number of units debited from a *SessionId* to carry out this function is driven primarily by the complexity of the expression involved and the number of calls to other Nematrian functions included in the expression.

EvaluateMultiValues
[MnEvaluateMultiValues]

Evaluates an expression (*InputExpression*) using a specified computer language (*Language*) and (if applicable) given a specific entry point (*EntryPoint*) into the expression a multiple number of times, each time substituting a specific value for a selected variable. If *Language* is blank then it uses the expression convention described here.

The number of units debited from a *SessionId* to carry out this function is driven primarily by the complexity of the expression involved and the number of calls to other Nematrian functions included in the expression.

EvaluateUsing
[MnEvaluateUsing]

Evaluates an expression (*InputText*) using a specified computer language (*Language*) and (if applicable) given a specific entry point (*EntryPoint*) into the expression, substituting a specific values for selected variables. If *Language* is blank then it uses the expression convention described here.

The number of units debited from a *SessionId* to carry out this function is driven primarily by the complexity of the expression involved and the number of calls to other Nematrian functions included in the expression.

ExampleDataGMMandKMCDataSeries
[MnExampleDataGMMandKMCDataSeries]

Returns an example data set for Gaussian mixture modelling and *k*-means clustering involving 2 series and 170 data points for each series.

Exp
[MnExp]

Returns the exponential of a real number, i.e. $y = e^x \equiv exp(x)$

ExperienceAnalysisSinglePeriodCounts
[MnExperienceAnalysisSinglePeriodCounts]

Returns the results of carrying out an ‘experience’, i.e. ‘movement’, counts analysis, given a set of (unique) *StartIDs*, a set of (unique) *EndIDs*, and classifications applied to each ID at either the start or the end of the period.

See Experience Analyses – Introduction for further details

ExponDistCdf
[MnExponDistCdf]

Returns the cumulative distribution function of the exponential distribution. Provided for compatibility with EXPONDIST and EXPON.DIST functions in Microsoft Excel 2010 and later (with cumulative parameter set to true).

Equivalent to calling MnProbDistCdf using as the distribution parameter “exponential”.

ExponDistPdf

[MnExponDistPdf]

Returns the probability density function of the exponential distribution. Provided for compatibility with EXPONDIST and EXPON.DIST functions in Microsoft Excel 2010 and later (with cumulative parameter set to false).

Equivalent to calling MnProbDistPdf using as the distribution parameter “exponential”.

Exponent

[MnExponent]

Returns the exponent of a number when expressed in scientific notation, e.g. 0.31 is written as 3.1×10^{-1} in scientific notation and thus has an exponent of -1.

ExtractFromArray

[MnExtractFromArray]

Returns an extract from a numeric array (i.e. an array of ‘double precision’ values) consisting of consecutive elements starting part way through the array, in much the same way as the MID function extracts a part of a string

ExtractFromBooleanArray

[MnExtractFromBooleanArray]

Returns an extract from a Boolean array consisting of consecutive elements starting part way through the array, in much the same way as the MID function extracts a part of a string.

ExtractFromDateArray

[MnExtractFromDateArray]

Returns an extract from a date array consisting of consecutive elements starting part way through the array, in much the same way as the MID function extracts a part of a string.

ExtractFromIntegerArray

[MnExtractFromIntegerArray]

Returns an extract from an integer array consisting of consecutive elements starting part way through the array, in much the same way as the MID function extracts a part of a string.

ExtractFromStringArray

[MnExtractFromStringArray]

Returns an extract from a string array consisting of consecutive elements starting part way through the array, in much the same way as the MID function extracts a part of a string.

Factorial

[MnFactorial]

Returns $n!$, i.e.

$$1 \times 2 \times \dots \times n = \prod_{i=1}^{n} i$$

Has a simple relationship with the gamma function, $\Gamma(x)$, see MnGamma, if $x+1$ is a positive integer

FactorialDouble

[MnFactorialDouble]

Returns $n!!$, i.e.:

(a) If n is even then $2 \times 4 \times \dots \times n$

(b) If n is odd then $1 \times 3 \times \dots \times n$

FDistCdf

[MnFDistCdf]

Returns the cumulative distribution function of the *F* distribution. Provided for compatibility with FDIST, F.DIST (with cumulative parameter set to true) and F.DIST.RT functions in Microsoft Excel 2010 and later.

Note FDIST = F.DIST.RT = 1- F.DIST(cumulative=true) and F.DIST(cumulative=true) = MnFDistCdf

Please also bear in mind that this function accepts non-integral values of *nu1* and *nu2* whereas the Excel functions are limited to integral values.

Equivalent to calling MnProbDistCdf using as the distribution parameter “f”.

FDistPdf

[MnFDistPdf]

Returns the probability density function of the *F distribution*. Provided for compatibility with FDIST and F.DIST functions in Microsoft Excel 2010 and later (with cumulative parameter set to false).

Please also bear in mind that this function accepts non-integral values of *nu1* and *nu2* whereas the Excel functions are limited to integral values.

Equivalent to calling MnProbDistPdf using as the distribution parameter “f”.

FFT

[MnFFT]

Returns the fast Fourier transform (‘FFT’) (if *CarryOutForwardTransform* is true) or the inverse FFT (if *CarryOutForwardTransform* is false) of an array of n complex numbers (i.e. input array has $2n$ term, the real part followed by the imaginary part), where n must be an integral power of two.

If the input to the FFT is h_k (each h_k is a complex number) where $k = 0, \dots, N-1 \equiv 2^m - 1$ then the output (of the forward transform) is another array of N complex numbers as follows:

$$H_n = \sum_{k=0}^{N-1} W^{nk} h_k$$

where $W \equiv e^{-2\pi i/N}$. When inverting, a constant multiplier is applied to all terms to ensure that the result of applying the transform to the output of the forward transform and then applying the transform returns the original series.

The FFT calculates the H_n using $O(N \log_2 N)$ rather than the $O(N^2)$ that might appear to be needed given the above formula. This involves a very large speed up for large data analyses, e.g. analysing large pictures or other datasets.

Some writers use $W \equiv e^{+2\pi i/N}$, e.g. Press et al. (2007), and some writers multiply the expression with constants, e.g. $1/(2\pi)$. The use of $W \equiv e^{-2\pi i/N}$ means that the end result is compatible with output from the Microsoft Excel Data Analysis Toolpack.

Filter

[MnFilter]

Returns an array of elements that contains some or all of the elements of *InputArray*.

If *IncludeMatches* is true then elements of *InputArray* are included in the output array only if the relevant element contains *MatchValue*. If IncludeMatches is false then elements of *InputArray* are included in the output array only if the relevant element does not contain *MatchValue*.

If *IsCaseSensitive* is true then function uses ‘binary’ comparison so differentiates between upper and lower case letters, otherwise uses ‘text’ comparison and so does not differentiate between upper and lower case letters.

FinanceDDB
[MnFinanceDDB]

Returns the depreciation of an asset for a specific time period using a specified depreciation methodology

FinanceFV
[MnFinanceFV]

Returns the future value of an annuity-style cash flow involving periodic fixed payments and a term-independent interest rate

FinanceIPmt
[MnFinanceIPmt]

Returns the interest payment for a given period for an annuity-style cash flow involving periodic fixed payments and a term-independent interest rate

FinanceIRR
[MnFinanceIRR]

Returns the internal rate of return for a series of periodic payments and receipts

FinanceMIRR
[MnFinanceMIRR]

Returns a modified internal rate of return for a series of periodic payments and receipts taking into account financing and reinvestment rates

FinanceNPer
[MnFinanceNPer]

Returns the number of periods over which a set of annuity-style cash flows will last

FinanceNPV
[MnFinanceNPV]

Returns the net present value of a series of periodic payments and receipts

FinancePmt
[MnFinancePmt]

Returns the overall fixed payment (interest and principal) for an annuity-style cash flow involving periodic fixed payments and a term-independent interest rate

FinancePPmt

[MnFinancePPmt]

Returns the principal payment for a given period for an annuity-style cash flow involving periodic fixed payments and a term-independent interest rate

FinancePV

[MnFinancePV]

Returns the present value of an annuity-style cash flow involving periodic fixed payments and a term-independent interest rate

FinanceRate

[MnFinanceRate]

Returns the interest rate (per period) for an annuity cash flow

FinanceSLN

[MnFinanceSLN]

Returns the straight line depreciation of an asset for a single period (is same for each period).

FinanceSYD

[MnFinanceSYD]

Returns the sum of digits depreciation of an asset for a given period

FInv

[MnFInv]

Returns the inverse distribution function (i.e. quantile function) of the *F* distribution. Provided for compatibility with FINV, F.INV and F.INV.RT functions in Microsoft Excel 2010 and later.

Note FINV(q) = F.INV.RT(q) = F.INV(1-q) and F.INV(q) = MnFInv

Please also bear in mind that this function accepts non-integral values of *nu1* and *nu2* whereas the Excel functions are limited to integral values.

Equivalent to calling MnProbDistQuantile using as the distribution parameter “f”.

Fisher
[MnFisher]

Returns the Fisher transform, i.e.

$$z = \frac{1}{2}\frac{\log(1+x)}{\log(1-x)}$$

Provided for compatibility with the FISHER function in Microsoft Excel and some other spreadsheet systems.

FisherInv
[MnFisherInv]

Returns the inverse Fisher transform, i.e.

$$z = \frac{e^{2x}-1}{e^{2x}+1}$$

Provided for compatibility with the FISHERINV function in Microsoft Excel and some other spreadsheet systems.

Floor
[MnFloor]

Rounds a number towards negative infinity, i.e. 7.1 goes to 7, -7.1 to -8. Same as MnInt.

Forecast
[MnForecast]

Returns the predicted value from a linear regression, i.e. $a + bx$ where:

$$a = \bar{y} - b\bar{x}$$
$$b = \frac{\sum_{i=1}^{n}(y-\bar{y})(x-\bar{x})}{\sum_{i=1}^{n}(x-\bar{x})^2}$$
$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n}x_i \qquad \bar{y} = \frac{1}{n}\sum_{i=1}^{n}y_i$$

Format
[MnFormat]

Returns *InputValue* (a numerical value) as a string formatted using a format defined as per Microsoft formatting conventions.

FormatNumber

[MnFormatNumber]
Returns *InputValue* as a string, formatted appropriately.

The number of digits after the decimal point is defined by *NumberDigitsAfterDecimalPoint*. If *IncludeLeadingDigit* is true then any fractional values will be formatted as 0.xxx, otherwise the leading 0 will not be included If *UseParensForNegatives* is true then negative numbers will be bracketed by parentheses.

FormatPercent

[MnFormatPercent]

Returns *InputValue* as a string formatted as a percentage, i.e. multiplied by 100 and followed by a '%' sign.

The number of digits after the decimal point is defined by *NumberDigitsAfterDecimalPoint*. If *IncludeLeadingDigit* is true then any fractional values will be formatted as 0.xxx, otherwise the leading 0 will not be included If *UseParensForNegatives* is true then negative numbers will be bracketed by parentheses.

Frequency

[MnFrequency]

Returns an array showing the number of entries in *DataValues* that falls within bins defined by *BinValues*.

The output array contains one more entry than the number of entries in *BinValues*. The first entry of the output array indicates the number of entries that are <= the smallest entry in *BinValue*, the second the number that are > the smallest and <= the next smallest entry in *BinValues* etc. and the last entry of the output array the number of entries that are > the largest value of in *BinValues*.

FromImproperFraction

[MnFromImproperFraction]

Returns a number corresponding to the input proper fraction, i.e. y where

$$y = \frac{a}{b}$$

where the input array has 2 integer entries $= (a, b)$

FromProperFraction

[MnFromProperFraction]

Returns a number corresponding to the input proper fraction, i.e. y where

$$y = a + \frac{b}{c}$$

where the input array has 3 integer entries $= (a, b, c)$

FromSexagesimal

[MnFromSexagesimal]

Returns the value (as a real number) of a number expressed in sexagesimal format (taking as an input an array with 3 elements representing the degrees, minutes and seconds (angle) or hours, minutes and seconds (time) respectively. Please bear in mind that a negative number expressed in sexagesimal format such as -3° 4′ 22.5″ corresponds to the number $-(3 + 4/60 + 22.5/3600)$, i.e. the second and third entries should be positive and the sign of the first is applied to all three.

FTest

[MnFTest]

Returns the probability of an *F*-Test statistic being less than the observed statistic (if the standard null hypothesis applies)

FunctionExampleParameterString

[MnFunctionExampleParameterString]

Returns a string that can be used with a given function as an example expression evaluation using MnEvaluate.

FunctionNumberOfParameters

[MnFunctionNumberOfParameters]

Returns the number of parameters that a given Nematrian website function recognises

FunctionOutputArrayDimension

[MnFunctionOutputArrayDimension]

Returns the number of (array) dimensions of the output of a given Nematrian website function (if it does not throw an error during evaluation)

FunctionOutputArrayType

[MnFunctionOutputArrayType]

Returns a flag that indicates whether the output of a given Nematrian website function is an array, or a single value. The flag is “A” if is an array (use MnFunctionOutputArrayDimension to indicate how many dimensions it then has) or is “S” if it is a single value.

FunctionOutputType

[MnFunctionOutputType]

Returns the type of output of a given Nematrian website function (e.g. “String”, “Double”, “Date”, “Boolean”, “Integer”)

FunctionParameterArrayType

[MnFunctionParameterArrayType]

Returns “A” if the given Nematrian website function expects an array as input for the given parameter, or “S” if it expects a single value

FunctionParameterDescription

[MnFunctionParameterDescription]

Returns a short description of the relevant parameter of a given Nematrian website function

FunctionParameterExample

[MnFunctionParameterExample]

Provides an example that can be used as an input for the relevant parameter of a given Nematrian website function

FunctionParameterName

[MnFunctionParameterName]

Returns the name by which the relevant parameter of a given Nematrian website function is generally recognised

FunctionParameterType

[MnFunctionParameterType]

Returns the type of input type recognised for the relevant parameter of a given Nematrian website function (e.g. “String”, “Double”, “Date”, “Boolean”, “Integer”)

Gamma

[MnGamma]

The gamma function, $\Gamma(x)$ is defined as:

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

The gamma function can be thought of as the extension of the factorial to the entire real (or complex) number set. For non-negative integers it is merely the familiar factorial function, $n!$, but offset by 1, i.e. $n! = \Gamma(n+1)$. Like factorials, it satisfies the following recurrence relationship:

$$\Gamma(x+1) = x\Gamma(x)$$

The Nematrian website approximates the gamma function using a so-called Lanczos approximation, see also Press *et al.* (2007), Toth (2004) or Wikipedia: Lanczos approximation.

$$\Gamma(z) = \left(\frac{\sqrt{2\pi}}{z}\right)\left(c_0 + \sum_{i=1}^{n} \frac{c_i}{z+i}\right)\left(z + \gamma + \frac{1}{2}\right)^{\left(z+\frac{1}{2}\right)} e^{-\left(z+\gamma+\frac{1}{2}\right)}$$

The particular Lanczos approximation the Nematrian website uses involves:

$$\begin{gathered} n = 6 \\ \gamma = 5 \\ c_0 = 1.00000000019001 \\ c_1 = 76.1800917294714 \\ c_2 = -86.5053203294167 \\ c_3 = 24.0140982408309 \\ c_4 = -1.23173957245015 \\ c_5 = 1.20865097386617 \times 10^{-3} \\ c_6 = -5.395239384953 \times 10^{-6} \end{gathered}$$

For large z there is a risk of overflow, which can be mitigated by using the MnLogGamma function, defined as $\ln\Gamma(z) = \log(\Gamma(z))$.

The Lanczos approximation is valid for arguments in the right complex half-plane, but can be extended to the entire complex plane (where the function is not singular) using the reflection formula, i.e.

$$\Gamma(1-z)\Gamma(z) = \frac{\pi}{\sin \pi z}$$

See also MnCGamma.

GammaDistCdf

[MnGammaDistCdf]

Returns the cumulative distribution function of the gamma distribution. Provided for compatibility with GAMMADIST and GAMMA.DIST (with cumulative parameter set to true) functions in Microsoft Excel 2010 and later.

Equivalent to calling MnProbDistCdf using as the distribution parameter “gamma”.

GammaDistPdf
[MnGammaDistPdf]

Returns the probability density function of the gamma distribution. Provided for compatibility with GAMMADIST and GAMMA.DIST (with cumulative parameter set to false) functions in Microsoft Excel 2010 and later.

Equivalent to calling MnProbDistPdf using as the distribution parameter "gamma".

GammaInv
[MnGammaInv]

Returns the inverse distribution function (i.e. quantile function) of the gamma distribution. Provided for compatibility with GAMMA.INV function in Microsoft Excel 2010 and later.

Equivalent to calling MnProbDistQuantile using as the distribution parameter "gamma".

GammaLn
[MnGammaLn]

Returns the natural logarithm of the gamma function,

Provided for compatibility with GAMMALN function in Excel. Returns the same as MnLogGamma.

Gauss
[MnGauss]

Returns the Gauss function, i.e. the cumulative standard normal distribution function less 0.5.

Provided for compatibility with the GAUSS function in Microsoft Excel 2013 and later.

GaussianMixtureModellingSA
[MnGaussianMixtureModellingSA]

Returns the result (as a single array) of applying a Gaussian mixture modelling analysis, as per Press *et al.* (2007).

In Gaussian mixture modelling, we have *n* *m*-dimensional datapoints (where *n* = *NumberOfSeries*, *m* = *NumberOfDataPointsPerSeries*) and we assume that each datapoint may have come from one of a number (*NumberOfMixtureComponents*) of different multi-variate Normal (i.e. Gaussian) probability distributions. We use the EM algorithm to identify the Gaussian mixture model with that number of Gaussian distributions that best fits the data in a maximum likelihood sense. To do so we provide initial starting means (i.e. centres) for each cluster, an initial value to ascribe to the covariance matrix terms and a termination criterion which stops the algorithm when an iteration appears to have added very little to the (log) likelihood. Also included is a backstop *Itermax*, which stops the algorithm carrying out a very large number of iterations.

InputData is a 2d array of size *NumberOfSeries* x *NumberOfDataPointsPerSeries* and *StartMeans* is a 2d array of size *NumberOfClusters* x *NumberOfSeries*.

The output is an array with 3 + *NumberOfClusters* x *NumberOfSeries* + 1 + *NumberOfClusters* x *NumberOfSeries* x *NumberOfSeries* + *NumberOfClusters* + *NumberOfDataPointsPerSeries* x *NumberOfClusters* as follows:

First 3 values = *NumberOfSeries*, *NumberOfDataPointsPerSeries* and *NumberOfMixtureComponents*

Following *NumberOfClusters* x *NumberOfSeries* values = means (centres) of each mixture component

Next value = Log likelihood of the data given the selected Gaussian mixture

Following *NumberOfClusters* x *NumberOfSeries* x *NumberOfSeries* values = covariances of each mixture component

Following *NumberOfClusters* = probability of a random datapoint being drawn from a given mixture component

Following *NumberOfDataPointsPerSeries* x *NumberOfClusters* values = probability of each individual datapoint being in a particular mixture component.

See also *k*-means clustering (which can be thought of as akin to a simplified version of Gaussian mixture modelling) and example data series for Gaussian mixture modelling and *k*-means clustering.

The algorithm used is derived from Press *et al.* (2007).

WARNING

For some data series used to tested the algorithm the log likelihood function appears to have many local maxima using the implementation of the EM algorithm by the Nematrian website. This means that the precise mixture selected is sensitive to the starting values inserted into the algorithm, in particular here the initial starting means. This reduces the usefulness of the algorithm for finding robust estimates of the different mixture components.

GCD

[MnGCD]

Returns the greatest common divisor of two strictly positive integers a and b, i.e. the largest integer that integrally divides both a and b.

GCDExtended

[MnGCDExtended]

Returns an array whose first element is the greatest common divisor, g, of two strictly positive integers a and b, i.e. the largest integer that integrally divides both a and b (see MnGCD) and whose second and third elements are the Bezout coefficients, i.e. the (lowest positive) values of x and y such that $ax + by = g$

GeoMean
[MnGeoMean]

Returns the geometric mean of an array, i.e. if the elements of the array are x_i (for $i = 1, \dots, n$) then the geometric mean is y where:

$$y = \sqrt[n]{\prod_{i=1}^{n} x_i} = \exp\left(\frac{1}{n}\sum_{i=1}^{n} \log x_i\right)$$

The function returns an error if any $x_i \leq 0$

Provided for compatibility with the GEOMEAN function in Microsoft Excel and some other spreadsheet systems.

GetChar
[MnGetChar]

Returns the character at *CharPosition* in *InputString*

GT
[MnGT]

Returns TRUE if the first argument is strictly greater than the second, and FALSE otherwise (i.e. equivalent to the > operator).

Provided for compatibility with the GT function in Google Drive.

GTE
[MnGTE]

Returns TRUE if the first argument is greater than or equal to the second, and FALSE otherwise (i.e. equivalent to the >= operator).

Provided for compatibility with the GTE function in Google Drive.

HaltonSequence
[MnHaltonSequence]

Returns a Halton quasi-random sequence, where *p* is the base prime of the sequence, *n* is the number of elements in the sequence returned by the function and *istart* is the point in the sequence where we start to return elements.

To calculate the *j*'th element of a Halton sequence element (with *istart* = 0) we write *j* as a number in base *p* (*p* needs to be prime), e.g. 17 base 3 is 122. We then reverse the sequence and put a decimal point radix *p* in front of it, so e.g. the 17'th element of the Halton sequence in base 3 is 0.221 base 3, i.e. (2*9 + 2*3 + 1*1)/27. Different (positive) values of *istart* offset the elements of the sequence that are returned by the function.

Halton sequences are examples of quasi-random, i.e. low discrepancy, sequences that are more evenly spread out in multiple dimensions (if, for Halton sequences, the base primes are different for the different dimensions) than would be the case if elements of the sequences are chosen 'randomly', e.g. using merely the MnRand function (or the equivalent MnProbDistRand with *DistributionName* = "uniform" and *ParamValues* = {0, 1}). Every time the number of digits in *j* increases, *j*'s digits become a factor of *p* more finely meshed.

This added smoothness means that in some circumstances they offer better convergence properties than basic Monte Carlo simulation techniques but at the cost of it then being less clear what these convergence properties are.

This function does not trap for whether *p* is prime.

HarMean

[MnHarMean]

Returns the harmonic mean of an array, i.e. if the elements of the array are x_i (for $i = 1, \dots, n$) then the harmonic mean is y where:

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

The function returns an error if any $x_i \leq 0$

Provided for compatibility with the HARMEAN function in Microsoft Excel and some other spreadsheet systems.

HerfindahlHirschmanIndex

[MnHerfindahlHirschmanIndex]

Returns the Herfindahl-Hirschmann Index ('HHI') of market concentration (with market shares expressed as fractions, so maximum HHI is unity), see Herfindahl-Hirschman Index

Hex2Dec

[MnHex2Dec]

Converts a string representing the number in hexadecimal form (i.e. base 16 with digits being 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F, although lower case letters a-f are also accepted instead of A-F) to an (unsigned) integer. Similar to Excel's HEX2DEC function but accepting higher input values and rejecting negative input values.

HexadecimalNumerals

[MnHexadecimalNumerals]

Returns a string array containing a list of the hexadecimal numerals recognised by the site in ascending order

Hour

[MnHour]

Returns the hour of a given date/time

Iif

[MnIif]

IIf(a, b, c) returns *b* if *a* is true, otherwise returns *c*

IncompleteBeta

[MnIncompleteBeta]

The incomplete beta function $B_x(p,q)$ is defined as:

$$B_x(p,q) = \int_0^x t^{p-1}(1-t)^{q-1}dt \qquad (p,q > 0, 0 \leq x \leq 1)$$

IncompleteGamma

[MnIncompleteGamma]

The incomplete gamma function, $P(a,x)$ is defined as:

$$P(a,x) = \frac{1}{\Gamma(a)}\int_0^x t^{a-1}e^{-t}dt$$

Here $\Gamma(a)$ is the gamma function.

