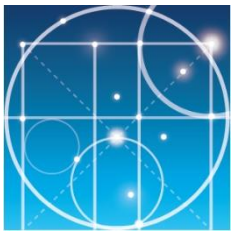
PrimaXL V2017
User's Reference Manual



FIAN RESEARCH

January, 2017

This page was left intentionally blank

Preface and Legal Notice

This is PrimaXL User's Reference Manual. Please, refer to the Quick Start Guide for information on the system requirement and the installation of the software.

PrimaXL is an add-in software for Microsoft Excel. PrimaXL is a trademark of FIAN Research. Microsoft, Windows, Excel, and the Office logo are trademarks or registered trademarks of Microsoft Corporation in the United States and other countries.

The information contained herein is subject to change without notice and is not warranted to be free of errors. If you find any errors, please report to us in writing.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except when expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Decompilation, disassembly, or reverse engineering of this software is prohibited.

This software is intended for general use in information management applications. It is not intended for use in any inherently risky or potentially harmful applications. If you intend to use this software in risky or potentially harmful applications, then you shall take all the appropriate precautionary measures to ensure the safe use of this software. In no event will FIAN Research or its suppliers be liable to the licensed user for any special, incidental or consequential damages in connection with the software. Nor will FIAN Research or its suppliers be liable for the lost revenue or profit in connection with the software product.

Your access to and use of this material is subject to the terms and conditions of the End User License Agreement (EULA) with which you agree to comply.

This manual is not distributed under a GPL license. You may create a printed copy of this manual solely for your own personal use. You shall not publish or distribute this manual in any form or on any media without the prior written consent from FIAN Research. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way.

For more information, please visit www.fianresearch.com.

Contents

DataTransformTest

DetrendPoly

FitBezier

FitPoly

FitPolyCoeff

FitPolyPt

InterpolateLinear

InterpolateLinearPt

InterpolatePoly

InterpolatePolyPt

InterpolateSpline

InterpolateSplinePt

Kalman

LogisticCoeff

LogisticCoeffOneVsAll

LogisticConfMatrix

LogisticConfMatrixMulti

LogisticForecast

LogisticForecastMulti

LogisticROC

MissingDataFill

MissingDataMask

NetworkCommunityStruct

NetworkCommunityStructAdj

NetworkModularity

NetworkModularityAdj

OLSANOVA

TableQQSTTPlot

TSACF

TSACFAR

TSACFARMA

TSACFMA

TSACFTest

TSARCharRoots

TSARCHFit

TSARCHForecast

TSARCHSimul

TSARCHVol

TSARCHVol2

TSARIMAFit

TSARMAForecast

TSARMASimul

TSARSimul

TSARStat

TSConvARMAtoMA

TSConvARtoMA

TSConvMAtoAR

TSDickeyFuller

TSDickeyFullerAugmented

TSDifference

TSEngleGranger

TSEngleGrangerSpread

TSEngleGrangerSpreadForecast

TSGARCHFit

TSGARCHForecast

TSGARCHSimul

TSGARCHVol

TSGARCHVol2

TSHoltFit

TSHoltForecast

TSHoltSmooth

TSIsInvertibleMA

TSIsStationaryAR

TSLjungBox

TSLogRate

TSShowLag

TSVARFit

TSVARForecast

TSVARSimul

TSVECMFit

TSVECMForecast

TSVECMSimul

Getting Help

Introductory Remarks

This is the Reference Manual of PrimaXL add-in. Add-in is a software that adds new features to Microsoft Excel.

Microsoft Excel is a spreadsheet application that features calculations, graphing tools, pivot tables, and a macro programming language called Visual Basic for Applications (VBA). Excel is without doubt a very powerful and versatile tool for capturing, processing, manipulating and visualizing of data. Excel is chosen by millions of users in business, academia, as well as by enthusiasts as their main platform for data related tasks.

PrimaXL was developed with the purpose of extending the capabilities of the standard Excel in the data mining and time series[1] analysis.

PrimaXL was developed for use by beginners as well as experts. The easiest way to execute the functions is through the intuitive ribbon menu. If necessary, the user may directly type the functions in the spreadsheet cells or may invoke the functions from a VBA script.

In this reference manual all the functions of PrimaXL are listed and their usage explained in details. In order to incorporate the functions in the applications, it is important to follow the syntax and interpret correctly the formatted output.

In the coming sections, we will cover the details of working with the PrimaXL functions.

[1] A representative example of the time series is that of the financial asset prices and returns.