If $x < a + 1$ then the website evaluates the incomplete gamma function using the following power series expansion:

$$P(a,x) = \Gamma(a)e^{-x}x^a \sum_{n=0}^{\infty} \frac{\Gamma(a)}{\Gamma(a+1+n)}x^n$$

If $x \geq a + 1$ then the website evaluates the function using the following continued fraction:

$$P(a,x) = \Gamma(a)e^{-x}x^{a}\left(\frac{1}{x+a-a-}\;\frac{1\cdot(1-a)}{x+3-a-}\;\frac{2\cdot(2-a)}{x+5-a-}\;\cdots\right)$$

IndexNonBlankStringArray

[MnIndexNonBlankStringArray]

Returns an (integer) array containing the location (base 0) indicating where in the *Index* array each entry in the *ArrayToBeIndexed* appears.

Elements of the *Index* array must be unique. If the entry in the *ArrayToBeIndexed* is blank then the corresponding entry in the output array is -2. If it does not appear in the *Index* array then the corresponding entry in the output array is -1. If it does appear then will be some value 0 to $m-1$ where there are m entries in the *Index* array.

There is no need for either the *Index* array or the *ArrayToBeIndexed* to be pre-sorted.

InsertIntoArray

[MnInsertIntoArray]

Returns a numerical array into which an element has been inserted at a particular place in the *InputArray*

InStr

[MnInStr]

Returns the (starting) position of the first occurrence of one string (*MatchString*) within another string (*InputString*), working forwards from the first character of *InputString*.

If *IsCaseSensitive* is true then function uses 'binary' comparison so differentiates between upper and lower case letters, otherwise uses 'text' comparison and so does not differentiate between upper and lower case letters.

InStrPart

[MnInStrPart]

Returns the (starting) position of the first occurrence of one string (*MatchString*) within the part of another string (*InputString*), working forwards from the *StartPosition* character of *InputString*.

If *IsCaseSensitive* is true then function uses 'binary' comparison so differentiates between upper and lower case letters, otherwise uses 'text' comparison and so does not differentiate between upper and lower case letters.

InStrRev

[MnInStrRev]

Returns the (starting) position of the last occurrence of one string (*MatchString*) within another string (*InputString*), working backwards from the last character of *InputString*.

If *IsCaseSensitive* is true then function uses 'binary' comparison so differentiates between upper and lower case letters, otherwise uses 'text' comparison and so does not differentiate between upper and lower case letters.

InStrRevPart

[MnInStrRevPart]

Returns the (starting) position of the last occurrence of one string (*MatchString*) within the part of another string (*InputString*), working backwards from end of *InputString* but only as far as the *StartPosition* character of *InputString*.

If *IsCaseSensitive* is true then function uses 'binary' comparison so differentiates between upper and lower case letters, otherwise uses 'text' comparison and so does not differentiate between upper and lower case letters.

Int

[MnInt]

Returns the integer part as per .NET (or Excel), i.e. returns the number rounded down to the nearest integer, e.g. an input of 7.1 returns 7 whilst an input of -7.1 returns -8. Same as MnFloor.

Intercept

[MnIntercept]

Returns the intercept from a linear regression, i.e. a where:

$$a = \bar{y} - b\bar{x}$$

$$b = \frac{\sum_{i=1}^{n}(y - \bar{y})(x - \bar{x})}{\sum_{i=1}^{n}(x - \bar{x})^2}$$

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

IntersectionOfDateSets

[MnIntersectionOfDateSets]

Returns a Date array representing the elements of the union of the distinct members of two input arrays, e.g. if InputArray1 is {3 Jan 1980, 5 Jan 1980, 5 Jan 1980, 4 Jan 1980} and InputArray2 is {2 Jan 1980, 5 Jan 1980, 4 Jan 1980, 4 Jan 1980} then returns {5 Jan 1980, 4 Jan 1980}

IntersectionOfIntegerSets

[MnIntersectionOfIntegerSets]

Returns an Integer array representing the elements of the union of the distinct members of two input arrays, e.g. if InputArray1 is {3, 5, 5, 4} and InputArray2 is {2, 5, 4, 4} then returns {5, 4}

IntersectionOfSets

[MnIntersectionOfSets]

Returns a numeric array (i.e. an array of ‘double precision’ values) representing the elements of the union of the distinct members of two input arrays, e.g. if *InputArray1* is {3.0, 5.0, 5.0, 4.0} and *InputArray2* is {2.0, 5.0, 4.0, 4.0} then returns {5.0, 4.0}

InverseNormal

[MnInverseNormal]

Returns $N^{-1}(p)$, the inverse of the unit normal cumulative distribution function. This is the value z for which $N(z) = p$ where $N(z)$ is the cumulative unit Normal distribution function, see MnCumulativeNormal. The function is not defined if $p \notin (0,1)$.

In practice, the website uses a rational approximation to the above function developed by Acklam (2005). This is not the same approach used in some versions of Microsoft Excel which appear to calculate this function using an iterative search method based on the methodology Microsoft use to calculate $N(z)$. Otherwise the equivalent function in Microsft Excel is NORMSINV.

$$N^{-1}(p) \cong \begin{cases} \dfrac{c_1q^5 + c_2q^4 + c_3q^3 + c_4q^2 + c_5q + c_6}{d_1q^4 + d_2q^3 + d_3q^2 + d_4q + 1} & if\ p < 0.02425 \\ \dfrac{a_1r^5 + a_2r^4 + a_3r^3 + a_4r^2 + a_5r + a_6}{b_1r^4 + b_2r^3 + b_3r^2 + b_4r + 1}\left(p - \dfrac{1}{2}\right) & if\ \left|p - \dfrac{1}{2}\right| < 0.47575 \\ \dfrac{c_1q^5 + c_2q^4 + c_3q^3 + c_4q^2 + c_5q + c_6}{d_1q^4 + d_2q^3 + d_3q^2 + d_4q + 1} & if\ p > 1 - 0.02425 \end{cases}$$

where:

$$q = \begin{cases} \sqrt{-2\log(p)} & if\ p < 0.02425 \\ \sqrt{-2\log(1-p)} & if\ p > 1 - 0.02425 \end{cases}$$

and:

$a_1 = -39.6968302866538$, $a_2 = 220.946098424521$, $a_3 = -275.928510446969$,
$a_4 = 138.357751867269$, $a_5 = -30.6647980661472$, $a_6 = 2.50662827745924$

$b_1 = -54.4760987982241$, $b_2 = 161.585836858041$,
$b_3 = -155.698979859887$, $b_4 = 66.8013118877197$, $b_5 = -13.2806815528857$

$c_1 = -0.00778489400243029$, $c_2 = -0.322396458041136$,
$c_3 = -2.40075827716184$, $c_4 = -2.54973253934373$
$c_5 = 4.37466414146497$, $c_6 = 2.93816398269878$

$d_1 = 0.00778469570904146$, $d_2 = 0.32246712907004$,

$d_3 = 2.445134137143$, $d_4 = 3.75440866190742$

IsAlphabetic

[MnIsAlphabetic]

Returns TRUE if all the characters of *s* are alphabetic, otherwise returns FALSE

IsAlphanumeric

[MnIsAlphanumeric]

Returns TRUE if all the characters of *s* are alphanumeric, otherwise returns FALSE

IsEven

[MnIsEven]

Returns TRUE if *x* is even, otherwise returns FALSE

IsOdd

[MnIsOdd]

Returns TRUE if *x* is odd, otherwise returns FALSE

IsPowerOf2

[MnIsPowerOf2]

Returns TRUE if *n* is an integral power of 2, otherwise returns FALSE

IsRecognisedPhysicalUnit

[MnIsRecognisedPhysicalUnit]

Returns TRUE if the *UnitName* is a physical unit recognised by the Nematrian website (i.e. one of the values of the array outputted by MnRecognisedPhysicalUnits), otherwise returns FALSE.

JohnsonSUMOMEstimates

[MnJohnsonSUMOMEstimates]

Returns method of moments estimators for the Johnson SU distribution, i.e. given a (sample) mean, standard deviation, skew and excess kurtosis will find (if they exist) the $\gamma, \delta, \lambda, \xi$ for which the Johnson SU distribution has these as its equivalent population characteristics.

See MnProbDistMLE and variants for maximum likelihood estimation tools for this distribution.

Join

[MnJoin]

Returns a string formed by joining substrings contained in an array, with consecutive substrings separated by a delimiter (which might be nothing if the input for *Delimiter* is an empty string)

KendalTauCoefficient

[MnKendalTauCoefficient]

Returns the Kendal's Tau $c_{1,2}$ between two series, $x_{1,t}$ and $x_{2,t}$.

If $(x_1, y_1), \dots, (x_n, y_n)$ are a set of joint observations from two random variables X and Y then pairs (x_i, y_i) and (x_j, y_j) $i \neq j$ are said to be 'concordant' if both $x_i > x_j$ and $y_i > y_j$ (or if both $x_i < x_j$ and $y_i < y_j$). They are said to be 'discordant' if $x_i > x_j$ and $y_i < y_j$ or if $x_i < x_j$ and $y_i > y_j$. Then the elements of the matrix are as follows, where each entry separately considers all possible pairs of i, j for $i = 1, \dots, n, j = 1, \dots, n$ and $i \neq j$

$$c_{1,2} = \frac{(number\ of\ concordant\ pairs) - (number\ of\ discordant\ pairs)}{(number\ of\ concordant\ pairs) + (number\ of\ discordant\ pairs)}$$

N.B. There are various possible approaches to ties. The one implemented here reduces the denominator so that we still have $-1 \leq \tau \leq +1$ and in the presence of ties, τ can take either extreme value in some circumstances.

KendalTauCoefficients

[MnKendalTauCoefficients]

Returns an array of m^2 elements corresponding to the elements of the matrix, C, formed by Kendal tau coefficients for each pairing of m series that are provided to the function as an array of mn elements, where the first n elements correspond to the terms in the first series, the next n elements to the corresponding terms in the second series etc.

If $(x_1, y_1), \dots, (x_n, y_n)$ are a set of joint observations from two random variables X and Y then pairs (x_i, y_i) and (x_j, y_j) $i \neq j$ are said to be 'concordant' if both $x_i > x_j$ and $y_i > y_j$ (or if both $x_i < x_j$ and $y_i < y_j$). They are said to be 'discordant' if $x_i > x_j$ and $y_i < y_j$ or if $x_i < x_j$ and $y_i > y_j$. Then the elements of the matrix are as follows, where each entry separately considers all possible pairs of i, j for $i = 1, \dots, n, j = 1, \dots, n$ and $i \neq j$

$$c_{a,b} = \frac{(number\ of\ concordant\ pairs) - (number\ of\ discordant\ pairs)}{(number\ of\ concordant\ pairs) + (number\ of\ discordant\ pairs)}$$

N.B. There are various possible approaches to ties. The one implemented here reduces the denominator so that we still have $-1 \leq \tau \leq +1$ and in the presence of ties, τ can take either extreme value in some circumstances.

KMeansClusteringSA

[MnKMeansClusteringSA]

Returns the result (in a single array) of applying a *k*-means clustering analysis, as per Press et al. (2007).

In *k*-means clustering, we have *n* *m*-dimensional datapoints (where *n* = *NumberOfSeries*, *m* = *NumberOfDataPointsPerSeries*) and we ascribe them to clusters depending on how near they are (in a spherical Euclidean sense) to potential cluster centres. We need to specify the number of clusters (NumberOfClusters) and some initial starting means (i.e. centres) for each cluster. The algorithm then uses a variant of the EM algorithm to find which datapoints belong to which clusters and where the cluster centres need to be to minimise the sum of the distance that the datapoints are away from their cluster centres.

InputData is a 2d array of size *NumberOfSeries* x *NumberOfDataPointsPerSeries* and *StartMeans* is a 2d array of size *NumberOfClusters* x *NumberOfSeries*.

The output is an array with 3 + *NumberOfClusters* x *NumberOfSeries* + *NumberOfDataPointsPerSeries* as follows:

First 3 values = *NumberOfSeries*, *NumberOfDataPointsPerSeries* and *NumberOfMixtureComponents*

Following *NumberOfClusters* x *NumberOfSeries* values = locations of means (centres) of each cluster

Following *NumberOfDataPointsPerSeries* values = cluster to which datapoint has been assigned (counting from 0).

The probability that a datapoint selected randomly is in a particular cluster can be found if needed by counting the proportion of the datapoints assigned to different clusters.

See also Gaussian mixture modelling (which can be thought of as akin to a generalisation of *k*-means clustering) and example data series for Gaussian mixture modelling and k-means clustering.

Kurt

[MnKurt]

Returns the (sample) kurtosis, γ_2 (or more precisely the 'excess' kurtosis) of a series x_i (for $i = 1, \dots, n$), calculated as follows:

$$\gamma_2 = \frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum \left(\frac{x_i - \bar{x}}{s} \right)^4 - \frac{3(n-1)^2}{(n-2)(n-3)}$$

Here, $\bar{x}$ is the mean and s is the (sample) standard deviation of the series

The equivalent function in Microsoft Excel is KURT.

LCase

[MnLCase]

Returns the lower case equivalent of *InputString* (i.e. replacing e.g. 'A' by 'a' etc.)

IsRecognisedPhysicalUnit

[MnLCM

Returns the least common multiple of two strictly positive integers *a* and *b*, i.e. the smallest integer that can be integrally divided by both *a* and *b*.

LeastSquaresGeneralisedCurveFit

[MnLeastSquaresGeneralisedCurveFit]

Returns an array consisting of the (interpolated) values corresponding to y_i derived by carrying out a least squares generalised curve fit to the y_i giving weights w_i to each point i. Such a curve fit identifies the choice of a_j that minimises $\sum w_i \left(\sum a_j f_{i,j} - y_i\right)^2$ where the $f_{i,j}$ correspond to the values (for a given i of the j basis functions used in the generalised least squares curve fit. A traditional regression in effect involves choosing as basis functions $f_{i,0} \equiv 1$ and $f_{i,1} \equiv x_i$.

LeastSquaresPolynomialCurveFit

[MnLeastSquaresPolynomialCurveFit]

Returns an array consisting of the (interpolated) values corresponding to y_i derived by carrying out a least squares curve fit (see here for more details) where the fit is based on the best fitting polynomial up to order *PolynomialOrder*. Different weights w_i can be assigned to each point i. Is a special case of MnLeastSquaresGeneralisedCurveFit.

Left

[MnLeft]

Returns a string containing the *n* left-most characters of *InputString*. If *n* = 0 then returns a blank string. If *n* is greater than the number of characters in *InputString* then it returns the entire *InputString*.

Len

[MnLen]

Returns the length of (i.e. number of characters in) *InputString*

LibraryVersion

[MnLibraryVersion]

Returns the version of the website function library currently accessible through the website.

LibraryVersionMessage

[MnLibraryVersionMessage]

Returns a standard user message relating to the version of the website function library being used.

LinearInterpolation
[MnLinearInterpolation]

Returns a value derived by linear interpolation from array, $y = (y_1, \dots, y_n)$ based on position of value to interpolate from (q, say) versus elements of array, $x = (x_1, \dots, x_n)$. The elements of x need to be in increasing order. If $q < x_1$ or $q > x_n$ then returns y_1 or y_n respectively, rather than extrapolating beyond the end of the range spanned by x.

Log
[MnLog]

Returns the natural logarithm (logarithm to base *e* = 2.71828…) of a (real, positive) number

Log10
[MnLog10]

Returns the logarithm to base 10 of a (real, positive) number

LogBaseN
[MnLogBaseN]

Returns the logarithm of a (real, positive) number, *x*, in base *n* (where *n* is a real positive number).

LogGamma
[MnLogGamma]

The log gamma function, $\ln\Gamma(z)$ is defined as $\ln\Gamma(z) = \log\big(\Gamma(z)\big)$, where $\Gamma(z)$ is calculated as per MnGamma.

Computation of the log gamma function is less likely to result in an overflow error for large z.

The equivalent function in recent versions of Microsoft Excel is GAMMALN. Returns the same as MnGammaLn

LognormDistCdf
[MnLognormDistCdf]

Returns the cumulative distribution function of the log-normal distribution. Provided for compatibility with LOGNORMDIST and LOGNORM.DIST (with cumulative parameter set to true) functions in Microsoft Excel 2010 and later.

Equivalent to calling MnProbDistCdf using as the distribution parameter “lognormal”.

LognormDistPdf

[MnLognormDistPdf]

Returns the probability density function of the log-normal distribution. Provided for compatibility with LOGNORM.DIST (with cumulative parameter set to false) functions in Microsoft Excel 2010 and later.

Equivalent to calling MnProbDistPdf using as the distribution parameter “lognormal”.

LognormInv

[MnLognormInv]

Returns the inverse distribution function (i.e. quantile function) of the log-normal distribution. Provided for compatibility with LOGINV and LOGNORM.INV functions in Microsoft Excel 2010 and later.

Equivalent to calling MnProbDistQuantile using as the distribution parameter “lognormal”.

LogSeries

[MnLogSeries]

Returns a series, y_i, whose elements are the natural logarithms of the elements of the input series, i.e.:

$$y_i = \log x_i$$

LSet

[MnLSet]

Returns a string that is a left aligned version of *InputString*, i.e. the first *n* characters of *InputString*, padded out with spaces if *n* is longer than the length of *InputString*.

LT

[MnLT]

Returns TRUE if the first argument is strictly less than the second, and FALSE otherwise (i.e. equivalent to the < operator).

Provided for compatibility with the LT function in Google Drive.

LTE

[MnLTE]

Returns TRUE if the first argument is less than or equal to the second, and FALSE otherwise (i.e. equivalent to the <= operator).

Provided for compatibility with the LTE function in Google Drive.

LTrim
[MnLTrim]

Returns *InputString* but without any leading spaces. See also MnRTrim and MnTrim.

Mantissa
[MnMantissa]

Returns the mantissa of a number when expressed in scientific notation, e.g. 0.31 is written as 3.1×10^{-1} in scientific notation and thus has a mantissa of 3.1.

MatAntiSym
[MnMatAntiSym]

Returns the antisymmetrised version of a square matrix M = *SquareMat*, i.e. $(M - M^T)/2$

MatDet
[MnMatDet]

Returns the determinant of a square matrix, i.e. $det(A) = |A|$.

MatInverse
[MnMatInverse]

Returns the inverse of a (square) matrix, A^{-1}.

The equivalent function in Microsoft Excel is MINV.

MatLT
[MnMatLT]

Returns the lower triangular version of a (symmetric) square matrix M = *SquareMat*, i.e. $(M + M^T)/2$ for elements on or below the leading diagonal and zero for elements above it.

MatLUDecomposition
[MnMatLUDecomposition]

Returns the LU decomposition (without index and permutation sign) of a $n \times n$ matrix matrix. The LU decomposition is a $n \times n$ matrix from which we can construct lower and upper diagonal matrices L and U such that if the input matrix is A then $LU = A$. As per Press et al. (2007) the algorithm returns permuted lower and upper triangular matrices, with the permutations separately returned using MnMatLUIndexAndPerm.

Using the LU decomposition (together with the LU index and permutation sign) speeds up solution of matrix equations etc.

MatLUDet

[MnMatLUDet]

Returns the determinant of a square matrix, given as inputs its LU decomposition and the corresponding LU index and permutation sign (see MnMatLUDecomposition and MnMatLUIndexAndPerm).

MatLUEqnSolve

[MnMatLUEqnSolve]

Returns the solution, $\mathbf{x}$, to a matrix equation $\mathbf{Mx} = \mathbf{B}$ given as inputs the LU decomposition and the corresponding LU index and permutation sign of the square ($n \times n$) matrix $\mathbf{M}$ (see MnMatLUDecomposition and MnMatLUIndexAndPerm) and $\mathbf{B}$ is the $n \times m$ matrix to solve for. The output is an $n \times m$ matrix.

MatLUIndexAndPerm

[MnMatLUIndexAndPerm]

Returns an array of $n + 1$ elements giving the LU index and the permutation sign of a $n \times n$ matrix applicable to the LU decomposition returned by MnMatLUDecomposition. The algorithm used is as per Press et al. (2007) with the first n elements of the output being the index array and the $n + 1$'th element being the permutation sign (relevant for calculating the matrix determinant).

MatLUInverse

[MnMatLUInverse]

Returns the inverse of a square matrix, given as inputs its LU decomposition and the corresponding LU index and permutation sign (see MnMatLUDecomposition and MnMatLUIndexAndPerm).

MatLUVecSolve

[MnMatLUVecSolve]

Returns the solution, $\mathbf{x}$, to a matrix equation $\mathbf{Mx} = \mathbf{b}$ given as inputs the LU decomposition and the corresponding LU index and permutation sign of the square ($n \times n$ matrix $\mathbf{M}$ (see MnMatLUDecomposition and MnMatLUIndexAndPerm) and $\mathbf{b}$ is the vector (with n elements) to solve for. The output is an n element vector.

MatProduct

[MnMatProduct]

Returns the matrix product $P = AB$ of two matrices, A and B. If the matrices are codified as two dimensional arrays then the matrix product is calculated as:

$$P_{i,j} = \sum_k A_{i,k} B_{k,j}$$

The equivalent function in Microsoft Excel is MMULT.

MatSym

[MnMatSym]

Returns the symmetrised version of a square matrix M = *SquareMat*, i.e. $(M + M^T)/2$

MatTrace

[MnMatTrace]

Returns the *trace* of an $n \times n$ (i.e. square) matrix A, i.e. the sum of the elements on the main diagonal (from upper left to lower right) of A, i.e.:

$$tr(A) = a_{1,1} + \cdots + a_{n,n} = \sum_{i=1}^{n} a_{i,i}$$

where $a_{i,j}$ is the entry on the *i*'th row and *j*'th column of A.

The trace is a linear map which is invariant under cyclic permutations, e.g. in general $tr(ABC) \neq tr(BAC)$ but $tr(ABCD) = tr(BCDA) = tr(CDAB) = tr(DABC)$ etc.

MatUnit

[MnMatUnit]

Returns a unit square matrix (i.e. with 1's along the leading diagonal and 0's elsewhere).

Included for compatibility with MUNIT in Microsoft Excel and some other spreadsheet systems.

MatUT

[MnMatUT]

Returns the upper triangular version of a (symmetric) square matrix M = *SquareMat*, i.e. $(M + M^T)/2$ for elements on or above the leading diagonal and zero for elements below it.

Max
[MnMax]

Returns the maximum of two values

MaximumOfArray
[MnMaximumOfArray]

Returns the maximum of an array of values

MaximumOfDateArray
[MnMaximumOfDateArray]

Returns the maximum of an array of dates

Mean
[MnMean]

Returns the mean $\bar{x}$ of a series x_i (for $i = 1, \dots, n$), calculated as follows:

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

The equivalent function in Microsoft Excel is AVERAGE.

MeanAbsDevVsMean
[MnMeanAbsDevVsMean]

Returns, a, the mean absolute deviation versus the mean of a series x_i (for $i = 1, \dots, n$), calculated as follows:

$$a = \frac{\sum_{i=1}^{n} |x_i - \bar{x}|}{n}$$

(where $\bar{x}$ is the mean of the series)

See also MnMeanAbsDevVsMedian.

MeanAbsDevVsMedian
[MnMeanAbsDevVsMedian]

Returns, b, the mean absolute deviation versus the median of a series x_i (for $i = 1, \dots, n$), calculated as follows:

$$b = \frac{\sum_{i=1}^{n} |x_i - x_{med}|}{n}$$

(where x_{med} is the median of the series)

See also MnMeanAbsDevVsMean which provides another common way of measuring mean absolute deviation. However, the mean absolute deviation is minimised if it is measured relative to the median of the series rather than relative to its mean.

Median

[MnMedian]

Returns the median of a series, i.e. the value above and below both are one-half of the series. Is a special case of MnPercentile.

The equivalent function in Microsoft Excel is MEDIAN.

Mid

[MnMid]

Returns string formed by *n* consecutive characters from the *InputString*. If the *StartPosition* is larger than the number of characters in the *InputString* then returns a zero length string. If there are fewer than *n* characters in the *InputString* then returns the whole *InputString*.

Min

[MnMin]

Returns the minimum of two values

MinimumOfArray

[MnMinimumOfArray]

Returns the minimum of an array of values

MinimumOfDateArray

[MnMinimumOfDateArray]

Returns the minimum of an array of dates

Minus

[MnMinus]

Returns the difference between two numbers (i.e. equivalent to the – operator).

Provided for compatibility with the MINUS function in Google Drive.

Minute
[MnMinute]

Returns the minute of a given date/time

ModeOfArray
[MnModeOfArray]

If the array has a mode (i.e. a unique value which appears in the array more than any other value) then it returns the mode, otherwise it throws an error. To test whether an array has a mode use MnDoesArrayHaveMode.

Month
[MnMonth]

Returns the month of a given date/time

MonthNameEnglish
[MnMonthNameEnglish]

Returns the (English) name of a given month, possibly abbreviated

MortalityTableMaxAge
[MnMortalityTableMaxAge]

Returns the maximum age applicable to a given mortality table.

Currently the Nematrian website supports a range of UK mortality tables using nomenclature derived from terminology adopted by the UK's Continuous Mortality Investigation Bureau.

MortalityTableMinAge
[MnMortalityTableMinAge]

Returns the minimum age applicable to a given mortality table.

Currently the Nematrian website supports a range of UK mortality tables using nomenclature derived from terminology adopted by the UK's Continuous Mortality Investigation Bureau.

MortalityTableSex
[MnMortalityTableSex]

Returns the sex applicable to a given mortality table.

Currently the Nematrian website supports a range of UK mortality tables using nomenclature derived from terminology adopted by the UK's Continuous Mortality Investigation Bureau.

Multiply

[MnMultiply]

Returns the product of two numbers (i.e. equivalent to the * operator).

Provided for compatibility with the MULTIPLY function in Google Drive.

MultiplySeriesByConstant

[MnMultiplySeriesByConstant]

Returns a series, y_i, calculated by multiplying each element of an input series, x_i by a scalar constant, k, i.e.:

$$y_i = kx_i$$

MultiplyTwoSeriesElementByElement

[MnMultiplyTwoSeriesElementByElement]

Returns a series, z_i, calculated by multiplying term-by-term the elements of two input series, x_i and y_i, i.e.:

$$z_i = x_i y_i$$

MyBrowserFeature

[MnMyBrowserFeature]

Returns a specific recognised feature of the browser that is accessing the function (such as browser type or screen width), if available.

If you want to access a list of the values of all such features for the browser accessing the function then use MnMyBrowserFeatures. A list of recognised features is also available using MnRecognisedBrowserFeatures. A list of descriptions of each recognised browser feature is available using MnBrowserFeatureDescriptions.

MyBrowserFeatures

[MnMyBrowserFeatures]

Returns a list of all recognised features of the browser (such as browser type or screen width) that is accessing the function, if available.

If you want to access a single feature then use MnMyBrowserFeature. A list of recognised features is available using MnRecognisedBrowserFeatures. A list of descriptions of each recognised browser feature is available using MnBrowserFeatureDescriptions.

MyIPAddress

[MnMyIPAddress]

Returns the apparent Internet Protocol (IP) address (the 'HOST_ADDRESS') of the computer accessing the Nematrian website.

If the Nematrian website is being accessed by a proxy server then this may be blank or may be the IP address of the proxy server rather than the original computer on which the browser calling the Nematrian website is located. The MnMyProxyIPAddress.aspx web function may help to clarify the IP address of the actual calling computer in such circumstances.

MyIPRequestVariables

[MnMyIPRequestVariables]

Returns the complete list of variables request variables being passed to the Nematrian website when by the computer attempting to access the website. Its output is a string array of $2n$ entries, where each consecutive pair involves the name of the variable (e.g. 'HOST_ADDRESS') and what the variable contains.

MyProxyIPAddress

[MnMyProxyIPAddress]

Returns the Internet Protocol (IP) address of a computer accessing the Nematrian website via a proxy server, unless the proxy server is deliberately set up to hide this address. More precisely it returns the value of the 'HTTP_X_FORWARDED_FOR' variable included in the request coming into the server, if this variable exists.

If the Nematrian website is being accessed directly rather than via a proxy server then this function is likely to return a blank string, and the IP address is instead accessible via MnMyIPAddress.

NE

[MnNE]

Returns TRUE if two specified values are not equal and FALSE otherwise (i.e. equivalent to the $\neq$ operator).

Provided for compatibility with the NE function in Google Drive.

NonNegativeIntegerToBaseNArray

[MnNonNegativeIntegerToBaseNArray]

Returns an array consisting of the digits representing the value of integer x in base $nBase$. The array is in increasing powers of $nBase$. For example, if $x = 6$ and $nBase$ is 2 then the binary representation of 6 is “110”, so the array returned by the function consists of the elements 0, 1 and 1 respectively.

NormaliseArray

[MnNormaliseArray]

Returns an array which is the Normalised equivalent of the input array, i.e. if the input array is x_i then returns

$$y_i = \frac{x_i - m}{s}$$

where m is the mean of the input array and s is its (sample) standard deviation.

NormaliseWeightedArray

[MnNormaliseWeightedArray]

Returns an array which is the (weighted) Normalised equivalent of the input array, i.e. if the input array is x_i then returns

$$y_i = \frac{x_i - \tilde{x}}{\tilde{s}}$$

where $\tilde{x}$ is the weighted mean of the input array and $\tilde{s}$ is its weighted (sample) standard deviation, see WeightedMomentsAndCumulants.

NormalMLFit

[MnNormalMLFit]

Returns an array of two elements consisting of the maximum likelihood estimates of the mean, $\hat{\mu}_{ML}$, and standard deviation, $\hat{\sigma}_{ML}$, assuming that the observations in *InputData* are independent draws from a normal distribution $N(\mu, \sigma^2)$.

NormalTailFit

[MnNormalTailFit]

Returns an array of two elements consisting of the tail fit estimates of the mean, $\hat{\mu}_{TF}$, and standard deviation, $\hat{\sigma}_{TF}$, assuming that the observations, x_i in *InputData* are independent draws from a Normal distribution $N(\mu, \sigma^2)$ and x_i corresponds to quantile q_i and is given weight w_i where the q_i are in *Quantiles* and the w_i are in *Weights*.

See tail fitting of a normal distribution for further details.

Now

[MnNow]

Returns current date and time (of server)

NthRoot

[MnNthRoot]

Returns the n'th root of a (real) number, x, where n is an integer, i.e. returns y where:

$$y = x^{1/n} = \sqrt[n]{x}$$

NumberOfDaysInMonth

[MnNumberOfDaysInMonth]

Returns the number of days in the month specified by a string of form YYYYMM.

Oct2Dec

[MnOct2Dec]

Converts a string representing a number in octal form (i.e. base 8 with digits being 0, 1, 2, 3, 4, 5, 6 and 7) to an (unsigned) integer. Similar to Excel's OCT2DEC function but accepting higher input values and rejecting negative input values.

OctalNumerals

[MnOctalNumerals]

Returns a string array containing a list of the octal numerals recognised by the site in ascending order

OptimisedTrinomialLatticeProbs

[MnOptimisedTrinomialLatticeProbs]

Returns optimised transition probabilities as an output array $(p_1, p_2, p_3)^T$ given lattice levels $(x_1, x_2, x_3)^T$ which an underlying at the current lattice node can move to, given that we want the sum of the probabilities to add to *SumWts* (will almost always be 1, as otherwise the p_i are not actually probabilities), the weighted mean of the underlying at the end of this period to be *WtdMean* and the one-period volatility of the underlying to be *WtdStd*.

We thus want:

$$\begin{gathered} a = SumWts = p_1 + p_2 + p_3 \\ b = WtdMean = x_1 p_1 + x_2 p_2 + x_3 p_3 \\ c = WtdMean^2 + WtdStd^2 = x_1^2 p_1 + x_2^2 p_2 + x_3^2 p_3 \end{gathered}$$

Thus

$$p_1 = \frac{x_2 x_3 a - (x_2 + x_3)b + c}{(x_2 - x_1)(x_3 - x_1)}$$
$$p_2 = \frac{x_3 x_1 a - (x_3 + x_1)b + c}{(x_3 - x_2)(x_1 - x_2)}$$
$$p_3 = \frac{x_1 x_2 a - (x_1 + x_2)b + c}{(x_1 - x_3)(x_2 - x_3)}$$

OrderOfArray
[MnOrderOfArray]

Returns an integer array that indicates the sorted order (indexed from zero) of a (double) array passed to it, sorted in ascending order.

OrderOfDateArray
[MnOrderOfDateArray]

Returns an integer array that indicates the sorted order (indexed from zero) of a date array passed to it, sorted in ascending order.

OrderOfIntegerArray
[MnOrderOfIntegerArray]

Returns an integer array that indicates the sorted order (indexed from zero) of an integer array passed to it, sorted in ascending order.

OrderOfStringArray
[MnOrderOfStringArray]

Returns an integer array that indicates the sorted order (indexed from zero) of a string array passed to it, sorted in ascending order.

There are several different conventions that can be used when ordering strings (e.g. do numbers come before or after letters, do capital letters come before lower case letters). Nematrian merely borrows the standard included in Visual Studio. We suggest that you play with the example page applicable to this function, see below, to ascertain exactly what this convention involves.

ParseBooleanArrayIntoString
[MnParseBooleanArrayIntoString]

Returns a string containing an array derived from the (Boolean) *InputArray* with each element delimited by the chosen *Delimiter*.

ParseDoubleArrayIntoString

[MnParseDoubleArrayIntoString]

Returns a string containing an array derived from the (numeric, i.e. consisting of variables of type Double) *InputArray* with each element delimited by the chosen *Delimiter*.

ParseIntegerArrayIntoString

[MnParseIntegerArrayIntoString]

Returns a string containing an array derived from the Integer *InputArray* with each element delimited by the chosen *Delimiter*.

ParseStringArrayIntoString

[MnParseStringArrayIntoString]

Returns a string containing an array derived from the String *InputArray* with each element delimited by the chosen *Delimiter*.

ParseStringInto2dBooleanArray

[MnParseStringInto2dBooleanArray]

Returns a Boolean array containing elements derived from *InputString*, each row being delimited by the chosen *RowDelimiter* and each element within the row by a *RowElementDelimiter*, with number of entries in each row defined by number of elements identified in *RowFormat*. Strings that start with *StartStringToDefineRowsToIgnore* are ignored.

A blank *RowDelimiter* or *RowElementDelimiter* is treated in the same way as a blank *Delimiter* in MnParseStringIntoStringArray.

ParseStringInto2dDateUKArray

[MnParseStringInto2dDateUKArray]

Returns a Date array containing elements derived from *InputString*, each row being delimited by the chosen *RowDelimiter* and each element within the row by a *RowElementDelimiter*, with number of entries in each row defined by number of elements identified in *RowFormat*. Strings that start with *StartStringToDefineRowsToIgnore* are ignored.

A blank *RowDelimiter* or *RowElementDelimiter* is treated in the same way as a blank *Delimiter* in MnParseStringIntoStringArray.

ParseStringInto2dDoubleArray

[MnParseStringInto2dDoubleArray]

Returns a numeric (i.e. Double) array containing elements derived from *InputString*, each row being delimited by the chosen *RowDelimiter* and each element within the row by a *RowElementDelimiter*, with number of entries in each row defined by number of elements identified in *RowFormat*. Strings that start with *StartStringToDefineRowsToIgnore* are ignored.

A blank *RowDelimiter* or *RowElementDelimiter* is treated in the same way as a blank *Delimiter* in MnParseStringIntoStringArray.

ParseStringInto2dIntegerArray

[MnParseStringInto2dIntegerArray]

Returns an Integer array containing elements derived from *InputString*, each row being delimited by the chosen *RowDelimiter* and each element within the row by a *RowElementDelimiter*, with number of entries in each row defined by number of elements identified in *RowFormat*. Strings that start with *StartStringToDefineRowsToIgnore* are ignored.

A blank *RowDelimiter* or *RowElementDelimiter* is treated in the same way as a blank *Delimiter* in MnParseStringIntoStringArray.

ParseStringInto2dStringArray

[MnParseStringInto2dStringArray]

Returns a String array containing elements derived from *InputString*, each row being delimited by the chosen *RowDelimiter* and each element within the row by a *RowElementDelimiter*, with number of entries in each row defined by number of elements identified in *RowFormat*. Strings that start with *StartStringToDefineRowsToIgnore* are ignored.

A blank *RowDelimiter* or *RowElementDelimiter* is treated in the same way as a blank *Delimiter* in MnParseStringIntoStringArray.

ParseStringIntoBooleanArray

[MnParseStringIntoBooleanArray]

Returns a (Boolean) array containing elements derived from *InputString*, each element being delimited by the chosen *Delimiter*.

A blank *Delimiter* is treated in the same way as a blank *Delimiter* in MnParseStringIntoStringArray.

ParseStringIntoDateUKArray

[MnParseStringIntoDateUKArray]

Returns a (Date) array containing elements derived from *InputString*, each element being delimited by the chosen *Delimiter*.

A blank *Delimiter* is treated in the same way as a blank *Delimiter* in MnParseStringIntoStringArray.

ParseStringIntoDoubleArray

[MnParseStringIntoDoubleArray]

Returns a numerical (i.e. Double) array containing elements derived from *InputString*, each element being delimited by the chosen *Delimiter*.

A blank *Delimiter* is treated in the same way as a blank *Delimiter* in MnParseStringIntoStringArray.

ParseStringIntoIntegerArray

[MnParseStringIntoIntegerArray]

Returns an (Integer) array containing elements derived from *InputString*, each element being delimited by the chosen *Delimiter*.

A blank *Delimiter* is treated in the same way as a blank *Delimiter* in MnParseStringIntoStringArray.

ParseStringIntoStringArray

[MnParseStringIntoStringArray]

Returns a (String) array containing elements derived from *InputString*, each element being delimited by the chosen *Delimiter*.

If *Delimiter* is blank then it is assumed that *InputString* should be broken up as follows:

(a) If *InputString* contains carriage return (‘Cr’), line feed (‘Lf’) or tab (‘Tab’) control characters then the string is broken up into consecutive character strings between each occurrence of these control characters (except for CrLf and LfCr which are treated as if they are a single control character)
(b) If *InputString* doesn’t contain any of these three control characters then the Delimiter is defaulted to a comma, i.e. “,”, and the string is broken up between consecutive commas

Percent

[MnPercent]

Returns the value of x per cent (i.e. 0.01 times x)

Percentile

[MnPercentile]

Returns j_m, the m’th percentile (m here being between 0 and 1) of a series of observations x_i (for $i = 1, \dots, n$).

Different commentators use different ways of identifying percentiles (i.e. quantiles) from finite sized samples. The Nematrian website uses the following methodology (for MnPercentile but not for, say, MnWeightedPercentile).

First sort the series in ascending order. The 0'th percentile (i.e. $m = 0$) is the value ranked first in this ordering and the 100'th percentile (i.e. $m = 1$) is the value ranked last in this ordering. The percentile ranking given to other observations are equally spaced between the 0 and 1 depending on the position of the observation in this ordered list. Other percentiles are calculated by linear interpolation between the two values straddling the given percentile rank.

The equivalent function in Microsoft Excel is PERCENTILE (or PERCENTILE.INC).

For a similar function that returns the minimum of the x_i for the $1/(n+1)$'th percentile and the maximum for the $n/(n+1)$'th percentile, i.e. corresponds to Microsoft Excel's PERCENTILE.EXC function, then see MnPercentileExc.

PercentileExc

[MnPercentileExc]

Returns j_m, the m'th percentile (m here being between 0 and 1) of a series of observations x_i (for $i = 1, \dots, n$) treating the lowest value as corresponding to the $1/(n+1)$'th percentile and the highest as corresponding to the $n/(n+1)$'th percentile.

Different commentators use different ways of identifying percentiles (i.e. quantiles) from finite sized samples. The Nematrian website uses the following methodology (for *MnPercentileExc* but not for, say, MnWeightedPercentile).

First sort the series in ascending order. The $1/(n+1)$'th percentile (i.e. $m = 1/(n+1)$) is the value ranked first in this ordering and the 100'th percentile (i.e. $n/(n+1)$) is the value ranked last in this ordering. The percentile ranking given to other observations are equally spaced between within this range depending on the position of the observation in this ordered list. Other percentiles are calculated by linear interpolation between the two values straddling the given percentile rank.

The equivalent function in Microsoft Excel is PERCENTILE.EXC.

For a similar function that returns the minimum of the x_i for the 0'th percentile and the maximum for the 1'th percentile, i.e. corresponds to Microsoft Excel's PERCENTILE or PERCENTILE.INC function, then see MnPercentile.

PFDecrementTable

[MnPFDecrementTable]

Returns an array that contains a decrement table in a defined format relevant to (pension fund) projection purposes.

The pension fund members for this purpose are assumed to be subdivided between various different statuses (e.g. 'active', 'deferred, 'pensioner', 'beneficiary', 'left/dead' etc.) and also subdivided by age. The aim of the decrement table is to characterise the proportion of scheme members (or to be more precise the proportion of a given type of benefit attributable to these scheme members) moving between different categories as a function of age at outset and time from now to date at which their membership status is to be projected. See PFDecrementTableHelp for further details.

PFProjectBenefitsDepletedOutgoWoundUp
[MnPFProjectBenefitsDepletedOutgoWoundUp]

Returns a two-dimensional array showing the outgo through time expected from each aggregate membership category conditional on scheme winding up at different points in time (with the sponsor not necessarily making good any shortfall on wind up).

The impact of setting *FactorDefaultAdjParam* or *WindUpDefaultAdjParam* to non-zero values is explained further in MnPFProjectBenefitsProbDefault.

PFProjectBenefitsOutgoWoundUp
[MnPFProjectBenefitsOutgoWoundUp]

Returns a two-dimensional array showing the outgo through time expected from each aggregate membership category conditional on scheme winding up at different points in time (assuming sponsor makes good any shortfall on wind up).

The impact of setting *FactorDefaultAdjParam* or *WindUpDefaultAdjParam* to non-zero values is explained further in MnPFProjectBenefitsProbDefault.

PFProjectBenefitsProbDefault
[MnPFProjectBenefitsProbDefault]

Returns an array showing the probability of sponsor default over any given year in a pension fund projection.

If *FactorDefaultAdjParam* and *WindUpDefaultAdjParam* are both zero then the computations are very straightforward, depending only on *BaseSponsorDefaultRates*.

However, if *FactorDefaultAdjParam* or *WindUpDefaultAdjParam* are non-zero then the probability of default is adjusted from the above in a manner that reflecting the progression of the *FactorDrivingDefault* or by how well funded on a discontinuance basis the scheme is projected to be at future points in time, by applying a scaling factor to *BaseSponsorDefaultRates* (min 0, max 1) defined as follows:

$$DefaultScalingFactor = 1 + F \times \left(r(i) - e(i)\right) + W \times \frac{V(i)}{A(i)}$$

where: i = year in projection, F = *FactorDefaultAdjParam*, $r(i)$ = Actual return (movement) on Factor Driving Default(i), $e(i)$ = Expected return on Factor Driving Default(i), W = *WindUpDefaultAdjParam*, $V(i)$ = TotalBenefitValue(i) (on a windup basis), $A(i)$ = Total Asset Value(i)

(values of $r(i)$, $e(i)$, $V(i)$ and $A(i)$ are computed using Nematrian's approximate pension projection algorithms)

PFProjectBenefitsPVSingleRun
[MnPFProjectBenefitsPVSingleRun]

Returns, for an illustrative pension scheme with benefit and other characteristics approximated by the input parameters, an array containing (for each aggregate membership category):

(1) The present value of members' future benefits allowing for the possibility of the scheme sponsor will default in the future, but assuming that when the sponsor defaults any shortfall in the cost of providing these benefits from elsewhere is met (e.g. by the sponsor's bankruptcy estate)

(2) The present value of members' future benefits allowing for the possibility of the scheme sponsor will default in the future, and allowing for the possibility that when the sponsor defaults there may be a shortfall in the assets available to meet the cost of providing these benefits from elsewhere (e.g. because there are insufficient assets available in the sponsor's bankruptcy estate)

(3) The equivalent implied spread on the benefits attributable to that category of members (i.e. the addition to the yield curve needed to result in a value as per (2) given cash flows as per (1)

(4) The present value of members' future benefits ignoring the possibility of the scheme sponsor defaulting in the future (which differs from (1) because some members' benefits, e.g. for active members, may uprate more rapidly as long as the sponsor has not defaulted / scheme has not been wound up)

The impact of setting *FactorDefaultAdjParam* or *WindUpDefaultAdjParam* to non-zero values is explained further in MnPFProjectBenefitsProbDefault.

PgpmFundPercentileRank

[MnPgpmFundPercentileRank]

Returns the percentile rank of a fund in a specified (pre-loaded) performance measurement dataset, i.e. returns 0 if the fund is the top ranked fund over the given period, 1 if it is the bottom ranked fund, with other rankings equally spaced between these two.

Note: The input period start/ends need to be consecutive, e.g. if there are two periods with the first one from 1 Jan 2000 to 31 Dec 2000 and the second from 1 Jan 2001 to 31 Dec 2001 then these should be passed as an array with three entries, 31 Dec 1999, 31 Dec 2000 and 31 Dec 2001 in that order. The sizes/format of the arrays passed as parameters for this web function as specified below assume that the number of Funds is m and the number of consecutive periods is n. Return data should be passed with the consecutive returns for the same fund in a single block, i.e. all of the first fund's returns, followed by all of the second fund's returns etc. If a fund does not have a return for a given period (e.g. the fund did not exist for the whole of the period) then use for that fund for that period the *ReturnInvalidCode*. When calculating per group level statistics, e.g. percentiles and median, funds that did not have a return for a given period are excluded.

PgpmFundRank

[MnPgpmFundRank]

Returns the rank of a fund in a specified (pre-loaded) performance measurement dataset, i.e. an integer between 1 and *n*, where there are *n* funds in the peer group with a return over the given period, the fund with the highest return being given a rank of 1, the fund with the next highest a rank of 2 etc.

Note: The input period start/ends need to be consecutive, e.g. if there are two periods with the first one from 1 Jan 2000 to 31 Dec 2000 and the second from 1 Jan 2001 to 31 Dec 2001 then these should be passed as an array with three entries, 31 Dec 1999, 31 Dec 2000 and 31 Dec 2001 in that order. The sizes/format of the arrays passed as parameters for this web function as specified below assume that the number of Funds is m and the number of consecutive periods is n. Return data should be passed with the consecutive returns for the same fund in a single block, i.e. all of the first fund's returns, followed by all of the second fund's returns etc. If a fund does not have a return for a given period (e.g. the fund did not exist for the whole of the period) then use for that fund for that period the *ReturnInvalidCode*. When calculating per group level statistics, e.g. percentiles and median, funds that did not have a return for a given period are excluded.

PgpmFundRet

[MnPgpmFundRet]

Returns the return of a specified fund in a specified (pre-loaded) peer group performance dataset for the specified period.

Note: The input period start/ends need to be consecutive, e.g. if there are two periods with the first one from 1 Jan 2000 to 31 Dec 2000 and the second from 1 Jan 2001 to 31 Dec 2001 then these should be passed as an array with three entries, 31 Dec 1999, 31 Dec 2000 and 31 Dec 2001 in that order. The sizes/format of the arrays passed as parameters for this web function as specified below assume that the number of Funds is m and the number of consecutive periods is n. Return data should be passed with the consecutive returns for the same fund in a single block, i.e. all of the first fund's returns, followed by all of the second fund's returns etc. If a fund does not have a return for a given period (e.g. the fund did not exist for the whole of the period) then use for that fund for that period the *ReturnInvalidCode*. When calculating per group level statistics, e.g. percentiles and median, funds that did not have a return for a given period are excluded.

PgpmFundRetExists

[MnPgpmFundRetExists]

Returns TRUE if it is possible to identify the return on a specified fund in a specified (pre-loaded) peer group performance dataset for the specified period, otherwise returns FALSE.

Note: The input period start/ends need to be consecutive, e.g. if there are two periods with the first one from 1 Jan 2000 to 31 Dec 2000 and the second from 1 Jan 2001 to 31 Dec 2001 then these should be passed as an array with three entries, 31 Dec 1999, 31 Dec 2000 and 31 Dec 2001 in that order. The sizes/format of the arrays passed as parameters for this web function as specified below assume that the number of Funds is m and the number of consecutive periods is n. Return data should be passed with the consecutive returns for the same fund in a single block, i.e. all of the first fund's returns, followed by all of the second fund's returns etc. If a fund does not have a return for a given period (e.g. the fund did not exist for the whole of the period) then use for that fund for that period the *ReturnInvalidCode*. When calculating per group level statistics, e.g. percentiles and median, funds that did not have a return for a given period are excluded.

PgpmFundRets

[MnPgpmFundRets]

Returns an array containing the returns of funds in a specified (pre-loaded) peer group performance dataset for the specified period.