1. Executing Functions

The PrimaXL functions can be applied in three different ways.

1) **From the ribbon menu**: This is the easiest way of executing the PrimaXL functions. Select the desired task and complete the blank fields in the menu form. In most cases, the input arguments of a function must follow a well-defined format. For example, the range may need to be a single column array in order to represent a univariate time series. The ribbon menu calls a COM routine that runs the basic syntax check before calling the actual functions. The actual functions reside within the XLL module.

2) **By typing the function manually into the spreadsheet cell(s)**: In this case, the user should strictly follow the required format of the input arguments. Incorrect arguments will return erroneous results, "#NUM!" error message or just value 0. Also, it is very important to remember that in many cases the returned value of the function is not a scalar but a vector or a matrix. Thus, the function must be executed as a *multi-cell array formula*: after selecting the output range of appropriate size, the combination **CTRL+SHIFT+ENTER** should be pressed.

3) **Call from a VBA script**: Use the VBA method Application.Run. In this way, it is possible to execute directly a function of the XLL module.

1.1 Executing the Menu Selection

Let us illustrate the use of PrimaXL with an example. Let's suppose that we have daily prices of the company A's stock. This time series is ordered ascendingly, such that the most recent observation comes at the top.

We can notice that this is likely a non-stationary time series. Thus, we'd like to apply Holt's exponential smoothing method and forecast the future. The function that does the actual task of forecasting is TSHoltForecast. However, you don't need to remember it: just go to **Smoothing → Exponential Smoothing Forecast** from the ribbon menu as shown in the Figure 1.1 below.

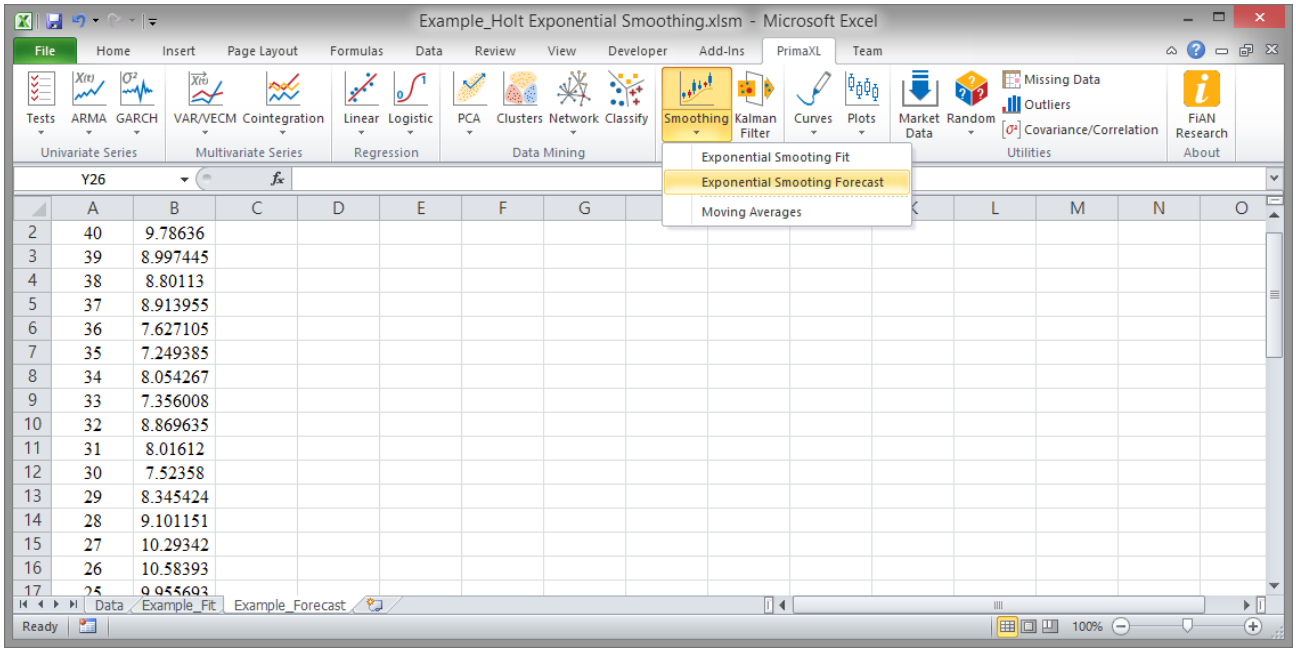


Figure 1.1

Then, a menu form as shown in the Figure 1.2 will appear. Please, notice that within the menu form there are two groups: **Input** and **Output**.