Note: The input period start/ends need to be consecutive, e.g. if there are two periods with the first one from 1 Jan 2000 to 31 Dec 2000 and the second from 1 Jan 2001 to 31 Dec 2001 then these should be passed as an array with three entries, 31 Dec 1999, 31 Dec 2000 and 31 Dec 2001 in that order. The sizes/format of the arrays passed as parameters for this web function as specified below assume that the number of Funds is m and the number of consecutive periods is n. Return data should be passed with the consecutive returns for the same fund in a single block, i.e. all of the first fund's returns, followed by all of the second fund's returns etc. If a fund does not have a return for a given period (e.g. the fund did not exist for the whole of the period) then use for that fund for that period the *ReturnInvalidCode*. When calculating per group level statistics, e.g. percentiles and median, funds that did not have a return for a given period are excluded.

PgpmFundRetsExist

[MnPgpmFundRetsExist]

Returns TRUE if it is possible to identify an array representing the returns on the different funds in a specified (pre-loaded) peer group performance dataset for the specified period, otherwise returns FALSE.

Note: The input period start/ends need to be consecutive, e.g. if there are two periods with the first one from 1 Jan 2000 to 31 Dec 2000 and the second from 1 Jan 2001 to 31 Dec 2001 then these should be passed as an array with three entries, 31 Dec 1999, 31 Dec 2000 and 31 Dec 2001 in that order. The sizes/format of the arrays passed as parameters for this web function as specified below assume that the number of Funds is m and the number of consecutive periods is n. Return data should be passed with the consecutive returns for the same fund in a single block, i.e. all of the first fund's returns, followed by all of the second fund's returns etc. If a fund does not have a return for a given period (e.g. the fund did not exist for the whole of the period) then use for that fund for that period the *ReturnInvalidCode*. When calculating per group level statistics, e.g. percentiles and median, funds that did not have a return for a given period are excluded.

PgpmFundVolatilityPopulation

[MnPgpmFundVolatilityPopulation]

Returns the volatility (population standard deviation) of returns of a specified fund in a specified (pre-loaded) peer group performance dataset.

Two variants are provided, one calculating the tracking error using sample standard deviations (MnPgpmFundVolatilitySample) and one using population standard deviations (this one).

Note: The input period start/ends need to be consecutive, e.g. if there are two periods with the first one from 1 Jan 2000 to 31 Dec 2000 and the second from 1 Jan 2001 to 31 Dec 2001 then these should be passed as an array with three entries, 31 Dec 1999, 31 Dec 2000 and 31 Dec 2001 in that order. The sizes/format of the arrays passed as parameters for this web function as specified below assume that the number of Funds is m and the number of consecutive periods is n. Return data should be passed with the consecutive returns for the same fund in a single block, i.e. all of the first fund's returns, followed by all of the second fund's returns etc. If a fund does not have a return for a given period (e.g. the fund did not exist for the whole of the period) then use for that fund for that period the *ReturnInvalidCode*. When calculating per group level statistics, e.g. percentiles and median, funds that did not have a return for a given period are excluded.

PgpmFundVolatilitySample

[MnPgpmFundVolatilitySample]

Returns the volatility (population standard deviation) of returns of a specified fund in a specified (pre-loaded) peer group performance dataset.

Two variants are provided, one using sample standard deviations (this one) and one using population standard deviations (MnPgpmFundVolatilityPopulation).

Note: The input period start/ends need to be consecutive, e.g. if there are two periods with the first one from 1 Jan 2000 to 31 Dec 2000 and the second from 1 Jan 2001 to 31 Dec 2001 then these should be passed as an array with three entries, 31 Dec 1999, 31 Dec 2000 and 31 Dec 2001 in that order. The sizes/format of the arrays passed as parameters for this web function as specified below assume that the number of Funds is m and the number of consecutive periods is n. Return data should be passed with the consecutive returns for the same fund in a single block, i.e. all of the first fund's returns, followed by all of the second fund's returns etc. If a fund does not have a return for a given period (e.g. the fund did not exist for the whole of the period) then use for that fund for that period the *ReturnInvalidCode*. When calculating per group level statistics, e.g. percentiles and median, funds that did not have a return for a given period are excluded.

PgpmMedianRet

[MnPgpmMedianRet]

Returns the median return of a specified (pre-loaded) peer group performance dataset. A special case of MnPgpmPercentileRet.

Note: The input period start/ends need to be consecutive, e.g. if there are two periods with the first one from 1 Jan 2000 to 31 Dec 2000 and the second from 1 Jan 2001 to 31 Dec 2001 then these should be passed as an array with three entries, 31 Dec 1999, 31 Dec 2000 and 31 Dec 2001 in that order. The sizes/format of the arrays passed as parameters for this web function as specified below assume that the number of Funds is m and the number of consecutive periods is n. Return data should be passed with the consecutive returns for the same fund in a single block, i.e. all of the first fund's returns, followed by all of the second fund's returns etc. If a fund does not have a return for a given period (e.g. the fund did not exist for the whole of the period) then use for that fund for that period the *ReturnInvalidCode*. When calculating per group level statistics, e.g. percentiles and median, funds that did not have a return for a given period are excluded.

PgpmPercentileRet

[MnPgpmPercentileRet]

Returns the return corresponding to a given percentile (quantile) in a specified (pre-loaded) peer group performance dataset, i.e. the volatility of relative returns.

Note: The input period start/ends need to be consecutive, e.g. if there are two periods with the first one from 1 Jan 2000 to 31 Dec 2000 and the second from 1 Jan 2001 to 31 Dec 2001 then these should be passed as an array with three entries, 31 Dec 1999, 31 Dec 2000 and 31 Dec 2001 in that order. The sizes/format of the arrays passed as parameters for this web function as specified below assume that the number of Funds is m and the number of consecutive periods is n. Return data should be passed with the consecutive returns for the same fund in a single block, i.e. all of the first fund's returns, followed by all of the second fund's returns etc. If a fund does not have a return for a given period (e.g. the fund did not exist for the whole of the period) then use for that fund for that period the *ReturnInvalidCode*. When calculating per group level statistics, e.g. percentiles and median, funds that did not have a return for a given period are excluded.

PgpmRelativeRet

[MnPgpmRelativeRet]

Returns the relative return (relative to the median) of a specified fund in a specified (pre-loaded) peer group performance dataset. Relative returns may be calculated arithmetically, geometrically or logarithmically, see RelativeReturnComputations.

Note: The input period start/ends need to be consecutive, e.g. if there are two periods with the first one from 1 Jan 2000 to 31 Dec 2000 and the second from 1 Jan 2001 to 31 Dec 2001 then these should be passed as an array with three entries, 31 Dec 1999, 31 Dec 2000 and 31 Dec 2001 in that order. The sizes/format of the arrays passed as parameters for this web function as specified below assume that the number of Funds is m and the number of consecutive periods is n. Return data should be passed with the consecutive returns for the same fund in a single block, i.e. all of the first fund's returns, followed by all of the second fund's returns etc. If a fund does not have a return for a given period (e.g. the fund did not exist for the whole of the period) then use for that fund for that period the *ReturnInvalidCode*. When calculating per group level statistics, e.g. percentiles and median, funds that did not have a return for a given period are excluded.

PgpmTrackingErrorPopulation

[MnPgpmTrackingErrorPopulation]

Returns the tracking error (relative to the median) of a specified fund in a specified (pre-loaded) peer group performance dataset, i.e. the volatility of relative returns. Relative returns may be calculated arithmetically, geometrically or logarithmically, see RelativeReturnComputations.

Two variants are provided, one calculating the tracking error using sample standard deviations (MnPgpmTrackingErrorSample) and one using population standard deviations (this one).

Note: The input period start/ends need to be consecutive, e.g. if there are two periods with the first one from 1 Jan 2000 to 31 Dec 2000 and the second from 1 Jan 2001 to 31 Dec 2001 then these should be passed as an array with three entries, 31 Dec 1999, 31 Dec 2000 and 31 Dec 2001 in that order. The sizes/format of the arrays passed as parameters for this web function as specified below assume that the number of Funds is m and the number of consecutive periods is n. Return data should be passed with the consecutive returns for the same fund in a single block, i.e. all of the first fund's returns, followed by all of the second fund's returns etc. If a fund does not have a return for a given period (e.g. the fund did not exist for the whole of the period) then use for that fund for that period the *ReturnInvalidCode*. When calculating per group level statistics, e.g. percentiles and median, funds that did not have a return for a given period are excluded.

PgpmTrackingErrorSample

[MnPgpmTrackingErrorSample]

Returns the tracking error (relative to the median) of a specified fund in a specified (pre-loaded) peer group performance dataset, i.e. the volatility of relative returns. Relative returns may be calculated arithmetically, geometrically or logarithmically, see RelativeReturnComputations.

Two variants are provided, one using sample standard deviations (this one) and one using population standard deviations (MnPgpmTrackingErrorPopulation).

Note: The input period start/ends need to be consecutive, e.g. if there are two periods with the first one from 1 Jan 2000 to 31 Dec 2000 and the second from 1 Jan 2001 to 31 Dec 2001 then these should be passed as an array with three entries, 31 Dec 1999, 31 Dec 2000 and 31 Dec 2001 in that order. The sizes/format of the arrays passed as parameters for this web function as specified below assume that the number of Funds is m and the number of consecutive periods is n. Return data should be passed with the consecutive returns for the same fund in a single block, i.e. all of the first fund's returns, followed by all of the second fund's returns etc. If a fund does not have a return for a given period (e.g. the fund did not exist for the whole of the period) then use for that fund for that period the *ReturnInvalidCode*. When calculating per group level statistics, e.g. percentiles and median, funds that did not have a return for a given period are excluded.

Pi

[MnPi]

Returns the ratio between the circumference of a circle and its diameter, i.e. π

MnPlotExpression

[MnPlotExpression]

Returns a plot of a function specified as per the *Expression* over a range of values from *xMinimum* to *xMaximum*. More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information.

The expression is evaluated using Nematrian's usual expression evaluation conventions, see e.g. MnEvaluate, MnEvaluateUsing or Expression Evaluation.

For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

The number of units debited from a *SessionId* to carry out this function is driven primarily by the complexity of the expression involved and the number of times other Nematrian functions are called in the plotting.

MnPlotExpressions

[MnPlotExpressions]

Returns a plot of a set of functions specified as per the elements of *Expressions* over a range of values from *xMinimum* to *xMaximum*. More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information.

The expressions are evaluated using Nematrian's usual expression evaluation conventions, see e.g. MnEvaluate, MnEvaluateUsing or Expression Evaluation.

For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

The number of units debited from a *SessionId* to carry out this function is driven primarily by the complexity of the expressions involved and the number of times other Nematrian functions are called in the plotting.

PlotQuadraticEfficientFrontier

[MnPlotQuadraticEfficientFrontier]

Returns a plot of a (constrained quadratic) efficient frontier. More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information.

The efficient frontier is a plot of return versus risk for portfolios which optimise the following utility (for a variety of risk-reward trade-off factors, λ), where x is a vector of portfolio weights, r is a vector of assumed returns on each asset and V is the covariance matrix ($= s^T Cs$, where s is the vector of risks on each asset class, here assumed to be characterised by their volatilities, as this approach is merely a mean-variance one, and C their correlation matrix):

$$U(x) = r.x - \lambda(x-b)^T V(x-b)$$

Inputs are as per MnConstrainedQuadraticPortfolioOptimiser (which is the web function used to calculate the underlying risk/return points forming the efficient frontier) except that:

(a) An array of λ's is supplied rather than a single value

(b) The user can select a Title for the chart (using *ChartTitle*)

(c) An additional *ChartFormat* parameter is available to specify how the chart should be formatted. For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

PlotQuadraticEfficientPortfolios

[MnPlotQuadraticEfficientPortfolios]

Returns a plot of the asset mixes underlying a (constrained quadratic) efficient frontier. More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information.

These asset mixes are the portfolios that optimise the following utility (for a variety of risk-reward trade-off factors, λ), where x is a vector of portfolio weights, r is a vector of assumed returns on each asset and V is the covariance matrix ($= s^T Cs$, where s is the vector of risks on each asset class, here assumed to be characterised by their volatilities, as this approach is merely a mean-variance one, and C their correlation matrix):

$$U(x) = r.x - \lambda(x-b)^T V(x-b)$$

Inputs are as per MnConstrainedQuadraticPortfolioOptimiser (which is the web function used to calculate the underlying portfolios contributing to the efficient frontier) except that:

(a) An array of λ's is supplied rather than a single value

(d) The user can select a Title for the chart (using *ChartTitle*)

(e) An additional *ChartFormat* parameter is available to specify how the chart should be formatted. For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

PlotStandardisedQQ

[MnPlotStandardisedQQ]

Returns a standardised Quantile-Quantile ('QQ') plot. More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information.

A standardised QQ-plot for a series of observations produces a plot of the observed quantile value versus the expected quantile value, with the series 'standardised' to have zero mean and unit standard deviation using a formula along the lines of:

$$y_i = \frac{x_i - \bar{x}}{s}$$

where s is the (sample) standard deviation of the data series x_i.

To allow users to plot several different series together, this function takes as input two different arrays, one containing all of the input data for all of the series (i.e. *InputData*), positioned consecutively and the other containing the number of entries in each series (i.e. *InputDataSizes*). So if *InputDataSizes* is {3,4} then two series are being analysed simultaneously, the first with three data points and the second with four, so *InputData* should have seven entries, the first three of which are for the first series and the last four for the second series.

For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

PlotStandardisedQQWithComparisons

[MnPlotStandardisedQQWithComparisons]

Returns a standardised Quantile-Quantile ('QQ') plot with some extra comparisons. More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information.

A standardised QQ-plot for a series of observations produces a plot of the observed quantile value versus the expected quantile value, with the series 'standardised' to have zero mean and unit standard deviation using a formula along the lines of:

$$y_i = \frac{x_i - \bar{x}}{s}$$

where s is the (sample) standard deviation of the data series x_i.

The extra comparisons available are:

(a) The corresponding Cornish Fisher (4th moment) asymptotic expansion, which in effect makes use of just the skew and kurtosis to estimate the shape of the QQ plot (set *IncludeCF4* as true if you want these comparisons); and

(b) The corresponding standard weighted cubic curve fit (set *IncludeWCCF* as true if you want these comparisons).

To allow users to plot several different series together, this function takes as input two different arrays, one containing all of the input data for all of the series (i.e. *InputData*), positioned consecutively and the other containing the number of entries in each series (i.e. *InputDataSizes*). So if *InputDataSizes* is {3,4} then two series are being analysed simultaneously, the first with three data points and the second with four, so *InputData* should have seven entries, the first three of which are for the first series and the last four for the second series.

For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

PlotStandardisedUpwards1dQQ

[MnPlotStandardisedUpwards1dQQ]

Returns a standardised 'upwards' (1-dimensional) Quantile-Quantile ('QQ') plot. More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information.

A standardised QQ-plot for a series of observations produces a plot of the observed quantile value versus the expected quantile value, with the series 'standardised' to have zero mean and unit standard deviation using a formula along the lines of:

$$y_i = \frac{x_i - \bar{x}}{s}$$

where s is the (sample) standard deviation of the data series x_i.

A standardised 'upwards' QQ-plot is one where the bottom half of this plot is flipped across the *x*-axis. It is particularly useful as a precursor to a standardised 2-dimensional plot, which provides a compact visual representation of how two series may jointly diverge from normality in their extremities.

To allow users to plot several different series together, this function takes as input two different arrays, one containing all of the input data for all of the series (i.e. *InputData*), positioned consecutively and the other containing the number of entries in each series (i.e. *InputDataSizes*). So if *InputDataSizes* is {3,4} then two series are being analysed simultaneously, the first with three data points and the second with four, so *InputData* should have seven entries, the first three of which are for the first series and the last four for the second series.

For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

PlotWeightedStandardisedQQ

[MnPlotWeightedStandardisedQQ]

Returns a weighted standardised Quantile-Quantile (‘QQ’) plot. More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information.

A weighted standardised QQ-plot for a series of observations produces a plot of the observed quantile value versus the expected quantile value, with the series ‘standardised’ to have zero (weighted) mean and unit (weighted) standard deviation using a formula along the lines of:

$$y_i = \frac{x_i - \tilde{x}}{\tilde{s}}$$

where s is the (sample) standard deviation of the data series x_i.

The observed (weighted) quantile also takes account of the weights given to the different data points.

To allow users to plot several different series together, this function takes as input two different arrays, one containing all of the input data for all of the series (i.e. *InputData*), positioned consecutively and the other containing the number of entries in each series (i.e. *InputDataSizes*). So if *InputDataSizes* is {3,4} then two series are being analysed simultaneously, the first with three data points and the second with four, so *InputData* should have seven entries, the first three of which are for the first series and the last four for the second series.

For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

PlotXdateYLineChart

[MnPlotXdateYLineChart]

Returns a *xy* line chart where the *x* are dates and the *y* are numeric. More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information.

To allow users to plot several different series together, this function takes as input two different arrays, one containing all of the input data for all of the series (i.e. *InputData*), positioned consecutively and the other containing the number of entries in each series (i.e. *InputDataSizes*). So if *InputDataSizes* is {3,4} then two series are being analysed simultaneously, the first with three data points and the second with four, so *InputData* should have seven entries, the first three of which are for the first series and the last four for the second series.

For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

PlotXYExpressionGeneral

[MnPlotXYExpressionGeneral]

Returns a plot of a set of functions specified as per the elements of *Expressions*. More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information. A generalisation of MnPlotExpression and MnPlotExpressions (and MnPlotExpression).

The expression is evaluated using Nematrian's usual expression evaluation conventions, see e.g. MnEvaluate, MnEvaluateUsing or Expression Evaluation.

For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

The number of units debited from a *SessionId* to carry out this function is driven primarily by the complexity of the expression involved and the number of times other Nematrian functions are called in the plotting.

PlotXYLineChart

[MnPlotXYLineChart]

Returns a *xy* line chart where the *x* and *y* are both numeric. More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information.

To allow users to plot several different series together, this function takes as input two different arrays, one containing all of the input data for all of the series (i.e. *InputData*), positioned consecutively and the other containing the number of entries in each series (i.e. *InputDataSizes*). So if *InputDataSizes* is {3,4} then two series are being analysed simultaneously, the first with three data points and the second with four, so *InputData* should have seven entries, the first three of which are for the first series and the last four for the second series.

For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

PoissonCdf

[MnPoissonCdf]

Returns the Poisson distribution cumulative distribution function. Included for compatibility with the POISSON and POISSON.DIST functions (with cumulative parameter=true) in Microsoft Excel and some other spreadsheet packages.

Equivalent to calling MnProbDistCdf using as the distribution parameter "poisson".

PoissonPmf

[MnPoissonPmf]

Returns the Poisson distribution probability mass function. Included for compatibility with the POISSON and POISSON.DIST functions (with cumulative parameter = false) in Microsoft Excel and some other spreadsheet packages.

Equivalent to calling MnProbDistPmf using as the distribution parameter "poisson".

PolarToCartesian

[MnPolarToCartesian]

Returns an array (x, y) which in Cartesian coordinates corresponds to the point which has polar coordinates as defined by the input array (r, θ), i.e. with $x = r\cos\theta$, $y = r\sin\theta$

PolarToCartesian2

[MnPolarToCartesian2]

Returns an array (x, y) which in Cartesian coordinates corresponds to the point which has polar coordinates, r and θ, i.e. $x = r\cos\theta$, $y = r\sin\theta$)

PolyInterp

[MnPolyInterp]

Returns the result of carrying out a polynomial interpolation given input arrays x and y and a value at which to interpolate, i.e. $xToInterpolateFrom$. The x and y need to be of the same size (say n). We find the (unique) polynomial $f(x)$ or order $n-1$ that fits the points $y_1 = f(x_1), \dots, y_n = f(x_n)$ and then return $f(xToInterpolateFrom)$.

PopulationCovariance

[MnPopulationCovariance]

Returns the population covariance $V_{1,2}$ between two series, $x_{1,t}$ and $x_{2,t}$.

The population covariance is calculated as:

$$V_{1,2} = \frac{1}{n}\sum_{t=1}^{n}(x_{1,t} - \bar{x}_1)(x_{2,t} - \bar{x}_2)$$

where $\bar{x}_i$ is the mean of the i-th series etc.

PopulationCovariances

[MnPopulationCovariances]

Returns an array of m^2 elements corresponding to the elements of the matrix, V, of the (sample) covariances for m series that are provided to the function as an array of mn elements, where the first n elements correspond to the terms in the first series, the next n elements to the corresponding terms in the second series etc.

Population covariances are calculated as:

$$V_{i,j} = \frac{1}{n}\sum_{t=1}^{n}(x_{i,t} - \bar{x}_i)(x_{j,t} - \bar{x}_j)$$

where $\bar{x}_i$ is the mean of the i-th series etc.

PopulationKurt

[MnPopulationKurt]

Returns the population kurtosis, γ_2 (or more precisely the 'excess' kurtosis) of a series x_i (for $i = 1, \dots, n$), calculated as follows:

$$\gamma_2 = \frac{1}{n}\sum\left(\frac{x_i - \bar{x}}{s_p}\right)^4 - 3$$

Here, $\bar{x}$ is the mean and s_p is the population standard deviation of the series

There is no immediately equivalent function in Microsoft Excel (the KURT command is equivalent to MnKurt).

PopulationSkew

[MnPopulationSkew]

Returns the population skew, $\gamma_{1,p}$ (also sometimes called 'skewness') of a series x_i (for $i = 1, \dots, n$), calculated as follows:

$$\gamma_{1,p} = \frac{1}{n}\sum\left(\frac{x_i - \bar{x}}{s_p}\right)^3$$

Here, $\bar{x}$ is the mean and s_p is the population standard deviation of the series

There is no immediately equivalent function in Microsoft Excel (the SKEW command is equivalent to MnSkew).

PopulationStdev

[MnPopulationStdev]

Returns the population standard deviation s_p of a series x_i (for $i = 1, \dots, n$), calculated as follows:

$$s_p = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

(where $\bar{x}$ is the mean of the series)

The equivalent function in Microsoft Excel is STDEVP.

PopulationVariance

[MnPopulationVariance]

Returns the population variance v_p of a series x_i (for $i = 1, \dots, n$), calculated as follows:

$$v_p = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

(where $\bar{x}$ is the mean of the series)

The equivalent function in Microsoft Excel is VARP.

Power

[MnPower]

Returns x to the power y, i.e. x^y

PrincipalComponents

[MnPrincipalComponents]

Returns an array of $m \times n$ entries corresponding to the (zero mean) series corresponding to the *m* principal components of *m* different data series each of *n* observations, ordered so that the most important principal components are first. The first *n* entries of the array returns correspond to the elements of the largest principal component etc.

See here for further details of how principal components are derived and on refinements that potentially include a greater focus on 'meaning' in their computation.

PrincipalComponentsSizes

[MnPrincipalComponentsSizes]

Returns an array corresponding to the sizes of the *m* principal components of *m* different data series of *n* observations, ordered so that the sizes of the most important principal components are first.

See here for further details of how principal components are derived and on refinements that potentially include a greater focus on 'meaning' in their computation.

PrincipalComponentsWeights
[MnPrincipalComponentsWeights]

Returns an array of m^2 elements corresponding to the weight coefficients that need to be applied to (zero mean equivalents of) m original series to reproduce the m principal components of corresponding to these series, ordered so that the coefficients (i.e. weights in the original series) for the most important principal component come first. So the first entry in the output is the weight of the first series in the first (i.e. most important) principal component, the second entry in the output is the weight of the second series in the same (i.e. most important) principal component etc.

See here for further details of how principal components are derived and on refinements that potentially include a greater focus on 'meaning' in their computation.

ProbDistCdf
[MnProbDistCdf]

Returns the cumulative distribution function for a given x for a probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames), i.e. $Pr(X \leq x)$.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistCtsBoundedLower
[MnProbDistCtsBoundedLower]

Returns true if the (continuous) probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames) has a lower bound other than $-\infty$ otherwise returns false.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistCtsBoundedUpper
[MnProbDistCtsBoundedUpper]

Returns true if the (continuous) probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames) has an upper bound other than $+\infty$ otherwise returns false.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistCtsLowerBound
[MnProbDistCtsLowerBound]

If the (continuous) probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames) has a lower bound other than $-\infty$ then returns this lower bound.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistCtsUpperBound

[MnProbDistCtsUpperBound]

If the (continuous) probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames) has an upper bound other than $+\infty$ then returns this upper bound.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistHazardFunction

[MnProbDistHazardFunction]

Returns the hazard function of the probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames), i.e.:

$$h(x) = \frac{f(x)}{1 - F(x)}$$

where $f(x)$ is the probability density function and $F(x)$ is the cumulative distribution function of the relevant distribution.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistIsCts

[MnProbDistIsCts]

Returns true if the probability distribution given by *DistributionName* is continuous (so has a probability density function) and false if it is discrete (so has a probability mass function).

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistKurt

[MnProbDistKurt]

Returns the (excess) kurtosis of a given probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames).

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistLikelihood
[MnProbDistLikelihood]

Returns the likelihood of a given dataset for a probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames).

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistLogLikelihood
[MnProbDistLogLikelihood]

Returns the log likelihood of a given dataset for a probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames).

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistMean
[MnProbDistMean]

Returns the mean of a probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames).

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistMLE
[MnProbDistMLE]

Returns the maximum likelihood estimation vector for a given dataset for parameters of a probability distribution given by *DistributionName* and with the order of parameter values matching the parameter names listed by MnProbDistParamNames).

The algorithm used starts from values for each parameter in line with its value in *OpeningParameterChoices* and searches for the (possibly just 'nearest') parameter selection that maximises the weighted (log) likelihood function.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistMLERestricted

[MnProbDistMLERestricted]

Returns the maximum likelihood estimation vector for a given dataset for parameters of a probability distribution given by *DistributionName* and with the order of parameter values matching the parameter names listed by MnProbDistParamNames) subject to the restriction that for those parameters with *AllowParameterToVary* = FALSE the estimator must be identical to its *OpeningParameterChoice*.

The algorithm used starts from values for each of the non-fixed parameters in line with its value in *OpeningParameterChoices* and searches for the (possibly just 'nearest') parameter selection that maximises the weighted (log) likelihood function.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistPage

[MnProbDistPage]

Returns the main Nematrian web page that provides details of the probability distribution given by *DistributionName*. See also here for further details of shift and/or scale variants of distributions.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistParamCount

[MnProbDistParamCount]

Returns the number of the parameters used in the specification of a probability distribution given by *DistributionName*.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistParamNames

[MnProbDistParamNames]

Returns the names of the parameters used in the specification of the probability distribution given by *DistributionName*.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistPdf

[MnProbDistPdf]

Returns the probability density function for a given x for a (continuous) probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames). To identify whether a given distribution is continuous (so has a probability density function) or discrete (so has a probability mass function) see MnProbDistIsCts.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistPlotCdf

[MnProbDistPlotCdf]

Returns a plot of the cumulative distribution function of the probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames). More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

ProbDistPlotCdfs

[MnProbDistPlotCdfs]

Returns a plot of the cumulative distribution functions of several probability distributions simultaneously, with the distribution names given by *DistributionNames* and with parameter values (for each distribution) given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames). The first set of elements of *ParamValues* refer to the first distribution, the next set to the second distribution etc. If the distributions being analysed have different numbers of parameters then you should work out the largest number of parameters any of them has and pad out the others so that all sets of elements of *ParamValues* have the same size. More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

ProbDistPlotPdf

[MnProbDistPlotPdf]

Returns a plot of the probability density function of the probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames). More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information.

To identify whether a given distribution is continuous (so has a probability density function) or discrete (so has a probability mass function) see MnProbDistIsCts. Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

ProbDistPlotPdfs

[MnProbDistPlotPdfs]

Returns a plot of the probability density functions of several probability distributions simultaneously, with the distribution names given by *DistributionNames* and with parameter values (for each distribution) given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames). The first set of elements of *ParamValues* refer to the first distribution, the next set to the second distribution etc. If the distributions being analysed have different numbers of parameters then you should work out the largest number of parameters any of them has and pad out the others so that all sets of elements of *ParamValues* have the same size. More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information.

To identify whether a given distribution is continuous (so has a probability density function) or discrete (so has a probability mass function) see MnProbDistIsCts. Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

ProbDistPlotPmf

[MnProbDistPlotPmf]

Returns a plot of the probability mass function of the probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames). More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information.

To identify whether a given distribution is continuous (so has a probability density function) or discrete (so has a probability mass function) see MnProbDistIsCts. Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

ProbDistPlotPmfs

[MnProbDistPlotPmfs]

Returns a plot of the probability mass functions of several probability distributions simultaneously, with the distribution names given by *DistributionNames* and with parameter values (for each distribution) given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames). The first set of elements of *ParamValues* refer to the first distribution, the next set to the second distribution etc. If the distributions being analysed have different numbers of parameters then you should work out the largest number of parameters any of them has and pad out the others so that all sets of elements of *ParamValues* have the same size. More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information.

To identify whether a given distribution is continuous (so has a probability density function) or discrete (so has a probability mass function) see MnProbDistIsCts. Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

ProbDistPlotQQ

[MnProbDistPlotQQ]

Returns a quantile-quantile plot (versus the normal distribution) of the quantile function of the probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames). More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information.

To make it easier to see how close to a normal distribution the given distribution appears we also show a ‘standardised’ expected QQ-plot for an equivalent normal distribution. As not all distributions have a standard deviation we standardise the expected QQ-plot so that it goes through the lower quartile and upper quartile of the selected distribution.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

ProbDistPlotQQs

[MnProbDistPlotQQs]

Returns a quantile-quantile plot (versus the standard normal distribution) of the quantile functions of several probability distributions simultaneously, with the distribution names given in

DistributionNames and with parameter values (for each distribution) given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames). The first set of elements of *ParamValues* refer to the first distribution, the next set to the second distribution etc. If the distributions being analysed have different numbers of parameters then you should work out the largest number of parameters any of them has and pad out the others so that all sets of elements of *ParamValues* have the same size. More precisely, returns a string which is the (temporary) code for a SmartChart that plots such information.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

For further details on how to use the *ChartFormat* parameter to format a chart see using the ChartFormat parameter.

ProbDistPmf

[MnProbDistPmf]

Returns the probability mass function for a given x for a (discrete) probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames). To identify whether a given distribution is continuous (so has a probability density function) or discrete (so has a probability mass function) see MnProbDistIsCts.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistPolyEval

[MnProbDistPolyEval]

Calculates the expected value of a polynomial $a_0x^0 + a_1x^1 + \cdots$ with coefficients as given in *PolyCoeffs* (i.e. $a_0, a_1, \ldots$) between specified lower and upper limits which can be infinite, for a probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames).

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistQuantile

[MnProbDistQuantile]

Returns the inverse cumulative distribution function, i.e. quantile function for a given q for a probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames).

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistRand
[MnProbDistRand]

Returns a single random draw from a probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames).

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistRandArray
[MnProbDistRandArray]

Returns a (seeded) random array of draws from a probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames).

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistRandArrayEqSp
[MnProbDistRandArrayEqSp]

Returns a (seeded) random array of draws, using equally spaced quantiles, from a probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames).

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistSkew
[MnProbDistSkew]

Returns the skew (skewness) of a given probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames).

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistStd
[MnProbDistStd]

Returns the standard deviation of a given probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames).

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistTailValueAtRisk

[MnProbDistTailValueAtRisk]

Returns the Tail Value-At-Risk, TVaR (also called Conditional Value-at-Risk, CVaR) for a given confidence level α for a probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames), i.e.:

$$TVaR_\alpha(X) = E(-X|X \leq -VaR_\alpha) = -\frac{1}{1-\alpha}\int_{-\infty}^{-VaR_\alpha} xp(x)dx$$

where $VaR_\alpha(X) = k \quad where \int_{-\infty}^{-k} p(x)dx = 1 - \alpha$

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistTWLS

[MnProbDistTWLS]

Returns the tail weighted least squares estimation vector for a given dataset for parameters of a probability distribution given by *DistributionName* and with the order of parameter values matching the parameter names listed by MnProbDistParamNames).

The algorithm used starts from values for each parameter in line with its value in *OpeningParameterChoices* and searches for the (possibly just 'nearest') parameter selection that maximises the weighted (log) likelihood function.

An explanation of the algorithm used is given in TailWeightedParameterEstimation.pdf.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistTWLSRestricted

[MnProbDistTWLSRestricted]

Returns the tail weighted least squares estimation vector for a given dataset for parameters of a probability distribution given by *DistributionName* and with the order of parameter values matching the parameter names listed by MnProbDistParamNames) subject to the restriction that for those parameters with *AllowParameterToVary* = FALSE the estimator must be identical to its *OpeningParameterChoice*.

The algorithm used starts from values for each of the non-fixed parameters in line with its value in *OpeningParameterChoices* and searches for the (possibly just 'nearest') parameter selection that maximises the weighted (log) likelihood function.

An explanation of the algorithm used is given in TailWeightedParameterEstimation.pdf.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistTWMLE

[MnProbDistTWMLE]

Returns the tail weighted maximum likelihood estimation vector for a given dataset for parameters of a probability distribution given by *DistributionName* and with the order of parameter values matching the parameter names listed by MnProbDistParamNames).

The algorithm used starts from values for each parameter in line with its value in *OpeningParameterChoices* and searches for the (possibly just 'nearest') parameter selection that maximises the weighted (log) likelihood function.

An explanation of the algorithm used is given in TailWeightedParameterEstimation.pdf.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistTWMLERestricted

[MnProbDistTWMLERestricted]

Returns the tail weighted maximum likelihood estimation vector for a given dataset for parameters of a probability distribution given by *DistributionName* and with the order of parameter values matching the parameter names listed by MnProbDistParamNames) subject to the restriction that for those parameters with *AllowParameterToVary* = FALSE the estimator must be identical to its *OpeningParameterChoice*.

The algorithm used starts from values for each of the non-fixed parameters in line with its value in *OpeningParameterChoices* and searches for the (possibly just 'nearest') parameter selection that maximises the weighted (log) likelihood function.

An explanation of the algorithm used is given in TailWeightedParameterEstimation.pdf.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistValueAtRisk

[MnProbDistValueAtRisk]

Returns the value at risk for a given confidence level α for a probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames), i.e.:

$$VaR_{\alpha}(X) = k \quad where \int_{-\infty}^{-k} p(x)dx = 1 - \alpha$$

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistVar

[MnProbDistVar]

Returns the variance of a given probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames).

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistWeightedLikelihood

[MnProbDistWeightedLikelihood]

Returns the weighted likelihood of a given dataset for a probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames).

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistWeightedLogLikelihood

[MnProbDistWeightedLogLikelihood]

Returns the weighted log likelihood of a given dataset for a probability distribution given by *DistributionName* and with parameter values given in *ParamValues* (with their order matching the parameter names listed by MnProbDistParamNames).

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistWeightedMLE

[MnProbDistWeightedMLE]

Returns the weighted maximum likelihood estimation vector for a given dataset for parameters of a probability distribution given by *DistributionName* and with the order of parameter values matching the parameter names listed by MnProbDistParamNames).

The algorithm used starts from values for each parameter in line with its value in *OpeningParameterChoices* and searches for the (possibly just 'nearest') parameter selection that maximises the weighted (log) likelihood function.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

ProbDistWeightedMLERestricted

[MnProbDistWeightedMLERestricted]

Returns the weighted maximum likelihood estimation vector for a given dataset for parameters of a probability distribution given by *DistributionName* and with the order of parameter values matching the parameter names listed by MnProbDistParamNames) subject to the restriction that for those parameters with *AllowParameterToVary* = FALSE the estimator must be identical to its *OpeningParameterChoice*.

The algorithm used starts from values for each of the non-fixed parameters in line with its value in *OpeningParameterChoices* and searches for the (possibly just 'nearest') parameter selection that maximises the weighted (log) likelihood function.

Probability distributions currently recognised by the website may be listed using MnRecognisedProbDists.

Product

[MnProduct]

Returns the product of the elements of an array (i.e. the result of multiplying them together).

Proper

[MnProper]

Capitalizes (converts to uppercase) the first letter in a text string and any other letters in text that follow any character other than a letter and converts all other letters to lowercase letters

PVAnnuityCts

[MnPVAnnuityCts]

Returns the value/price now, V, of a annuity of 1 payable continuously for the next T years, i.e.:

$$V = \frac{1 - e^{-rT}}{1 - r}$$

PVSwapCts

[MnPVSwapCts]

Returns the value/price now, V, of a annuity of 1 times the $FixedRate$ payable continuously for the next T years less floating rate payments for the next T years of 1 times the market floating rate, i.e.:

$$V = FixedRate \times \frac{1 - e^{-rT}}{1 - r} + e^{-rT} - 1$$

PVZero

[MnPVZero]

Returns the value/price now, V, of a zero coupon bond with maturity T, i.e. the value of a payment of 1 in T years time, discounted at an annualised (continuously compounded) rate of interest of r and hence:

$$V = e^{-rT}$$

Quartile

[MnQuartile]

Returns the quartile of a series, i.e. the value of $MnPercentile(InputArray, 0.25 \times q)$ where $q = 0{,}1{,}2{,}3\ or\ 4$.

The equivalent function in Microsoft Excel is QUARTILE. See also MnPercentile.

QuartileExc

[MnQuartileExc]

Returns the quartile of a series (treating the lowest value as corresponding to the $1/(n+1)$'th percentile and the highest as corresponding to the $n/(n+1)$'th percentile), i.e. the value of $MnPercentileExc(InputArray, 0.25 \times q)$ where$q = 1{,}2\ or\ 3$.

The equivalent function in Microsoft Excel is QUARTILE.EXC. See also MnPercentileExc.

RadianToDegree

[MnRadianToDegree]

Converts a value in radians into a value in degrees

Rand

[MnRand]

Returns a uniform random number between 0 and 1.

A special case of MnProbDistRand with *DistributionName* = “uniform” and *ParamValues* = {0, 1}.

Included for compatibility with RAND() in Microsoft Excel and some other spreadsheet systems.

RandBetween

[MnRandBetween]

Returns a uniform random number between *Bottom* and *Top*.

A special case of MnProbDistRand with *DistributionName* = “uniform” and *ParamValues* = {*Bottom*, *Top*}.

Included for compatibility with RANDBETWEEN(BOTTOM,TOP) in Microsoft Excel and some other spreadsheet systems.

RandomlySelectState

[MnRandomlySelectState]

Selects the state that a system is in (0 to *n*-1) based on given state probabilities and a number between 0 and 1, which might e.g. be a uniform random number.

RandomOrdering

[MnRandomOrdering]

Returns an array containing a random ordering of the integers 0 to *n* – 1

RandomPermutation

[MnRandomPermutation]

Returns an array containing a random ordering of the integers 0 to *n* – 1 using a pre-seeded random sequence. Same as MnRandomOrdering except that the output is pre-seeded by the *RandomNumberSeed* parameter.

RecognisedBrowserFeatures

[MnRecognisedBrowserFeatures]

Returns a list of the names of features (such as browser type or screen width) relating to the browser that is accessing the function that the Nematrian website recognises and can return using MnMyBrowserFeature or MnMyBrowserFeatures. MnMyBrowserFeature returns the value of a specific feature whilst MnMyBrowserFeatures returns a list of all available features recognised by the Nematrian website.

A list of descriptions of each recognised browser feature is available using MnBrowserFeatureDescriptions.

RecognisedChartLegendDockings

[MnRecognisedChartLegendDockings]

Returns a list of the acceptable values that *LegendDocking* can take in a *ChartFormat* parameter when specifying the format of a chart to be plotted by the Nematrian website. The LegendDocking sub-parameter defines where the legend is placed that describes the various series being plotted.

For further details on how to use the *ChartFormat* parameter to format a chart see formatting Nematrian charts.

RecognisedChartSeriesColors

[MnRecognisedChartSeriesColors]

Returns a list of the acceptable values that *SeriesColor* variables can take in a *ChartFormat* parameter when specifying the format of a chart to be plotted by the Nematrian website. The *SeriesColor* sub-parameter (which can be set differently for different series being plotted at the same time) defines the colour (color) used to plot the series (e.g. Red, Green, Blue, Black).

For further details on how to use the *ChartFormat* parameter to format a chart see formatting Nematrian charts.

RecognisedChartSeriesLineStyles

[MnRecognisedChartSeriesLineStyles]

Returns a list of the acceptable values that *SeriesLineStyle* variables can take in a *ChartFormat* parameter when specifying the format of a chart to be plotted by the Nematrian website. The *SeriesLineStyle* sub-parameter (which can be set differently for different series being plotted at the same time) defines the style used to plot the series (e.g. Solid, Dash, Dot).

For further details on how to use the *ChartFormat* parameter to format a chart see formatting Nematrian charts.

RecognisedChartSeriesTypes

[MnRecognisedChartSeriesTypes]

Returns a list of the acceptable values that *SeriesType* variables can take in a *ChartFormat* parameter when specifying the format of a chart to be plotted by the Nematrian website. The *SeriesType* sub-parameter (which can be set differently for different series being plotted at the same time) defines the type of chart used to plot the series (e.g. line or point).

For further details on how to use the *ChartFormat* parameter to format a chart see formatting Nematrian charts.

RecognisedMortalityTableNumberOfVariants

[MnRecognisedMortalityTableNumberOfVariants]

Returns the number of variants for a given mortality table recognised by the website (the list which variants it recognises is available using MnRecognisedMortalityTableVariants).

Currently the Nematrian website supports a range of UK mortality tables using nomenclature derived from terminology adopted by the UK's Continuous Mortality Investigation Bureau.

RecognisedMortalityTables

[MnRecognisedMortalityTables]

Returns an array indicating which mortality tables are recognised by the website, e.g. PCMA00, PCFA00, PML92, PFA92, PFL92, PFA92, a(55)m, a(55)f.

Currently the Nematrian website supports a range of UK mortality tables as named by the UK's Continuous Mortality Investigation Bureau.

RecognisedMortalityTableVariants

[MnRecognisedMortalityTableVariants]

Returns an array indicating which table variants are recognised by the website for a given mortality table, e.g. the website recognises the following variants for PML92: "HighCohort", "MediumCohort", "LowCohort", "Base" and "OriginalReductionFactor".

Currently the Nematrian website supports a range of UK mortality tables using nomenclature derived from terminology adopted by the UK's Continuous Mortality Investigation Bureau.

RecognisedPhysicalUnits

[MnRecognisedPhysicalUnits]

Returns an array containing a list of physical units (e.g. second, volt etc.) recognised by the Nematrian website

RecognisedProbDists

[MnRecognisedProbDists]

Returns a list of probability distributions that are recognised by the Nematrian website probability distribution web functions.

RecognisedSearchSpiders

[MnRecognisedSearchSpiders]

Returns an array of string variables that include websites or names associated by the Nematrian website with search engine spiders (or 'bots' as they are colloquially called) or the equivalent. This list is unlikely to be exhaustive as new search engines come into existence and others change their names. The main purpose of this function is to help website creators identify ‘real’ traffic as opposed to traffic that might arise purely from the activities of such ‘bots’.

To test whether a string contains any of these names or website strings then use MnSearchForSearchSpiders.

RegularisedIncompleteBeta

[MnRegularisedIncompleteBeta]

The regularised incomplete beta function $I_x(p,q)$ is defined as:

$$I_x(p,q) = \frac{B_x(p,q)}{B(p,q)}$$

where $B(p,q)$ is the beta function and $B_x(p,q)$ is the incomplete beta function.

RelativeVolUsingCorr

[MnRelativeVolUsingCorr]

Returns the relative volatility (i.e. tracking error) between a portfolio and a benchmark given input standard deviations and correlations.

RelativeVolUsingCov

[MnRelativeVolUsingCov]

Returns the relative volatility (i.e. tracking error) between a portfolio and a benchmark given input covariances.

RemoveAllInstancesOfStringFromString

[MnRemoveAllInstancesOfStringFromString]

Returns a string which excludes all instances of a particular string (the *StringToRemove*) from the *InputString*.

ReorderSeries

[MnReorderSeries]

Returns a reordered series, i.e. y_i where $y_i = x_{index(i)}$. The base for $index(i)$ is taken as the lowest value of the start at *base*. Most computer advanced computer languages nowadays have base = 0, so the first entry is indexed by 0, the second by 1 etc.

RepeatString

[MnRepeatString]

Returns a string that involves an *n*-fold repeat of *StringToRepeat*.

Replace

[MnReplace]

Returns a string derived from *InputString* in which occurrences of a specified substring (*FindString*) are replaced by *ReplacementString*. The other replacements only start from the *StartingPosition* character and only the first *NumberOfTimesToReplace* are replaced (unless *NumberOfTimesToReplace* = -1 in which case all relevant occurrences are replaced).

If *IsCaseSensitive* is true then function uses 'binary' comparison so differentiates between upper and lower case letters, otherwise uses 'text' comparison and so does not differentiate between upper and lower case letters.

ResampledPortfolioOptimiser

[MnResampledPortfolioOptimiser]

Returns the mean-variance 'resampled efficient' portfolio, and its risk and return characteristics, for a given risk-reward trade-off. The risks and returns are those derived on the basis of the original (overall) *ForecastReturns* etc., from which the simulations are derived, rather than the ones being assumed in any particular simulation to be the correct values to use.

Additionally provides a range of percentile points for the simulated distribution of each individual portfolio component, if *CalculatePercentiles* is set to true, the percentiles are those passed to the function using the *PercentilePoints* array.

Output is in the form of *n* different asset mixes, then the sum of these asset mixes (which is usually constrained to equal 1), then the return and risk of the portfolio mix, then for each percentile point a value for each asset category derived from the simulated distribution of the size of that particular asset exposure for that particular risk-reward trade-off.

RestateYieldOrDiscount

[MnRestateYieldOrDiscount]

Returns a yield or discount rate from one annualisation convention onto another. The conventions used are explained in Annualisation Conventions and Annualisation Conventions: Illustrative Table.

ReverseArray

[MnReverseArray]

Returns numeric array (i.e. an array of 'double precision' values) z_i that has the order of the elements of the input array, x_i reversed, i.e. if there are n elements in the original array then $z_1 = x_n$, $z_2 = x_{n-1}$, etc.

ReverseBooleanArray

[MnReverseBooleanArray]

Returns a Boolean array z_i that has the order of the elements of the input array, x_i reversed, i.e. if there are n elements in the original array then $z_1 = x_n$, $z_2 = x_{n-1}$, etc.

ReverseDateArray

[MnReverseDateArray]

Returns a Date array z_i that has the order of the elements of the input array, x_i reversed, i.e. if there are n elements in the original array then $z_1 = x_n$, $z_2 = x_{n-1}$, etc.

ReverseIntegerArray

[MnReverseIntegerArray]

Returns an Integer array z_i that has the order of the elements of the input array, x_i reversed, i.e. if there are n elements in the original array then $z_1 = x_n$, $z_2 = x_{n-1}$, etc.

ReverseQuadraticPortfolioOptimiser

[MnReverseQuadraticPortfolioOptimiser]

Returns a vector containing the 'implied alphas' for a given set of active positions, i.e. the return assumptions that need to be held for a portfolio to be optimal (ignoring constraints), given active positions, standard deviations, a correlation matrix and a trade-off factor (i.e. risk aversion factor) that corresponds to the investor's chosen trade-off between return and risk. It is assumed that the investor has a quadratic utility function of the following form, where r is a vector of returns, V is a covariance matrix and a is a vector of active weights (i.e. $a = x - b$, where x is a vector of portfolio weights with n elements and b is the corresponding vector of benchmark weights):

$$U(x) = r.a - \lambda a^T V a$$

Please bear in mind that if a given set of returns, r, is optimal in this context then so is the set of returns defined by $pr + q$ for any constant (scalar, i.e. asset class independent) values of p and q. This function adopts the convention that $r = \lambda V a$ or equivalently $a = \lambda V^{-1} r$, i.e. that $q = 0$ and that p scales in line with the risk-return trade-off factor, i.e. λ.

Please also bear in mind that if the active position within a live portfolio is at the limit of a constraint (e.g. for a long-only portfolio the portfolio weight is zero, i.e. constrained by the long-only constraint) then it is not possible to calculate accurately the implied alpha for that position, since we do not then know (merely from the portfolio weights) how positive or negative is the view that the manager is assigning to that position.

ReverseStringArray

[MnReverseStringArray]

Returns a String array z_i that has the order of the elements of the input array, x_i reversed, i.e. if there are n elements in the original array then $z_1 = x_n$, $z_2 = x_{n-1}$, etc.