First, you have to complete the **Input** group. At the topmost field you need to enter the range corresponding to a time series data. You can either type directly in the field or press the small button on the right side to invoke the range input form. This last method is easier and preferable way of entering error free range values.

In the **Order** subgroup two choices are available: *ascending* or *descending*. You have to specify whether the ordering of the time series is top-down (descending) or otherwise (ascending).

Next, you can proceed to the **Output** group. Here, you have to specify the location of the output (within an existing worksheet) or check the option *Output to a new sheet* in order to create a new worksheet and stream the output to it. Then, you can specify the number of the forecast steps. You can also calculate[2] the confidence interval by selecting the appropriate *Interval* option. Finally, you can select a *Plot* option to display the forecast with a variable length of the past steps.

[2] By Monte Carlo simulation method.

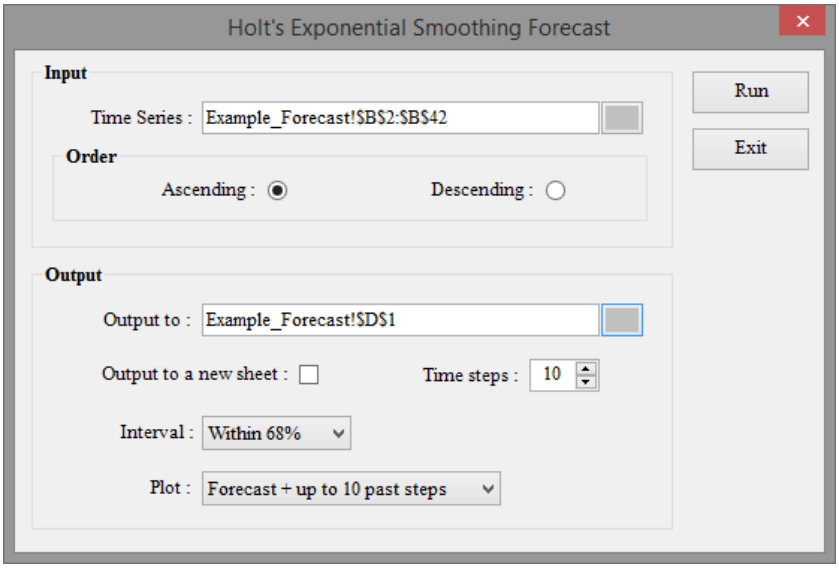


Figure 1.2

When you are ready[3], press the **RUN** button to execute the function in the spreadsheet. The result is shown in the following Figure 1.3 and Figure 1.4. Please, notice that the first row of the output shows the title "**HOLT'S EXPONENTIAL SMOOTHING FORECAST**". Then, the second row shows the labels such as "*Time Step*", "*Low*", "*Middle*" and *"High"*. We can see that the time step is numbered from 0 through 10. Here, the row of time step 0 just repeats the most recent data: *Cell E3 = Cell F3 = Cell G3 = Cell B2*. We can notice that the title and the labels are not part of the function result[4]. In the plot (Figure 1.4) the blue and red dashed lines define the confidence interval.

O25

	A	B	C	D	E	F	G	H	I	J	K	L
1	**Time Step**	**Value**		**HOLT'S EXPONENTIAL SMOOTHING FORECAST (68%)**								
2	40	9.78636		*Time Step*	*Low*	*Middle*	*High*		*Time Step*	*Value*	*Mov. Avg.*	
3	39	8.997445		0	9.78636	9.78636	9.78636		-9	8.01612	7.361454	
4	38	8.80113		1	9.138287	9.927835	10.74016		-8	8.869635	7.924289	
5	37	8.913955		2	8.880414	10.0702	11.27097		-7	7.356008	8.881433	
6	36	7.627105		3	8.666806	10.21257	11.74659		-6	8.054267	7.203052	
7	35	7.249385		4	8.459875	10.35494	12.22183		-5	7.249385	7.992019	
8	34	8.054267		5	8.271828	10.49731	12.69639		-4	7.627105	7.10741	
9	33	7.356008		6	8.101584	10.63968	13.17635		-3	8.913955	7.540741	
10	32	8.869635		7	7.933695	10.78205	13.64404		-2	8.80113	8.977593	
11	31	8.01612		8	7.730641	10.92442	14.10964		-1	8.997445	8.847264	
12	30	7.52358		9	7.522631	11.06678	14.60412		0	9.78636	9.059697	
13	29	8.345424		10	7.351423	11.20915	15.11377					
14	28	9.101151										
15	27	10.29342										

Figure 1.3

[3] Here, we assume that the menu form was completed as Figure 1.2.