Right
[MnRight]

Returns a string containing the *n* right-most characters of *InputString*. If *n* = 0 then returns a blank string. If *n* is greater than the number of characters in *InputString* then it returns the entire *InputString*.

RoundToNdp
[MnRoundToNdp]

Returns the input value, *x*, rounded to *n* decimal places

RoundToNsf
[MnRoundToNsf]

Returns the input value, *x*, rounded to *n* significant figures

RSet
[MnRSet]

Returns a string that is a right aligned version of *InputString*, i.e. the first *n* characters of *InputString*, padded out with spaces if *n* is longer than the length of *InputString*.

RSq
[MnRSq]

Returns the r^2 of a regression analysis of known y_i versus known x_i, i.e. the square of the (Pearson) correlation coefficient between these two series.

RTrim
[MnRTrim]
Returns *InputString* but without any trailing spaces. See also MnLTrim and MnTrim.

SearchForSearchSpiders
[MnSearchForSearchSpiders]

Returns the first search spider website/name (or 'bot' as they are colloquially called) found by the Nematrian website in the input string (or returns a blank string if no such website/name is found). See MnRecognisedSearchSpiders for a list of websites or names that the Nematrian website currently views as search engine spiders or the equivalent.

This function cycles through the entries in MnRecognisedSearchSpiders stopping as soon as a match with any entry is found.

Sec
[MnSec]

Returns the secant of an angle. This is the reciprocal of the cosine function:

$$\sec x = \frac{1}{\cos x}$$

Sech
[MnSech]

Returns the hyperbolic secant of an angle. This is the reciprocal of the hyperbolic cosine function:

$$\operatorname{sech} x = \frac{1}{\cosh x}$$

Second
[MnSecond]

Returns the second of a given date/time

SeriesSum
[MnSeriesSum]

Returns the sum of a power series, so if the coefficients are a_i $(i = 1, \dots, j)$ then returns:

$$S = a_1 x^n + a_2 x^{n+m} + \cdots + a_j x^{n+m\times(j-1)}$$

SessionIdExpires
[MnSessionIdExpires]

Returns the date / time when the relevant *SessionId* expires

SetOfMembersOfArray
[MnSetOfMembersOfArray]

Returns a numeric array (i.e. an array of 'double precision' values) containing the distinct members of the *InputArray*, e.g. if *InputArray* is {3.0, 5.0, 5.0, 4.0} then it returns an array with 3 elements namely {3.0, 5.0, 4.0}

SetOfMembersOfDateArray
[MnSetOfMembersOfDateArray]

Returns a Date array containing the distinct members of the *InputArray*, e.g. if *InputArray* is {3 Jan 1980, 5 Jan 1980, 5 Jan 1980, 4 Jan 1980} then it returns an array with 3 elements namely {3 Jan 1980, 5 Jan 1980, 4 Jan 1980}

SetOfMembersOfIntegerArray
[MnSetOfMembersOfIntegerArray]

Returns an Integer array containing the distinct members of the *InputArray*, e.g. if *InputArray* is {3, 5, 5, 4} then it returns an array with 3 elements namely {3, 5, 4}

SetOfMembersOfStringArray
[MnSetOfMembersOfStringArray]

Returns a String array containing the distinct members of the *InputArray*, e.g. if *InputArray* is {"3", "5", "5", "4"} then it returns an array with 3 elements namely {"3", "5", "4"}

SIDimensionNames
[MnSIDimensionNames]

Returns the 7 base SI dimensions, see also SIUnitDefinitions.

SIDimensionsOf
[MnSIDimensionsOf]

Returns the SI Dimensions of the physical unit referred to by *UnitName*. This is an array of seven entries showing the applicable exponent of the 7 base SI dimensions, see SIUnitDefinitions and MnSIDimensionNames.

Sign
[MnSign]

Returns (as a number) the sign of another number, x, i.e. -1 if $x < 0$, 1 if $x > 0$ and 0 if $x = 0$

SimplifyFraction
[MnSimplifyFraction]

Returns a two element array containing as its first term c and its second term d which is the canonically simplified form of the input fraction a/b where a and b are both integral and the input is in the form of a two element array consisting of a and b respectively.

The canonically simplified form of such a fraction is c/d where d is positive and the greatest common divisor of $|c|$ and d is 1.

SimplifyFraction2

[MnSimplifyFraction2]

Returns a two element array containing as its first term c and its second term d which is the canonically simplified form of the input fraction a/b where a and b are both integral and the input is in the form of two separate integers, a and b respectively.

The canonically simplified form of such a fraction is c/d where d is positive and the greatest common divisor of $|c|$ and d is 1.

SimulateGaussianMixture

[MnSimulateGaussianMixture]

Returns an array providing simulated output from a multivariate time series model of the world involving one or more states or regimes, each of which is characterised by a Gaussian (i.e. multivariate normal) distribution, with a Markov chain process indicating how likely it is to move between each state over a given time period. The output is 2 dimensional, with the first dimension characterising the simulation and the time period and the second dimension providing a vector of the variables themselves.

Models where each state itself consists of a predefined (distributional) mixture of multivariate normal distributions can be accommodated in such a model by defining the Markov chain appropriately.

The function includes parameters that:

(a) define the starting state or how it may itself be simulated
(b) include a random number seed so that the results can be reproduced subsequently
(c) include sampling algorithms that help to reduce run times by sampling in a uniform manner across the quantile range that the individual random variables can take

Sin

[MnSin]

Returns the sine of a (real) number (the input value being expressed in radians)

Sinh

[MnSinh]

Returns the hyperbolic sine of a (real) number, i.e.

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

Skew

[MnSkew]

Returns the (sample) skew, γ_1 (also sometimes called ‘skewness’) of a series x_i (for $i = 1, \dots, n$), calculated as follows:

$$\gamma_1 = \frac{n}{(n-1)(n-2)} \sum \left(\frac{x_i - \bar{x}}{s} \right)^3$$

Here, $\bar{x}$ is the mean and s is the (sample) standard deviation of the series.

The equivalent function in Microsoft Excel is SKEW.

Slice2dArray

[MnSlice2dArray]

Returns a vector array that is a cross-sectional slice from a two dimensional array

Slope

[MnSlope]

Returns the slope from a linear regression, i.e. b where:

$$b = \frac{\sum_{i=1}^{n}(y - \bar{y})(x - \bar{x})}{\sum_{i=1}^{n}(x - \bar{x})^2}$$

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

SolvencyII_SCRSFCombineStresses

[MnSolvencyII_SCRSFCombineStresses]

Returns the results of combining different stresses (for different sub-modules and for the overall Basic SCR) when calculating the Solvency II standard formula Solvency Capital Requirement. For market risk module and for some Stress Sets, one or other of the ‘interest rate (down)’ and ‘interest rate (up)’ stresses needs to be zero for the computations to work correctly.

SolvencyII_SCRSFLifeCorrs

[MnSolvencyII_SCRSFLifeCorrs]

Returns an array containing correlation coefficients applicable to the life risk sub-module of the Solvency II standard formula SCR calculation. If there are n different stress names then is an array with n^2 terms, ordered consistently with the ordering of the stress names given by MnSolvencyII_SCRSFLifeStressNames.

SolvencyII_SCRSFLifeStressNames
[MnSolvencyII_SCRSFLifeStressNames]

Returns an array containing Stress Names to which the correlations extracted by MnSolvencyII_SCRSFLifeCorrs apply, for a recognised *StressSetName*. Recognised *StressSetNames* are given by MnSolvencyII_SCRSFStressSetNames.

See also Nematrian Solvency II Tools.

SolvencyII_SCRSFMktCorrs
[MnSolvencyII_SCRSFMktCorrs]

Returns an array containing correlation coefficients applicable to the market risk sub-module of the Solvency II standard formula SCR calculation. If there are n different stress names then is an array with n^2 terms, ordered consistently with the ordering of the stress names given by MnSolvencyII_SCRSFMktStressNames.

SolvencyII_SCRSFMktStressNames
[MnSolvencyII_SCRSFMktStressNames]

Returns an array containing Stress Names to which the correlations extracted by MnSolvencyII_SCRSFMktCorrs apply, for a recognised *StressSetName*. Recognised *StressSetNames* are given by MnSolvencyII_SCRSFStressSetNames.

See also Nematrian Solvency II Tools.

SolvencyII_SCRSFNonLifeCorrs
[MnSolvencyII_SCRSFNonLifeCorrs]

Returns an array containing correlation coefficients applicable to the market risk sub-module of the Solvency II standard formula SCR calculation. If there are n different stress names then is an array with n^2 terms, ordered consistently with the ordering of the stress names given by MnSolvencyII_SCRSFNonLifeStressNames.

MnSolvencyII_SCRSFNonLifeStressNames

Returns an array containing Stress Names to which the correlations extracted by MnSolvencyII_SCRSFNonLifeCorrs apply, for a recognised *StressSetName*. Recognised *StressSetNames* are given by MnSolvencyII_SCRSFStressSetNames.

See also Nematrian Solvency II Tools.

SolvencyII_SCRSFOverallCorrs
[MnSolvencyII_SCRSFOverallCorrs]

Returns an array containing correlation coefficients applicable to the market risk sub-module of the Solvency II standard formula SCR calculation. If there are n different stress names then is an array with n^2 terms, ordered consistently with the ordering of the stress names given by MnSolvencyII_SCRSFOverallStressNames.

SolvencyII_SCRSFOverallStressNames

[MnSolvencyII_SCRSFOverallStressNames]

Returns an array containing Stress Names to which the correlations extracted by MnSolvencyII_SCRSFOverallCorrs apply, for a recognised *StressSetName*. Recognised *StressSetNames* are given by MnSolvencyII_SCRSFStressSetNames.

See also Nematrian Solvency II Tools.

SolvencyII_SCRSFStressSetNames

[MnSolvencyII_SCRSFStressSetNames]

Returns an array containing names of Stress Sets currently recognised by:

- MnSolvencyII_SCRSFMktCorrs
- MnSolvencyII_SCRSFLifeCorrs
- MnSolvencyII_SCRSFNonLifeCorrs
- MnSolvencyII_SCRSFOverallCorrs
- MnSolvencyII_SCRSFCombineStresses
- MnSolvencyII_SCRSFTermStructureStressFactors and
- MnSolvencyII_SCRSFTermStructureStressedRate

for Solvency II standard formula SCR purposes.

See also Nematrian Solvency II Tools.

SolvencyII_SCRSFTermStructureStressedRate

[MnSolvencyII_SCRSFTermStructureStressedRate]

Returns the stressed interest rate applicable by applying a given Stress (as defined by *StressName*) to a cash flow payable at time *t* from now, assuming that the unstressed interest rate is *r*. Proposed (term dependent) factor has changed during consultation process, so table returned varies according to choice of *StressSetName*, see MnSolvencyII_SCRSFStressSetNames and according to whether the stress direction is “up” or “down”.

See Nematrian Solvency II Tools for further details.

SolvencyII_SCRSFTermStructureStressFactors

[MnSolvencyII_SCRSFTermStructureStressFactors]

Returns an array *MktInt* containing a table setting out the (term dependent) *MktInt* stress factor mandated under the Solvency II standard form Solvency Capital Requirement for given time to cash flow. Stress test involves testing both an upward and a downward move in interest rates (and potentially involves a floor on the minimum magnitude of the downward movement thus tested). Proposed (term dependent) factors have changed during consultation process, so table returned varies according to choice of *StressSetName*, see MnSolvencyII_SCRSFStressSetNames and according to whether the stress direction is “up” or “down”.

See Nematrian Solvency II Tools for further details.

SortedArray

[MnSortedArray]

Returns a numeric array (i.e. an array of ‘double precision’ values) which is the result of sorting the *InputArray*, sorted in ascending order.

SimplifyFraction

[MnSortedDateArray

Returns a Date array which is the result of sorting the *InputArray*, sorted in ascending order.

SortedIntegerArray

[MnSortedIntegerArray]

Returns an Integer array which is the result of sorting the *InputArray*, sorted in ascending order.

SortedIntersectionOfDateSets

[MnSortedIntersectionOfDateSets]

Returns a sorted Date array representing the elements of the union of the distinct members of two input arrays, e.g. if *InputArray1* is {3 Jan 1980, 5 Jan 1980, 5 Jan 1980, 4 Jan 1980} and *InputArray2* is {2 Jan 1980, 5 Jan 1980, 4 Jan 1980, 4 Jan 1980} then returns {4 Jan 1980, 5 Jan 1980}

SortedIntersectionOfDateSets

[MnSortedIntersectionOfDateSets]

Returns a sorted Date array representing the elements of the union of the distinct members of two input arrays, e.g. if *InputArray1* is {3 Jan 1980, 5 Jan 1980, 5 Jan 1980, 4 Jan 1980} and *InputArray2* is {2 Jan 1980, 5 Jan 1980, 4 Jan 1980, 4 Jan 1980} then returns {4 Jan 1980, 5 Jan 1980}

SortedIntersectionOfIntegerSets

[MnSortedIntersectionOfIntegerSets]

Returns a sorted Integer array representing the elements of the union of the distinct members of two input arrays, e.g. if *InputArray1* is {3, 5, 5, 4} and *InputArray2* is {2, 5, 4, 4} then returns {4, 5}

SortedIntersectionOfSets

[MnSortedIntersectionOfSets]

Returns a sorted numeric array (i.e. an array of 'double precision' values) representing the elements of the union of the distinct members of two input arrays, e.g. if *InputArray1* is {3.0, 5.0, 5.0, 4.0} and *InputArray2* is {2.0, 5.0, 4.0, 4.0} then returns {4, 5}

SortedIntersectionOfStringSets

[MnSortedIntersectionOfStringSets]

Returns a sorted String array representing the elements of the union of the distinct members of two input arrays, e.g. if *InputArray1* is {"3", "5", "5", "4"} and *InputArray2* is {"2", "5", "4", "4"} then returns {"4", "5"}

SortedSetOfMembersOfArray

[MnSortedSetOfMembersOfArray]

Returns a sorted numeric array (i.e. an array of 'double precision' values) containing the distinct members of the *InputArray*, e.g. if *InputArray* is {3.0, 5.0, 5.0, 4.0} then it returns a sorted array with 3 elements namely {3.0, 4.0, 5.0}

SortedSetOfMembersOfDateArray

[MnSortedSetOfMembersOfDateArray]

Returns a sorted Date array containing the distinct members of the *InputArray*, e.g. if *InputArray* is {3 Jan 1980, 5 Jan 1980, 5 Jan 1980, 4 Jan 1980} then it returns a sorted array with 3 elements namely {3 Jan 1980, 4 Jan 1980, 5 Jan 1980}

SortedSetOfMembersOfIntegerArray

[MnSortedSetOfMembersOfIntegerArray]

Returns a sorted Integer array containing the distinct members of the *InputArray*, e.g. if *InputArray* is {3, 5, 5, 4} then it returns a sorted array with 3 elements namely {3, 4, 5}

SortedSetOfMembersOfStringArray

[MnSortedSetOfMembersOfStringArray]

Returns a sorted String array containing the distinct members of the *InputArray*, e.g. if *InputArray* is {"3", "5", "5", "4"} then it returns a sorted array with 3 elements namely {"3", "4", "5"}

SortedStringArray

[MnSortedStringArray]

Returns a String array which is the result of sorting the *InputArray*, sorted in ascending order.

There are several different conventions that can be used when ordering strings (e.g. do numbers come before or after letters, do capital letters come before lower case letters). Nematrian merely borrows the standard included in Visual Studio. We suggest that you play with the example page applicable to this function, see below, to ascertain exactly what this convention involves.

SortedUnionOfDateSets

[MnSortedUnionOfDateSets]

Returns a sorted Date array representing the elements of the union of the distinct members of two input arrays, e.g. if *InputArray1* is {3 Jan 1980, 5 Jan 1980, 5 Jan 1980, 4 Jan 1980} and *InputArray2* is {2 Jan 1980, 5 Jan 1980, 4 Jan 1980, 4 Jan 1980} then returns {2 Jan 1980, 3 Jan 1980, 4 Jan 1980, 5 Jan 1980}

SortedUnionOfIntegerSets

[MnSortedUnionOfIntegerSets]

Returns a sorted Integer array representing the elements of the union of the distinct members of two input arrays, e.g. if *InputArray1* is {3, 5, 5, 4} and *InputArray2* is {3, 5, 5, 2} then returns {2, 3, 4, 5}

SortedUnionOfSets

[MnSortedUnionOfSets]

Returns a sorted numeric array (i.e. an array of ‘double precision’ values) representing the elements of the union of the distinct members of two input arrays, e.g. if *InputArray1* is {3.0, 5.0, 5.0, 4.0} and *InputArray2* is {3.0, 5.0, 5.0, 2.0} then returns {2.0, 3.0, 4.0, 5.0}

SortedUnionOfStringSets

[MnSortedUnionOfStringSets]

Returns a sorted String array representing the elements of the union of the distinct members of two input arrays, e.g. if *InputArray1* is {“3”, “5”, “5”, “4”} and *InputArray2* is {“2”, “5”, “4”, “4”} then returns {“2”, “3”, “4”, “5”}

Space

[MnSpace]

Returns a string that involves the space character repeated *n* times. A special case of MnRepeatString

SpearmanRankCorrelation

[MnSpearmanRankCorrelation]

Returns the Spearman rank correlation coefficient $c_{1,2}$ between two series, $x_{1,t}$ and $x_{2,t}$.

The Spearman rank correlation coefficient is calculated as:

$$c_{1,2} = \frac{\frac{1}{n-1}\sum_{t=1}^{n}(q_{1,t}-\bar{q}_1)(q_{2,t}-\bar{q}_2)}{\sqrt{\frac{1}{n-1}\sum_{t=1}^{n}(q_{1,t}-\bar{q}_1)^2 \cdot \frac{1}{n-1}\sum_{t=1}^{n}(q_{2,t}-\bar{q}_2)^2}}$$

where $q_{i,t}$ are the ranks of the values in the series $x_{i,t}$, i.e. $q_{i,1}$ is the largest value in the series $\{x_{i,t}: t = 1, \dots, n\}$, $\bar{q}_i$ is the mean of the $\{q_{i,t}: t = 1, \dots, n\}$ etc.

SpearmanRankCorrelations

[MnSpearmanRankCorrelations]

Returns an array of m^2 elements corresponding to the elements of the matrix, C, of Spearman rank correlation coefficients for m series that are provided to the function as an array of mn elements, where the first n elements correspond to the terms in the first series, the next n elements to the corresponding terms in the second series etc.

Spearman rank correlation coefficients are calculated as:

$$c_{i,j} = \frac{\frac{1}{n-1}\sum_{t=1}^{n}(q_{i,t}-\bar{q}_i)(q_{j,t}-\bar{q}_j)}{\sqrt{\frac{1}{n-1}\sum_{t=1}^{n}(q_{i,t}-\bar{q}_i)^2 \cdot \frac{1}{n-1}\sum_{t=1}^{n}(q_{j,t}-\bar{q}_j)^2}}$$

where $q_{i,t}$ are the ranks of the values in the series $x_{i,t}$, i.e. $q_{i,1}$ is the largest value in the series $\{x_{i,t}: t = 1, \dots, n\}$, $\bar{q}_i$ is the mean of the $\{q_{i,t}: t = 1, \dots, n\}$ etc.

Split

[MnSplit]

Subdivides a single string into an array of sub-strings, identified in the original string because they are separated by the *Delimiter*. The maximum number of sub-strings is *Limit* (unless *Limit* = -1 when an exhaustive splitting is carried out).

If *IsCaseSensitive* is true then function uses ‘binary’ comparison so differentiates between upper and lower case letters, otherwise uses ‘text’ comparison and so does not differentiate between upper and lower case letters.

SpearmanRankCorrelations

[MnSqrt

Returns the square root, $\sqrt{x}$, of a (real, non-negative) number

SqrtPi

[MnSqrtPi]

Returns $\sqrt{\pi x}$ where x is the input value and π is the ratio between the circumference of a circle and its diameter.

StandardisedNormalQuantiles

[MnStandardisedNormalQuantiles]

Returns a series which contains the standardised normal quantile points, i.e. if size is n then returns a series whose elements x_i $(i = 1, \dots, n)$ are:

$$x_i = N^{-1}\left(\frac{i - 1/2}{n}\right)$$

StandardWeightedCubicQuantileFit

[MnStandardWeightedCubicQuantileFit]

Returns an array corresponding to a weighted cubic curve fit to observed quantile values, where the weights correspond to the distance between consecutive points (so that the fit gives greater weight to the tails of the distribution). Requires as inputs an ordered array of x values (e.g. expected quantiles, as per MnStandardisedNormalQuantiles and an ordered array of y values corresponding to the actually observed values.

The curve fit is derived using the Nematrian least squares generalised curve fit MnLeastSquaresGeneralisedCurveFit, (specifically the polynomial version).

The weights, w_i, used in this function, if the ordered x-values are x_i and there are n such values, are:

$$w_i = \begin{cases} 0, \ \ i = 1 \\ (x_{i+1} - x_{i-1})/2 \, , \ \ 1 < i < n \\ 0, \ \ i = n \end{cases}$$

This potentially under allows for the two end points. An alternative that gives greater weight to the two end points is MnStandardWeightedCubicQuantileFitInclEnds.

Please bear in mind that a cubic is not always a valid form for a quantile-quantile function to take (e.g. if the slope of the cubic becomes negative anywhere). If you wish to avoid this problem then we suggest fitting a distributional form using e.g. (weighted) maximum likelihood, see e.g. MnProbDistWeightedMLE.

StandardWeightedCubicQuantileFitInclEnds

[MnStandardWeightedCubicQuantileFitInclEnds]

Returns an array corresponding to a weighted cubic curve fit to observed quantile values, where the weights correspond to the distance between consecutive points (so that the fit gives greater weight to the tails of the distribution). Requires as inputs an ordered array of x values (e.g. expected quantiles, as per MnStandardisedNormalQuantiles and an ordered array of y values corresponding to the actually observed values.

The curve fit is derived using the Nematrian least squares generalised curve fit MnLeastSquaresGeneralisedCurveFit, (specifically the polynomial version).

The weights, w_i, used in this function, if the ordered x-values are x_i and there are n such values, are:

$$w_i = \begin{cases} x_2 - x_1, & i = 1 \\ (x_{i+1} - x_{i-1})/2, & 1 < i < n \\ x_n - x_{n-1}, & i = n \end{cases}$$

An alternative that gives zero weight to the two end points is MnStandardWeightedCubicQuantileFit.

Please bear in mind that a cubic is not always a valid form for a quantile-quantile function to take (e.g. if the slope of the cubic becomes negative anywhere). If you wish to avoid this problem then we suggest fitting a distributional form using e.g. (weighted) maximum likelihood, see e.g. MnProbDistWeightedMLE.

Stdev

[MnStdev]

Returns the (sample) standard deviation s of a series x_i (for $i = 1, \dots, n$), calculated as follows:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

(where $\bar{x}$ is the mean of the series)

The equivalent function in Microsoft Excel is STDEV.

Str

[MnStr]

Returns a string representation of *InputValue*

StrComp

[MnStrComp]

Returns a value of -1, 0 or +1 depending on whether *InputString1* is deemed less than, equal to or greater than *InputString2*.

If *IsCaseSensitive* is true then function uses ‘binary’ comparison so differentiates between upper and lower case letters, otherwise uses ‘text’ comparison and so does not differentiate between upper and lower case letters.

StringToBaseNArray

[MnStringToBaseNArray]

Returns a integer array representation of an integer represented as an string, *InputString*, with number base *nBase*, with order of elements of the output being in order of increasing power of *nBase* but with the characters of *InputString* being in order of decreasing power of *nBase* (in line with e.g. normal Arabic decimal and binary representation conventions).

StrReverse

[MnStrReverse]

Returns *InputString* with the character order reversed

SudokuSolve

[MnSudokuSolve]

Returns the solution to a Sudoku-style puzzle if given input in a suitable form. See also Standard Sudoku Solver and Killer Sudoku Solver, which provide interactive ways of accessing this tool.

The easiest way to understand the required input format is to consider the nature of a Killer Sudoku puzzle. In this Sudoku-variant, there are various groupings of cells scattered across the 9 by 9 grid (usually contiguous). These each have defined numbers of cells within them and a defined total (e.g. there might be a grouping of 2 cells the total of which needs to be 3 (and thus the first cell of the pair forming this group can only be 1 and the second 2 or vice-versa). Each entry in a group needs to be different, so there are at most 9 cells in any one group. For, say, the i'th group, we thus know at outset the number of cells, n_i, in the group, the sum of the entries, T_i, of the cells in the group and the coordinates of each cell, $\left((x_1, y_1), (x_2, y_2), \ldots\right)_i$.

The required input form involves an array of bytes, in groups of 20 consecutive entries (so if there are 30 groups then the input array needs to be $30 \times 20 = 600$ entries long), ordered as follows:

- First entry = number of cells in the group, i.e. n_i
- Second entry = required sum of entries in these cells, i.e. T_i
- Third entry = x-coordinate of the first cell in the group, i.e. x_1
- Fourth entry = y-coordinate of the first cell in the group, i.e. y_1
- Fifth entry = x-coordinate of the second cell in the group, i.e. x_2
- Sixth entry = y-coordinate of the second cell in the group, i.e. y_2, etc. (with the last entries in the group of 20 ignored if the group has less than 9 cells in it).

Also takes as a further input the type of the Sudoku puzzle (i.e. 'Killer', 'Standard', 'Samurai' or 'General'). These simplify the inputs (for the first three) so that you do not need each time to specify the standard features of the puzzle. For example, with a Killer Sudoku puzzle, we can think of the overarching grid layout as defining a further set of groups, since each row, column and principal 3-by-3 group needs to contain all the numbers 1 to 9. They can therefore be coded as akin to a group like the one above but with $n = 9$, $T = 45$ and the remaining entries in the set of 20 consecutive bytes defining the positioning of each cell (from 1 to 9). If you choose the 'General' type you will need to enter these groups yourself. The solver will also not then check the input data for consistency (e.g. with a Killer Sudoku puzzle all cell coordinates in the 9 by 9 grid need to be in the range 1 to 9, each cell needs to appears just once in the definition of groups as above, and the sum of all of the group totals is 405).

With a 'Standard' Sudoku puzzle, the same input format is used as above. Each known entry in the grid forms corresponds to its own group, with $n = 1$, T = the (known) entry in the cell, and (x_1, y_1) identifying the location of that cell.

'Samurai' Sudoku involves a more complex grid, but can be solved in a similar fashion, just with more initial groups.

The solver uses an exhaustive search approach, so should identify a unique solution if there is one. However, it does include an iteration upper limit which you might need to set reasonably high for more complicated puzzles.

Sum

[MnSum]

Returns the sum of the elements of a numeric array.

SumProduct

[MnSumProduct]

Returns the sum of the product of two series, i.e. if the elements of the first array are x_i and the elements of the second array are y_i (for $i = 1, \dots, n$) then returns S where

$$S = \sum_{i=1}^{n} x_i y_i$$

Provided for compatibility with the SUMPRODUCT function in Microsoft Excel and some other spreadsheet systems.

SumSq

[MnSumSq]

Returns the sum of the squares of an array, i.e. if the elements of the array are x_i (for $i = 1, \dots, n$) then returns y where

$$y = \sum_{i=1}^{n} x_i^2$$

Provided for compatibility with the SUMSQ function in Microsoft Excel and some other spreadsheet systems.

SumX2MY2

[MnSumX2MY2]

Returns the sum of the differences of the squares of two arrays, i.e. if the elements of the first array are x_i and the elements of the second array are y_i (for $i = 1, \dots, n$) then returns S where

$$S = \sum_{i=1}^{n} x_i^2 - y_i^2$$

Provided for compatibility with the SUMX2MY2 function in Microsoft Excel and some other spreadsheet systems.

SumX2PY2

[MnSumX2PY2]

Returns the sum of the sums of the squares of two arrays, i.e. if the elements of the first array are x_i and the elements of the second array are y_i (for $i = 1, \dots, n$) then returns S where:

$$S = \sum_{i=1}^{n} x_i^2 + y_i^2$$

Provided for compatibility with the SUMX2PY2 function in Microsoft Excel and some other spreadsheet systems.

SumXMY2

[MnSumXMY2]

Returns the sum of the squares of the differences between two arrays, i.e. if the elements of the first array are x_i and the elements of the second array are y_i (for $i = 1, \dots, n$) then returns S where:

$$S = \sum_{i=1}^{n} (x_i - y_i)^2$$

Provided for compatibility with the SUMXMY2 function in Microsoft Excel and some other spreadsheet systems.

Tan

[MnTan]

Returns the tangent of a (real) number (the input value being expressed in radians)

SumXMY2

[MnTanh

Returns the hyperbolic tangent of a (real) number, i.e.:

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

TDistCdf

[MnTDistCdf]

Returns the cumulative distribution function of the Student's *t* distribution. Provided for compatibility with TDIST, T.DIST (with cumulative parameter set to true), T.DIST.2T and T.DIST.RT functions in Microsoft Excel 2010 and later.

Note:

(a) TDIST is only defined if $x \geq 0$ and as the Student's t distribution is symmetric this means that TDIST (1 tailed) is ≤ 0.5 and TDIST (2 tailed) is ≤ 1
(b) T.DIST.RT = 1 - T.DIST(cumulative=true) and T.DIST(cumulative=true) = MnTDistCdf
(c) If if $x \geq 0$ then TDIST (1 tailed) = T.DIST.RT and TDIST(2 tailed) = 2 * TDIST (1 tailed)
(d) T.DIST.2T = TDIST (2 tailed)

Please also bear in mind that this function accepts non-integral values of *nu* whereas the corresponding Excel functions are limited to integral values.

Equivalent to calling MnProbDistCdf using as the distribution parameter "student's t".

TDistPdf

[MnTDistPdf]

Returns the probability density function of the Student's *t* distribution. Provided for compatibility with T.DIST (with cumulative parameter set to false) functions in Microsoft Excel 2010 and later.

Please also bear in mind that this function accepts non-integral values of *nu* whereas the corresponding Excel functions are limited to integral values.

Equivalent to calling MnProbDistPdf using as the distribution parameter "student's t".

TimeOfDay

[MnTimeOfDay]

Returns the current time of day (of server) as a Date value

Timer

[MnTimer]

Returns the number of seconds that have elapsed since midnight (for the server)

SumXMY2

[MnTimeSerial

Returns a time given separate inputs for hour, minute and second

TimeString

[MnTimeString]

Returns a string representation of the current date/time (of the server)

TimeValue

[MnTimeValue]

Returns the time represented by a given date/time in string representation, ignoring the date

TInv

[MnTInv]

Returns the inverse distribution function (i.e. quantile function) of the Student’s *t* distribution. Provided for compatibility with T.INV and LOGNORM.INV functions in Microsoft Excel 2010 and later.

Please also bear in mind that this function accepts non-integral values of *nu* whereas the corresponding Excel functions are limited to integral values.

Equivalent to calling MnProbDistQuantile using as the distribution parameter “student's t”.

Today

[MnToday]

Returns current date (of server)

ToSexagesimal

[MnToSexagesimal]

Returns an array corresponding to a number expressed in sexagesimal format. Please bear in mind that a negative number expressed in sexagesimal format such as -3° 4´ 22.5˝ corresponds to the number $-(3 + 4/60 + 22.5/3600)$, i.e. the second and third entries should be positive and the sign of the first is applied to all three.

TrailingVolatilityAdjustArray

[MnTrailingVolatilityAdjustArray]

Returns a series, y_i, derived by applying a trailing volatility adjustment (over the preceding m entries) to an input series, x_i, i.e.:

$$y_i = \frac{x_{i+m}}{sd_i}$$

where sd_i is the standard deviation of the entries $(x_i, x_{i+1}, \ldots, x_{i+m-1})$

Trim

[MnTrim]

Returns *InputString* but without any leading and trailing spaces. See also MnLTrim and MnTrim.

Truncate

[MnTruncate]

Rounds a number towards 0, e.g. rounds 7.1 to 7 and -7.1 to -7

UCase

[MnUCase]

Returns the upper case equivalent of *InputString* (i.e. replacing e.g. ‘a’ by ‘A’ etc.)

Uminus

[MnUminus]

Returns a number with the sign reversed (i.e. equivalent to the unary minus operator).

Provided for compatibility with the UMINUS function in Google Drive.

UnionOfDateSets

[MnUnionOfDateSets]

Returns a Date array representing the elements of the union of the distinct members of two input arrays, e.g. if *InputArray1* is {3 Jan 1980, 5 Jan 1980, 5 Jan 1980, 4 Jan 1980} and *InputArray2* is {2 Jan 1980, 5 Jan 1980, 4 Jan 1980, 4 Jan 1980} then returns {3 Jan 1980, 5 Jan 1980, 4 Jan 1980, 2 Jan 1980}

UnionOfIntegerSets

[MnUnionOfIntegerSets]

Returns an Integer array representing the elements of the union of the distinct members of two input arrays, e.g. if *InputArray1* is {3, 5, 5, 4} and *InputArray2* is {3, 5, 5, 2} then returns {3, 5, 4, 2}

UnionOfSets
[MnUnionOfSets]

Returns a numeric array (i.e. an array of 'double precision' values) representing the elements of the union of the distinct members of two input arrays, e.g. if *InputArray1* is {3.0, 5.0, 5.0, 4.0} and *InputArray2* is {3.0, 5.0, 5.0, 2.0} then returns {3.0, 5.0, 4.0, 2.0}

UnionOfStringSets
[MnUnionOfStringSets]

Returns a String array representing the elements of the union of the distinct members of two input arrays, e.g. if *InputArray1* is {"3", "5", "5", "4"} and *InputArray2* is {"2", "5", "4", "4"} then returns {"3", "5", "4", "2"}

UnitNormalDensity
[MnUnitNormalDensity]

Returns $f(z)$, the probability density function of a unit normal, i.e. $N(0,1)$, distribution, where:

$$f(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2}$$

The equivalent function in Microsoft Excel is NORMSDENS.

Is a special case of MnProbDistPdf.

UnivariatePolynomialMultiply
[MnUnivariatePolynomialMultiply]

Returns an array with the coefficients of a polynomial formed by multiplying together two polynomials (each one specified by an array of coefficients, where $a_0 + a_1x + a_2x^2 + \cdots$ is characterised by the array $\{a_0, a_1, a_2, \dots\}$

UploadPGPD
[MnUploadPGPD]

Temporarily uploads a peer group performance data set onto the Nematrian server.

Note: The input period start/ends need to be consecutive, e.g. if there are two periods with the first one from 1 Jan 2000 to 31 Dec 2000 and the second from 1 Jan 2001 to 31 Dec 2001 then these should be passed as an array with three entries, 31 Dec 1999, 31 Dec 2000 and 31 Dec 2001 in that order. The sizes/format of the arrays passed as parameters for this web function as specified below assume that the number of Funds is m and the number of consecutive periods is n. Return data should be passed with the consecutive returns for the same fund in a single block, i.e. all of the first fund's returns, followed by all of the second fund's returns etc. If a fund does not have a return for a given period (e.g. the fund did not exist for the whole of the period) then use for that fund for that period the *ReturnInvalidCode*. When calculating per group level statistics, e.g. percentiles and median, funds that did not have a return for a given period are excluded.

Uplus
[MnUplus]

Returns a specified number, unchanged (i.e. equivalent to the unary plus operator).

Provided for compatibility with the UPLUS function in Google Drive.

Val
[MnVal]

Converts *InputString* to a number, if possible

ValidNematrianWebFunctions
[MnValidNematrianWebFunctions]

Returns a list of all currently recognised Nematrian web functions.

ValueFromValueWithBracketedError
[MnValueFromValueWithBracketedError]

Returns the 'base' value from an input (string) expression that involves a measurement together with a bracketed term (and possibly an exponent), e.g. applying this to 1.345(34)e-12 or 1.345(34)E-12 should return 1.345 x 10^(-12), whilst 1.345(34) or 1.345 should return 1.345. All space spaces in the input string are ignored (it is conventional in some contexts to write, say, 1.345674 as 1.345 674). Please bear in mind that for this type of function the Nematrian website only accepts decimal delimiters in the form of a full stop, i.e. '.' and only accepts integral exponents.

As with all other Nematrian floating point functions, accuracy is limited to the machine accuracy of the Nematrian servers.

ValueOfAnnuity
[MnValueOfAnnuity]

Returns the approximate value of an annuity payable to a person of a given age, with possible contingent spouse's annuity payable on death of main life. Function caters for guarantee periods and for various different mortality tables and table variants but uses approximations to cater for, e.g. different payment frequencies.

A specific input parameter defines choice of annualisation convention for the input interest rate. If interest rates are annualised then the appropriate input value for *AnnualisationConvention* is 1. If interest rates are continuously compounded then the appropriate input value for

AnnualisationConvention is 0 etc. The function MnRestateYieldOrDiscount may be used to convert between different annualisation conventions.

PaymentTimingDescription needs to be one of “arrears”, “advance” or “continuous”. If it is “arrears” or “advance” then payments occur at the end or the beginning of periods of length 1/*PaymentFrequency* (in years). If it is “continuous” then payments are assumed to occur continuously (and *PaymentFrequency* needs to be zero).

Currently the Nematrian website supports a range of UK mortality tables using nomenclature derived from terminology adopted by the UK’s Continuous Mortality Investigation Bureau. Please speak to your usual Nematrian contact if you would like this function to calculate annuities using mortality tables and improvement factors that are not yet supported by the Nematrian website.

Variance

[MnVariance]

Returns the (sample) variance v of a series x_i (for $i = 1, \dots, n$), calculated as follows:

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

(where $\bar{x}$ is the mean of the series)

The equivalent function in Microsoft Excel is VAR.

If the observations are drawn from a normal distribution then the sample standard deviation is an unbiased estimate of the underlying population standard deviation.

WeightedCorrelation

[MnWeightedCorrelation]

Returns the weighted (Pearson) correlation between two series.

The weighted correlation for two series, 1 and 2, given weights w_i (for $i = 1, \dots, n$), is calculated as:

$$\tilde{c}_{1,2} = \frac{\tilde{v}_{1,2}}{\sqrt{\tilde{s}_1^2 \tilde{s}_2^2}}$$

where

$$\tilde{v}_{j,k} = \frac{(\sum w_i)(\sum w_i(x_{j,i} - \tilde{x}_j)(x_{k,i} - \tilde{x}_k))}{(\sum w_i)^2 - \sum w_i^2}$$

$$\tilde{s}_j = \sqrt{\frac{(\sum w_i)\left(\sum w_i(x_{j,i} - \tilde{x}_j)^2\right)}{(\sum w_i)^2 - \sum w_i^2}}$$

$$\tilde{x}_j = \frac{\sum w_i x_{j,i}}{\sum w_i}$$

See Weighted Moments and Cumulants for further details and explanation of terms in this formula. If the weights are equal then returns the same as MnCorrelation.

WeightedCorrelations

[MnWeightedCorrelations]

Returns an array of m^2 elements corresponding to the elements of the matrix, C, of weighted (Pearson) correlation coefficients for m series that are provided to the function as an array of mn elements, where the first n elements correspond to the terms in the first series, the next n elements to the corresponding terms in the second series etc.

Weighted correlation coefficients given weights w_i (for $i = 1, \dots, n$), are calculated as:

$$\tilde{c}_{j,k} = \frac{\tilde{v}_{j,k}}{\sqrt{\tilde{s}_j^2 \tilde{s}_k^2}}$$

where

$$\tilde{v}_{j,k} = \frac{(\sum w_i)(\sum w_i (x_{j,i} - \tilde{x}_j)(x_{k,i} - \tilde{x}_k))}{(\sum w_i)^2 - \sum w_i^2}$$
$$\tilde{s}_j = \sqrt{\frac{(\sum w_i)\left(\sum w_i (x_{j,i} - \tilde{x}_j)^2\right)}{(\sum w_i)^2 - \sum w_i^2}}$$
$$\tilde{x}_j = \frac{\sum w_i x_{j,i}}{\sum w_i}$$

See Weighted Moments and Cumulants for further details and explanation of terms in this formula. If the weights are equal then returns the same as MnCorrelations.

WeightedCovariance

[MnWeightedCovariance]

Returns the weighted (sample) covariance between two series.

Weighted (sample) covariances, given weights w_i (for $i = 1, \dots, n$), are calculated as:

$$\tilde{V}_{j,k} = \frac{(\sum w_i)(\sum w_i (x_i - \tilde{x})(y_i - \tilde{y}))}{(\sum w_i)^2 - \sum w_i^2}$$

where

$$\tilde{x} = \frac{\sum w_i x_i}{\sum w_i}$$
$$\tilde{y} = \frac{\sum w_i y_i}{\sum w_i}$$

See Weighted Moments and Cumulants for further details and explanation of terms in this formula. If the weights are equal then returns the same as MnCovariance.

WeightedCovariances

[MnWeightedCovariances]

Returns an array of m^2 elements corresponding to the elements of the matrix, $\tilde{V}$, of the weighted (sample) covariances for m series that are provided to the function as an array of mn elements, where the first n elements correspond to the terms in the first series, the next n elements to the corresponding terms in the second series etc.

Weighted (sample) covariances, given weights w_i (for $i = 1, \dots, n$), are calculated as:

$$\tilde{V}_{j,k} = \frac{(\sum w_i)\left(\sum w_i \left(x_{j,i} - \tilde{x}_j\right)\left(x_{k,i} - \tilde{x}_k\right)\right)}{(\sum w_i)^2 - \sum w_i^2}$$

where

$$\tilde{x}_j = \frac{\sum w_i x_{j,i}}{\sum w_i}$$

See Weighted Moments and Cumulants for further details and explanation of terms in this formula. If the weights are equal then returns the same as MnCovariances.

WeightedDesmooth_AR1

[MnWeightedDesmooth_AR1]

Returns an array corresponding to the AR(1) weighted de-smoothed (or 'de-correlated') values of a series, as described in the book Extreme Events. This involves:

(a) Postulating (for, say, a return series) that there is some underlying 'true' return series, $\tilde{r}_t$, and that the observed series, r_t, derives from it via a first order autoregressive model, $r_t = (1-\rho)\tilde{r}_t + \rho\tilde{r}_{t-1}$;

(b) Identifying the value of ρ (between 0 and 0.5) for which $\tilde{r}_t$ has zero weighted first order correlation (which implies that the weighted covariance between $\{r_1, \dots, r_{n-1}\}$ and $\{r_2, \dots, r_n\}$ is zero; and

(c) Using this value of ρ deriving $\tilde{r}_t$ using the following formula:

$$\tilde{r}_t = \begin{cases} r_t \quad \text{if } t = 1 \\ \dfrac{r_t - \rho\tilde{r}_{t-1}}{1-\rho} \quad \text{if } t > 1 \end{cases}$$

In step (b) we weight the elements in the covariance computation using weights $\{w_1, \dots, w_{n-1}\}$ (where w_1 corresponds to the weight given to the component arising from the term in $w_1 w_2$.

The value of ρ in (b) can be accessed using MnWeightedDesmooth_AR1_rho.

If the weights are equal then returns the same as MnDesmooth_AR1.

WeightedDesmooth_AR1_rho

[MnWeightedDesmooth_AR1_rho]

Returns the de-smoothing parameter corresponding to the AR(1) weighted de-smoothed (or 'de-correlated') values of a series, as described in the book Extreme Events. This involves:

(a) Postulating (for, say, a return series) that there is some underlying 'true' return series, $\tilde{r}_t$, and that the observed series, r_t, derives from it via a first order autoregressive model, $r_t = (1-\rho)\tilde{r}_t + \rho\tilde{r}_{t-1}$; and

(b) Identifying the value of ρ (between 0 and 0.5) for which $\tilde{r}_t$ has zero weighted first order correlation (which implies that the covariance between $\{r_1, \dots, r_{n-1}\}$ and $\{r_2, \dots, r_n\}$ is zero.

The de-smoothed series, $\tilde{r}_t$, can then be derived using the following formula, see MnWeightedDesmooth_AR1:

$$\tilde{r}_t = \begin{cases} r_t & \text{if } t = 1 \\ \dfrac{r_t - \rho\tilde{r}_{t-1}}{1-\rho} & \text{if } t > 1 \end{cases}$$

In step (b) we weight the elements in the covariance computation using weights $\{w_1, \dots, w_{n-1}\}$ (where w_1 corresponds to the weight given to the component arising from the term in $w_1 w_2$). If the weights are equal then returns the same as MnDesmooth_AR1_rho.

WeightedMean

[MnWeightedMean]

Returns $\tilde{x}$, the weighted mean of a series x_i given weights w_i (for $i = 1, \dots, n$), calculated as follows:

$$\tilde{x} = \frac{\sum w_i x_i}{\sum w_i}$$

There is no immediately equivalent function in Microsoft Excel, although it can be calculated relatively easily using SUMPRODUCT and SUM.

See Weighted Moments and Cumulants for further details and explanation of terms in this formula. If the weights are equal then returns the same as MnMean.

WeightedMeanAbsDevVsMean

[MnWeightedMeanAbsDevVsMean]

Returns, $\tilde{a}$, the weighted mean absolute deviation versus the mean of a series x_i (for $i = 1, \dots, n$), calculated as follows:

$$\tilde{a} = \frac{\sum w_i |x_i - \tilde{x}|}{\sum w_i}$$

(where $\tilde{x}$ is the weighted mean of the series)

See Weighted Moments and Cumulants for further details and explanation of terms in this formula. If the weights are equal then returns the same as MnMeanAbsDevVsMean.

WeightedMeanAbsDevVsMedian

[MnWeightedMeanAbsDevVsMedian]

Returns, $\tilde{b}$, the weighted mean absolute deviation versus the median of a series x_i (for $i = 1, \dots, n$), calculated as follows:

$$\tilde{b} = \frac{\sum w_i |x_i - \tilde{x}_{med}|}{\sum w_i}$$

(where $\tilde{x}_{med}$ is the weighted median of the series, see MnWeightedMedian)

See also MnWeightedMeanAbsDevVsMean which provides another common way of measuring mean absolute deviation. However, the mean absolute deviation is minimised if it is measured relative to the median of the series rather than relative to its mean.

See Weighted Moments and Cumulants for further details and explanation of terms in this formula. If the weights are equal then returns the same as MnMeanAbsDevVsMedian.

WeightedMedian

[MnWeightedMedian]

Returns $\tilde{x}_{med}$, the weighted median of a series of observations series x_i given weights w_i (for $i = 1, \dots, n$), i.e. the same as the value of $MnWtdPercentile(x, w, 0.5)$.

If the weights are equal then returns the same as MnMedian.

WeightedPercentile

[MnWeightedPercentile]

Returns $\tilde{J}_m$, the m'th weighted percentile (m here being between 0 and 1) of a series of observations series x_i given weights w_i (for $i = 1, \dots, n$).

Different commentators use different ways of identifying percentiles (i.e. quantiles) from finite sized samples. See MnPercentile and MnPercentileExc for details of how the Nematrian website calculates equally-weighted percentiles, which match the approaches used by Microsoft.

For *weighted* percentiles, the Nematrian website uses the following approach, which is not fully compatible with the approach it uses for unweighted percentiles:

- First sort the series in ascending order
- Next build up a cumulative distribution based on the weights, and assume that the percentile points for the mid-points of each bin are the x_i in question (so, if the weight of the first in the ordered list is $w_{(1)}$ then the $w_{(1)}/\sum w_i$'th percentile is deemed to be exactly $x_{(1)}$, the $\left(w_{(1)} + w_{(2)}\right)/\sum w_i$'th percentile is deemed to be exactly $x_{(2)}$, etc.)
- Finally use linear interpolation as per the MnLinearInterpolation function to interpolate remaining percentiles.

The MnLinearInterpolation function does not extrapolate beyond the minimum and maximum values, so the 0'th and 1'th quantiles should be the minimum and maximum values of the series, and hence be consistent with what would be returned for these quantile values by the MnPercentile function. However, intermediate values with the w_i all equal will not correspond exactly to those that would be returned by the MnPercentile function.

WeightedPopulationCovariance

[MnWeightedPopulationCovariance]

Returns the weighted (population) covariance between two series.

Weighted population covariances given weights w_i (for $i = 1, \dots, n$), are calculated as:

$$\tilde{V}_{p;j,k} = \frac{\left(\sum w_i (x_i - \tilde{x})(y_i - \tilde{y})\right)}{\sum w_i}$$

See Weighted Moments and Cumulants for further details and explanation of terms in this formula. If the weights are equal then returns the same as MnPopulationCovariance.

WeightedPopulationCovariances

[MnWeightedPopulationCovariances]

Returns an array of m^2 elements corresponding to the elements of the matrix, $\tilde{V}_p$, of the weighted (population) covariances for m series that are provided to the function as an array of mn elements, where the first n elements correspond to the terms in the first series, the next n elements to the corresponding terms in the second series etc.