[4] The title and the labels are displayed outside the function output cells.

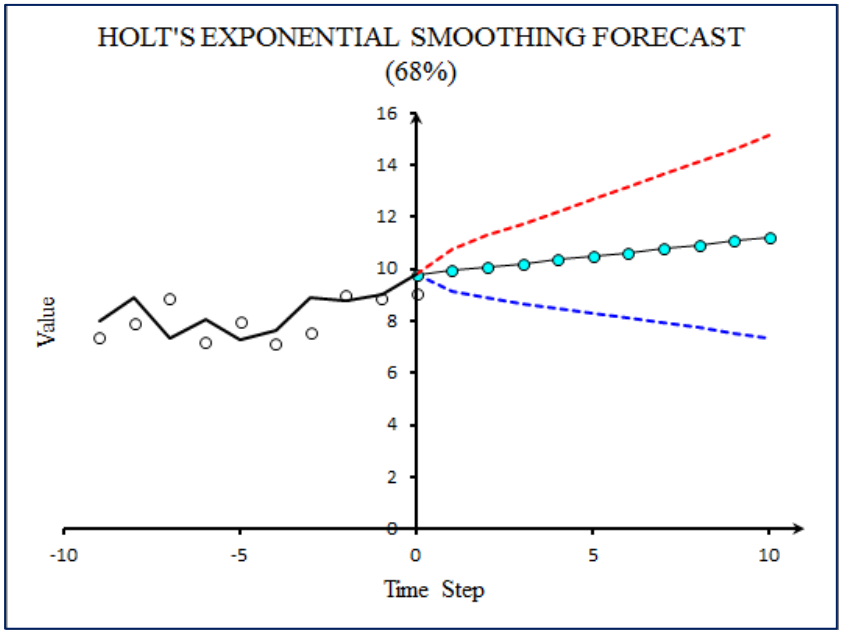


Figure 1.4

In fact, when you select with mouse the spreadsheet cell range as in the Figure 1.5, the function name with its arguments appears explicitly in the formula bar. You should notice that the entire formula is enclosed by a pair of matching curly brackets {} indicating that this is a *multi-cell array formula*.

E3 fx {=TSHoltForecast(Example_Forecast!$B$2:$B$42, TRUE, 10, 1, 30000)}

	A	B	C	D	E	F	G	H	I	J	K	L
1	Time Step	Value		HOLT'S EXPONENTIAL SMOOTHING FORECAST (68%)								
2	40	9.78636		*Time Step*	*Low*	*Middle*	*High*		*Time Step*	*Value*	*Mov. Avg.*	
3	39	8.997445		0	9.78636	9.78636	9.78636		-9	8.01612	7.361454	
4	38	8.80113		1	9.138287	9.927835	10.74016		-8	8.869635	7.924289	
5	37	8.913955		2	8.880414	10.0702	11.27097		-7	7.356008	8.881433	
6	36	7.627105		3	8.666806	10.21257	11.74659		-6	8.054267	7.203052	
7	35	7.249385		4	8.459875	10.35494	12.22183		-5	7.249385	7.992019	
8	34	8.054267		5	8.271828	10.49731	12.69639		-4	7.627105	7.10741	
9	33	7.356008		6	8.101584	10.63968	13.17635		-3	8.913955	7.540741	
10	32	8.869635		7	7.933695	10.78205	13.64404		-2	8.80113	8.977593	
11	31	8.01612		8	7.730641	10.92442	14.10964		-1	8.997445	8.847264	
12	30	7.52358		9	7.522631	11.06678	14.60412		0	9.78636	9.059697	
13	29	8.345424		10	7.351423	11.20915	15.11377					
14	28	9.101151										
15	27	10.29342										

Figure 1.5

1.2 Executing Functions on the Spreadsheet

In this case, you need to have detailed knowledge of the function you are willing to execute. Let us continue with the same data set as in the Figure 1.1. Again, we'd like to apply Holt's exponential smoothing method and forecast the future.

As mentioned previously, the function to use is TSHoltForecast. Suppose that we'd like to forecast 10 steps with 68% confidence interval. Then, the output range must have size equal to 11×3 (11 rows and 3 columns). You should select a range somewhere on the spreadsheet that matches with this expected output size (see Figure 1.6). Please, notice that the output size is not 10×3. The additional row in the output is there to repeat the most recent observation of the time series data. Then, in the formula bar you can type in the function with the arguments as following.