Weighted population covariances, given weights w_i (for $i = 1, \dots, n$), are calculated as:

$$\tilde{V}_{p;j,k} = \frac{\left(\sum w_i \left(x_{j,i} - \tilde{x}_j\right)\left(x_{k,i} - \tilde{x}_k\right)\right)}{\sum w_i}$$

See Weighted Moments and Cumulants for further details and explanation of terms in this formula. If the weights are equal then returns the same as MnPopulationCovariances.

WeightedPopulationKurt

[MnWeightedPopulationKurt]

Returns $\tilde{\gamma}_{2,p}$, the weighted population (excess) kurtosis of a series x_i given weights w_i (for $i = 1, \dots, n$), calculated as follows:

$$\tilde{\gamma}_{2,p} = \frac{\sum w_i \left(\frac{x_i - \tilde{x}}{\tilde{s}_p}\right)^4}{\sum w_i} - 3$$

See Weighted Moments and Cumulants for further details and explanation of terms in this formula. If the weights are equal then returns the same as MnPopulationKurt.

WeightedPopulationSkew

[MnWeightedPopulationSkew]

Returns $\tilde{\gamma}_{1,p}$, the weighted population skew of a series x_i given weights w_i (for $i = 1, \dots, n$), calculated as follows:

$$\tilde{\gamma}_{1,p} = \frac{\sum w_i \left(\frac{x_i - \tilde{x}}{\tilde{s}_p}\right)^3}{\sum w_i}$$

See Weighted Moments and Cumulants for further details and explanation of terms in this formula. If the weights are equal then returns the same as MnPopulationSkew.

WeightedPopulationStdev

[MnWeightedPopulationStdev]

Returns $\tilde{s}_p$, the weighted population standard deviation of a series x_i given weights w_i (for $i = 1, \dots, n$), calculated as follows:

$$\tilde{s}_p = \sqrt{\frac{\sum w_i (x_i - \tilde{x})^2}{\sum w_i}}$$

See Weighted Moments and Cumulants for further details and explanation of terms in this formula. If the weights are equal then returns the same as MnPopulationStdev.

WeightedPopulationVariance

[MnWeightedPopulationVariance]

Returns $\tilde{v}_p$, the weighted population variance of a series x_i given weights w_i (for $i = 1, \dots, n$), calculated as follows:

$$\tilde{v}_p = \frac{\sum w_i (x_i - \tilde{x})^2}{\sum w_i}$$

See Weighted Moments and Cumulants for further details and explanation of terms in this formula. If the weights are equal then returns the same as MnPopulationVariance.

WeightedSkew

[MnWeightedSkew]

Returns $\tilde{\gamma}_1$, the weighted (sample) skew of a series x_i given weights w_i (for $i = 1, \dots, n$), calculated as follows:

$$\tilde{\gamma}_1 = \frac{(\sum w_i)^2 \left(\sum w_i \left(\frac{x_i - \tilde{x}}{\tilde{s}} \right)^3 \right)}{(\sum w_i)^3 - 3(\sum w_i^2)(\sum w_i) + 2(\sum w_i^3)}$$

See Weighted Moments and Cumulants for further details and explanation of terms in this formula. If the weights are equal then returns the same as MnSkew.

WeightedSpearmanRankCorrelation

[MnWeightedSpearmanRankCorrelation]

Returns the weighted Spearman correlation between two series.

The weighted Spearman rank correlation for two series, 1 and 2, given weights w_i (for $i = 1, \dots, n$), is calculated as:

$$\tilde{c}_{1,2} = \frac{\tilde{v}_{1,2}}{\sqrt{\tilde{s}_1^2 \tilde{s}_2^2}}$$

where

$$\tilde{v}_{j,k} = \frac{(\sum w_i)(\sum w_i (q_{j,i} - \tilde{q}_j)(q_{k,i} - \tilde{q}_k))}{(\sum w_i)^2 - \sum w_i^2}$$

$$\tilde{s}_j = \sqrt{\frac{(\sum w_i)\left(\sum w_i (q_{j,i} - \tilde{q}_j)^2\right)}{(\sum w_i)^2 - \sum w_i^2}}$$

$$\tilde{q}_j = \frac{\sum w_i q_{j,i}}{\sum w_i}$$

and $q_{i,t}$ are the ranks of the values in the series $x_{i,t}$, i.e. $q_{i,1}$ is the largest value in the series $\{x_{i,t} : t = 1, \dots, n\}$

See Weighted Moments and Cumulants for further details and explanation of terms in this formula. If the weights are equal then returns the same as MnSpearmanRankCorrelation.

WeightedStdev

[MnWeightedStdev]

Returns $\tilde{s}$, the weighted (sample) standard deviation of a series x_i given weights w_i (for $i = 1, \dots, n$), calculated as follows:

$$\tilde{s} = \sqrt{\frac{(\sum w_i)(\sum w_i (x_i - \tilde{x})^2)}{(\sum w_i)^2 - \sum w_i^2}}$$

See Weighted Moments and Cumulants for further details and explanation of terms in this formula. If the weights are equal then returns the same as MnStdev.

WeightedVariance

[MnWeightedVariance]

Returns $\tilde{v}$, the weighted (sample) variance of a series x_i given weights w_i (for $i = 1, \dots, n$), calculated as follows:

$$\tilde{v} = \frac{(\sum w_i)(\sum w_i (x_i - \tilde{x})^2)}{(\sum w_i)^2 - \sum w_i^2}$$

See Weighted Moments and Cumulants for further details and explanation of terms in this formula. If the weights are equal then returns the same as MnVariance.

WeightOverlap

[MnWeightOverlap]

Returns the weight overlap between two (long-only) portfolios, P and B, i.e.:

$$WO(P,B) = \sum \min\left(|w_{i,P}|, |w_{i,B}|\right)$$

Where the weights in the portfolio (rescaled to add to unity) and in the benchmark (also rescaled to add to unity) are $w_{i,P}$ and $w_{i,B}$ respectively.

Year

[MnYear]

Returns the year of a given date/time

APPENDIX D: USING THE NEMATRIAN FUNCTION LIBRARY PROGRAMMATICALLY

Download Illustration Spreadsheet

[WebServiceExampleSpreadsheets]

To download a spreadsheet that illustrates a range of functions then please press the relevant button. Those with underlying macros are typically set up so that when users download the spreadsheet from the website they have immediate access to the online function library (for suitably chosen browsers) within the spreadsheet. Those without macros typically illustrate how to access the same set of functions using just the WEBSERVICE() function within Excel. The xla contains just the underlying macros.

To access spreadsheets that illustrate other functions go to the relevant function page or to the relevant example spreadsheet page. To view the underlying VBA code used to access the Nematrian function library see here.

The Nematrian spreadsheet library

[IntroductionSpreadsheets]

Nematrian makes available under its Nematrian Extensions brand a wide range of accurate, efficient and well-documented online analytical tools. You can access these tools in many different ways, see here. Most have been packaged into a series of spreadsheets that illustrate how the functions involved can be accessed in a manner akin to built-in spreadsheet functions. This page provides links to these illustrative spreadsheets, each of which illustrates functions that target specific types of analysis:

Spreadsheet	**Illustrates**
Example	A simple example of how to access the Nematrian function library via macros
Probability distributions	Probability distribution functions including moments, probability density and mass functions (pdfs and pmfs), cumulative distribution functions (cdfs), inverse cdfs (i.e. quantile functions), random variate generation and parameter estimation tools for a wide range of probability distributions
Probability distributions (individual)	Additional probability distribution functions for some individual distribution
Tail weighted probability distributions	Functions for fitting probability distributions to particular regions of the distributional form (typically their tails)
Calculator functions	Common (non-trigonometric) functions available on a straightforward electronic calculator as well as a range of matrix, statistical and return data manipulation functions
Trigonometric functions	Trigonometric functions for both real valued and complex valued inputs
Portfolio optimisation	Portfolio optimisation and reverse optimisation functions primarily focusing on mean-variance optimisation
Black-Scholes option pricing	Black-Scholes option pricing functions, including option greeks, for vanilla and binary put and call options

Tri-segmented Monte Carlo	Functions demonstrating small-scale tri-segmented Monte Carlo simulation
Solvency II	Functions designed to facilitate calculation of a Solvency II standard formula SCR
Set manipulations	Functions for manipulating sets
Array and series manipulations	Functions for manipulating arrays and (time) series
IP (Internet Protocol) functions	Functions relating to IP access and web services
PCA and variants	Functions for carrying out Principal Components Analysis and related activities
Risk management functions	Functions for carrying out a range of risk management computations
Statistical tools	Statistical tools (not included in other example function library spreadsheets)
Data Analysis equivalent	Functions that emulate some of the analyses available via Excel's Data Analysis Toolpack
Sudoku Solver	Functions for carrying out automated solving of Sudoku puzzles
Physical unit conversions	Functions for carrying out conversions between values expressed in different physical units
Plotting and other SmartUtility related functions	Functions that streamline creation of some types of more complicated charts or other SmartUtilities
Peer group performance measurement functions	Functions for manipulating performance measurement of peer groups
Miscellaneous life insurance functions	Functions for carrying out specific life insurance computations
Resampled portfolio optimisation	Functions for carrying out resampled (bootstrapped) portfolio optimisation
Pension fund projection functions	Functions for facilitating pension fund projections (including sponsor covenant analysis)
Optimisation (generic)	Functions for carrying out generic optimisation problems
String orientated	Functions primarily involving strings or string manipulations
Date orientated	Functions primarily involving dates or date manipulation
(Basic) finance	Functions mimicking some of the finance functions typically found in spreadsheet systems, e.g. IRR, NPV, DDB
Expression evaluation	Functions to help with evaluating expressions
Markov processes	Functions for simulating and analysing Markov processes
Matrix	Functions for carrying out operations on matrices

In each case several variants are provided which are designed to work with different spreadsheet systems. Each spreadsheet aims to provide access to the complete suite of publicly functions rather than just those specifically being illustrated by the spreadsheet.

Help accessing Nematrian Web Services through Microsoft Excel: Frequently Asked Questions (FAQ)

[WebServiceAccessFAQ]

If you experience trouble accessing Nematrian web functions and the following questions and answers do not help then please contact us so that we can answer your query and maximise the benefit others can get from our site.

Question: What do I do if I'm trying to access Nematrian web functions using Microsoft Excel spreadsheets and I see lots of ?NAME and don't seem to be able to get any of the macros to work?

Answer: Try enabling the macros within the spreadsheet. By default Microsoft Excel often opens spreadsheets containing macros with them disabled. Alternatively, use a spreadsheet without macros and load in the equivalent Nematrian add-in, as the macros in these will then be enabled.

Question: What do I do if I'm trying to access Nematrian web functions using Microsoft Excel spreadsheets that I download from the Nematrian website and the macro falls over at a statement involving something like 'Dim xmlhtp As New XMLHTTP60' (usually commented out) or 'Dim xmlhp As Object' and 'Set xmlhtp = CreateObject("MSXML2.XMLHTTP")' (usually now the default)?

Answer: These few lines form the only place in the spreadsheet VBA module that refers to software that is not specifically part of Excel and therefore may not be loaded when Excel first starts. Specifically, the two lines now included as default assume that the Excel program on your PC has access to a Microsoft XML library. The module will then load whatever your PC views as the default version of this library. If your PC doesn't recognise these statements then you may need to include a manual reference to, say, version 6 of the Microsoft XML library (using Tools>References in the VBA editor) and then replace the two lines beginning 'Dim xmlhp As Object' with a line like 'Dim xmlhtp As New XMLHTTP60'. However, Excel may not be able to understand what "XMLHTTP60" refers to depending on the XML library you may have access to. If a reference to an older XML library is available then you could try disabling that reference and enabling a newer version of the XML library or try replacing XMLHTTP60 with any one of XMLHTTP, XMLHTTP30, XMLHTTP40, … that does successfully compile. If there are no XML libraries available then check that you have installed all of Microsoft Excel perhaps after searching through the object browser. Different legal versions of Excel generally do come with at least one XML library version.

Question: What do I do if I am trying to access Nematrian web functions using Microsoft Excel spreadsheets and I get a lot of #VALUE errors appearing that I do not expect?

Answer: Functions can return #VALUE for a variety of reasons. Often this is because the spreadsheet has not recalculated itself when first loaded. You can trick the spreadsheet into doing so by deleting a row (e.g. the row after the last row in which any cells are non-blank. Sometimes the existence of #VALUE entries reflect the computations being carried out. For example the fractional power of a negative number is not well defined, so the relevant Nematrian web function that returns the result of such a calculation returns such a value. At other times, it may be because you have entered the wrong number or the wrong sorts of parameters. Sometimes the Nematrian website or links to it may not be working properly. If you set "show detailed error messages" to TRUE in the standard configuration of example spreadsheets then the spreadsheet should provide an error message each time it comes across an error. However, be warned! If your spreadsheet is showing #VALUE in lots of places then there may be lots of error messages that the spreadsheet throws back to you.

Question: What do I do if I am trying to access Nematrian web functions using Microsoft Excel spreadsheets and I get a lot of #VALUE errors appearing that I do not expect even after sorting out all errors as above?

Answer: Excel spreadsheets sometimes recalculate themselves when first opened and sometimes do not. Check that the option recalculate is not set to manual or if it is then recalculate the spreadsheet manually. If there are still #VALUE errors appearing try recalculating individual formulae by hand. For example you could click on the cell, press F2 (to begin as if to edit the formula) but then immediately ENTER so that you do not actually amend the formula in the cell but it is recalculated. Pease bear in mind that some Nematrian web functions produce array output spanning several cells, so in these cases you need to edit/recalculate the entire output range all at once.

Question: How do I access Nematrian web functions using other spreadsheet packages such as OpenOffice or Apple or Google products?

The Nematrian Web Service Code Generator page contains details of how to access Nematrian web functions through OpenOffice. Individual functions can be accessed through Google Drive, see spreadsheet downloads available from the relevant function page. At the time of writing spreadsheets typically bundled with Apple products did not support a WEBSERVICE function or the equivalent making it more challenging to access the Nematrian web functions through Apple products except via a browser.

Accessing the Nematrian website's web services in Microsoft Excel (and other Microsoft Office components) using VBA

[WebServiceAccessVBA]

Using a spreadsheet which already has in-built access to these web services

Calling Nematrian web functions from within Microsoft Excel or equivalents from other providers is very easy to do if you already have access to a spreadsheet that has been set up to give you this access. The spreadsheet library include the facility to download spreadsheets that illustrate how to access many of the functions in the Nematrian function library as do the pages describing individual web functions. These downloadable spreadsheets generally have embedded within them the entire Nematrian function library.

Building a spreadsheet from scratch: Part I

It is also not difficult to build such a spreadsheet yourself. If you can get the example VBA code set out below to work then you should be able to use any Nematrian web service without too much difficulty.

[See webpage]

If you are comfortable that you understand this code and that making use of it will not compromise you or your computer then you could try to insert the above code into a blank VBA module in a blank Excel spreadsheet.

To do this, you would first open Excel and create a new (blank) spreadsheet. Then you would open VBA within Excel by, for example, pressing the ALT and the F11 keys down simultaneously. A VBA Project referring to the worksheet should appear somewhere in the list of VBA Projects on the left hand side, but it will probably not have any VBA modules within it. So, you would first insert a blank module into the Project that seems to be linked to your blank worksheet (using Insert - Module *not* Insert – Class Module) and then copy and paste this code into this module.

You also need to include a reference to a Microsoft XML library. You can do this by using the Tools - References facility within VBA. This facility typically opens up a long list of possible references that you could include, and checkboxes alongside ones you might include. You would need to tick one called something like *Microsoft XML, v6.0*, although earlier (and possibly later) versions also seem to work fine (although you may then have to replace the "60" with some other number at the end of the lines involving *Dim xmlhtp As New XMLHTTP60* and *Dim xmlDoc As New DOMDocument60*.

Next try running the TestWebService macro shown above. Several possible things might happen, e.g.:

(a) Excel/VBA may refuse to allow macros to run. This means that the macro functionality within Excel is *disabled*. It will need to be *enabled* before you can make use of the web service. See the Excel Help facility to work out how to do this.

(b) If macros are enabled then Excel/VBA may show a *user-defined type not defined* compile error and simultaneously highlight the text *xmlhtp As New XMLHTTP60* and/or *xmlDoc As New DOMDocument60*. This indicates that you haven't included a reference as above to one of the Microsoft XML libraries.

(c) If you have enabled macros and added a reference to one of the Microsoft XML libraries then the macro should run. Assuming the website is working properly, you should get a message box popping up showing 1+1 = 2 and 'a' & 'b' = 'ab'. Well done, you have called the Nematrian website using a 'complex' web service from within Microsoft Excel! All other Nematrian web services can be accessed in a similar type of manner, except that they generally also require you to provide a valid SessionId.

Building a spreadsheet from scratch: Part II

Using more complicated Nematrian web services from within Microsoft Excel is plain sailing once you have got to the end of Part I, as long as you use the Nematrian Web Service Example Code Generator. This creates all the VBA code you would need to access any available Nematrian web function. Even easier, of course, is to download one of the example spreadsheets that already has embedded within it the relevant elements to allow it to access the Nematrian online web function library.

To use most Nematrian web functions you will need to obtain a Session Id. You can either get one by logging in to the website or you can obtain a temporary *SessionId* from the menu at the top of the website.

More advanced programmers wanting to understand the structure of what is returned from the Nematrian website when you call a Nematrian web function may find this link helpful.

Using other types of spreadsheet

It is not difficult to create equivalent spreadsheets in, say, OpenOffice as long as you alter the macros that they call so that they send http requests to the Nematrian website and receive them back in a manner understood by OpenOffice. See individual web function description pages for more details.

Warning

Users should not insert software generated by the website into their own programs or spreadsheets unless they understand what it does and are sure that it will not damage or corrupt any of their existing work.

The Nematrian website takes no responsibility for losses arising from use of code sourced from this page or the from the web service example code generator facility.

Please bear in mind that it is possible that hackers may tamper with the Nematrian website and introduce malicious code in the hope that you will run this code assuming that it will work as you expect. If you have any doubts about whether the code you see on this page may be malicious then please refer to an expert.

Use of the Nematrian website, including its web services, is subject to the terms of the Nematrian Web Services Software License Agreement. By making use of the site, including its web services functionality, you agree to be bound by this Agreement.

Nematrian Web Service VBA Code Generator and Example Excel spreadsheet

[WebServiceCodeGenerator]

To make it easy for you to use Nematrian web functions, especially in Excel, we have included within the website:

(a) Examples of spreadsheets that are already pre-populated with relevant macros and with examples of how to access Nematrian web functions using Excel cell formulae. These examples work in a manner akin to Microsoft built-in functions such as SIN, PERCENTILE or STDEV). See web pages describing individual functions for further details.

(b) An automated code generator that produces the VBA code used in (a) which you can insert into any standalone module in Microsoft Excel or other Microsoft Office components. Please remember to include a reference to a *Microsoft XML library* in the module in which you insert this VBA code. Please refer to how to access the web services through VBA for more details.

[See webpage]

Warning

Users should not insert software or enable macros in spreadsheets generated by the website into their own programs or spreadsheets unless they understand what the software does and are sure that it will not damage or corrupt any of their existing work.

The Nematrian website takes no responsibility for losses arising from use of code sourced from the website.

Please bear in mind that it is possible that hackers may tamper with the Nematrian website and may introduce malicious code in the hope that you will run this code assuming that it will work as you expect. If you have any doubts about whether any automatically generated code contains such material then please do not use it.

Use of the Nematrian website and software obtained from it is subject to the terms and conditions of the Nematrian License Agreement.

Nematrian Web Service Function Answer Structures

[WebServiceAnswerStructures]

More advanced programmers may wish to note the format of results returned by Nematrian web service functions.

Nearly all Nematrian web service functions require a live Session Id to run. These return their answers to the user in a variety of bespoke formats which have names such as *MnDouble*, *MnDoubleArray* etc.

Each of these bespoke formats consists of three parts:

(a) ErrFlag (a *String* variable): indicates any errors that may have occurred

(b) Debit (a *Double* variable): indicates amount of credit deducted from your SessionId when the web function is run

(c) Answer: provides the answer

The structure taken by the "Answer" element varies according to the purpose of the function and explains the range of formats actually used. It is shown on the relevant function page. For example, if "Answer" is a *Double* variable, then the name of the overall structure is MnDouble. If "Answer" is a *Double* array ("Double()") then the name of the overall structure is MnDoubleArray.

Some of the more complicated web functions also include a "Sizes" array to allow them to return multi-dimensional answers.

Using Nematrian Web Functions in more sophisticated programming environments such as Visual Studio

[Nematrian website page: WebServiceAccess, © Nematrian 2015]

As explained in Web Services Introductory Pages, the Nematrian website provides a wide range of web services that users can access directly via the Internet. To do this, the user needs to include calls to the relevant web services within some suitable software container, e.g. in programs written in VBA code within Excel (or another Microsoft Office application), see here, directly in a web page in HTML and Javascript or in software written using Visual Basic or C++ code, see below.

In what follows we assume that users are trying to access Nematrian web functions using via Visual Basic within Visual Studio although many other Software Development Kits (SDK) operate in a relatively similar fashion.

We suggest that you first try accessing Nematrian web services by inserting into your browser the following URL: http://www.nematrian.com/WebServices.asmx

If this comes up with a long list of functions then you should be able to access Nematrian's web service functions using this URL or equivalent URL's for other Nematrian public servers. Almost all of the web services accessible via this .asmx page require you to have a live Nematrian site Session ID, with the exception of 'Complex Sum'.

Almost all Nematrian web service functions return back answers using a bespoke structure that includes an error flag, an indication of the amount of credit deducted from your Session Id when you ran the function and the result of running the function (if the error flag was not triggered). This is to make the functions as versatile as possible. The elements in each structure are called .ErrFlag, .Debit and .Answer respectively. .ErrFlag is a *String* variable and .Debit is a *Double* variable. The structure taken by .Answer varies according to the purpose of the function and can be found from the relevant function description page.

To access the software within your code you first need to add a *reference* to the relevant software (e.g. by using the *Add a web reference* command in the "Website" drop-down menu). The dialogue box opened via this Visual Studio menu command allows you to browse for the URL for the web service you are attempting to access. To avoid excessive usage of a single URL, the Nematrian web site may distribute its delivery of web services across several parallel web sites (i.e. several URLs). Where practical, it will endeavour to assign you the same URL whenever you log in (to avoid you needing to recompile your code to refer to a new reference URL). The dialogue box allows you to name the Nematrian web service anything you like; it is assumed below that you have called it *mnWebService*.

Software along the lines of the following will then, for example, call the Nematrian Sudoku Solver (it is assumed here that you are using Visual Basic as the programming language within Visual Studio):

```
Option Explicit On
Imports System.Web.Services.Protocols

Private Function SudokuSolve( _
ByVal SessionID As String, _
ByVal SudokuType As String, _
ByVal BaseStartingData() As Byte, _
ByVal IterationLimit As Integer) As mnWebService.mnSudokuSolution

Dim ws As New mnWebService.WebServices
SudokuSolve = ws.SudokuSolve(SessionID, SudokuType, BaseStartingData, IterationLimit)
End Function
```

The input parameters for this function (and the corresponding web service) are:

Name	**Type**	**Description of Variable**
SessionID	String	User's SessionID as obtained by logging into the Nematrian website
SudokuType	String	Type of Sudoku puzzle being solved, "Standard", "Samurai", "Killer" or "General"
BaseStartingData()	Byte	Unidimensional byte array that contains the base starting data
IterationLimit	Integer	a large number (e.g. 100000) limiting no of iterations Solver will execute

The output of this function has the following structure

```
Structure mnSudokuSolution
Dim ErrFlag As String
Dim Xcoord() As Byte
Dim Ycoord() As Byte
Dim SolutionValue() As Byte
Dim TimeTaken As Double
Dim NoTimesAccessed As Integer
End Structure
```

So, for example, we can tell whether the Sudoku puzzle has a (unique) solution by using code such as:

```
Dim SudokuSoln As mnWebService.mnSudokuSolution
Dim ErrFlag As String
SudokuSoln = SudokuSolve(SessionID, SudokuType, BaseStartingData, IterationLimit)
If SudokuSoln.ErrFlag<>"" Then
...
End If
```

Disclaimer