=TSHoltForecast(Example_Forecast!B2:B42, TRUE, 10, 1, 20000)

The first argument *Example_Forecast!B2:B42* is the range of the data set and the second argument *TRUE* tells whether the time series data is in the ascending order or not. The following argument *10* is the number of forecast steps, *1* means 68% confidence interval and *20000* is the number of simulated samples for calculating the confidence interval.

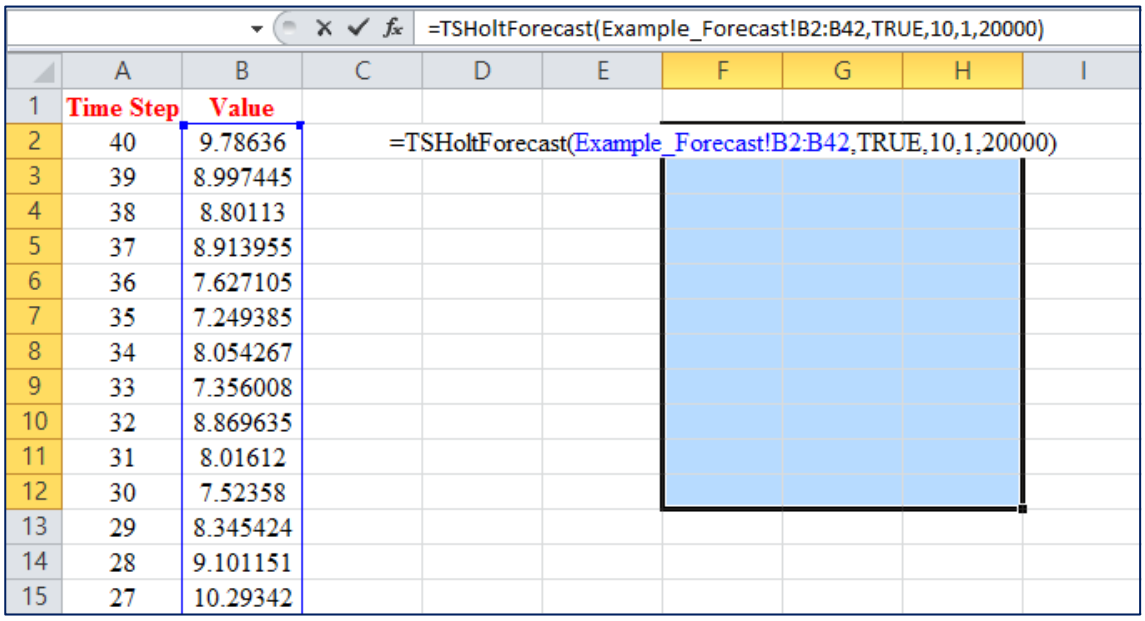


Figure 1.6

Since this is a multi-cell array formula, you should press the combination **CTRL+SHIFT+ENTER** in order to execute. The result is shown in the following screen shot (Figure 1.7).

F2 {=TSHoltForecast(Example_Forecast!B2:B42,TRUE,10,1,20000)}

	A	B	C	D	E	F	G	H	I
1	Time Step	Value							
2	40	9.78636				9.78636	9.78636	9.78636	
3	39	8.997445				9.138561	9.927835	10.74261	
4	38	8.80113				8.878554	10.0702	11.26729	
5	37	8.913955				8.667639	10.21257	11.74358	
6	36	7.627105				8.457203	10.35494	12.21157	
7	35	7.249385				8.270928	10.49731	12.70812	
8	34	8.054267				8.090529	10.63968	13.17825	
9	33	7.356008				7.93392	10.78205	13.65445	
10	32	8.869635				7.738261	10.92442	14.1217	
11	31	8.01612				7.51977	11.06678	14.60343	
12	30	7.52358				7.360159	11.20915	15.12323	
13	29	8.345424							
14	28	9.101151							
15	27	10.29342							

Figure 1.7

Please, remember that for other functions where the output can be contained in just one cell, pressing **ENTER** would suffice to execute. For example: AvgExp.

1.3 Executing Functions from a VBA Script

In order to write a VBA script you need to open the Visual Basic Editor from the **Developer** tab. However, in some instances of Excel the **Developer** tab may appear disabled. This is because VBA scripts (macros) are so powerful that they could cause a lot of damage if you are not absolutely sure of what you are doing. So Excel has the VBA scripting turned off by default. In order to display the **Developer** tab and enable the VBA scripting, please follow the steps below:

1. On the **File** tab, choose **Options** to open the **Excel Options** dialog box.
2. Click **Customize Ribbon** on the left side of the dialog box.
3. In the **Customize the Ribbon** drop down list, select **Main Tabs** and then turn on the **Developer** check box.
4. Click **OK**.

After Excel displays the **Developer** tab, click on it and press the **Visual Basic** button from the ribbon menu. From the top menu execute **Insert → Module** to insert a code module. Now, you are ready to write your VBA scripts.

A VBA script that executes the function TSHoltForecast may look like the following Figure 1.8. Please, notice that the method Application.Run was used to execute the actual function. The array named "*res*" of type double will contain the resulting output.

```
Option Explicit
Option Base 1

Sub RunFunction()

Dim vres As Variant
Dim res() As Double
Dim rngSource As Range
Dim boolAscend As Boolean
Dim intN As Integer
Dim intInterval As Integer
Dim intSamples As Integer
Dim i As Integer

' Input arguments.
Set rngSource = ActiveSheet.Range("B2:B42")
boolAscend = True
intN = 10
intInterval = 1
intSamples = 20000

ReDim res(intN + 1, 3) As Double

' Execute the function.
vres = Application.Run("TSHoltForecast", rngSource, boolAscend, intN, intInterval, intSamples)

' Copy to a typed array.
For i = 1 To intN + 1
    res(i, 1) = vres(i, 1)
    res(i, 2) = vres(i, 2)
    res(i, 3) = vres(i, 3)
Next

'
' res is now a 11x3 matrix with the output result.
'

End Sub
```

Figure 1.8

2. Function List

From the following page and on, a detailed list of PrimaXL functions with purpose, syntax, input arguments, output and remarks is given. In some cases, references to relevant journal articles or books are also provided.

AdjMatrix

Purpose

Creates a simple adjacency matrix representation of a network using cosine similarity.

Syntax

AdjMatrix(*rangeData, boolRowOrdered, realFraction*)

Input Arguments

rangeData	Cell range containing the data. The data can be interpreted as coordinates (vectors) of the nodes. These coordinates can be ordered as rows or columns (see *boolRowOrdered*).
boolRowOrdered	*TRUE* means that different nodes are ordered as rows and different features ordered as columns. While, *FALSE* means that the nodes are ordered as columns and the features ordered as rows.
realFraction	When the "strength" of edges between different nodes is ranked according to the cosine similarity, this value specifies the upper fraction of the edges that are kept in the adjacency matrix.

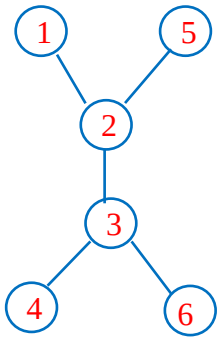
Output

The output is a square matrix of dimension $N_{nodes} \times N_{nodes}$ where N_{nodes} denotes the number of nodes (observations) in the input data set. A simple adjacency matrix is a transpose symmetric matrix containing elements 0s and 1s. An element equal to 1 denotes the existence of an undirected edge between the nodes. In this case, the cosine similarity is large enough that would rank among the top fraction specified by *realFraction*. An element equal to 0 means that the cosine similarity between the nodes is not large enough.

Please, notice that although the cosine similarity is used to build the adjacency matrix, details are not shown in the final output: only 0s and 1s.

For example, the following adjacency matrix is a representation of the network below. The node numbers correspond to the row (column) position in the adjacency matrix.

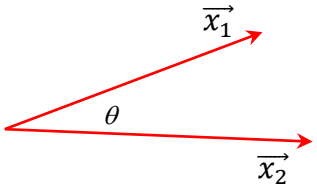
0	1	0	0	0	0
1	0	1	0	1	0
0	1	0	1	0	1
0	0	1	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0



Remarks

If we represent the nodes as vectors, the cosine similarity is the cosine of the angle between any two vectors (that is, nodes). The smaller this angle is, the more closely aligned the vectors are. The cosine reaches its maximum value at zero angle and as the angle increases its value decreases. In general, if we denote the vectors as $\overrightarrow{x_1}$ and $\overrightarrow{x_2}$ the cosine of the angle between them is given by the following relation. Here, $\overrightarrow{x_1}, \overrightarrow{x_2}$ denotes the vector inner product (dot product).

$$\cos\theta = \frac{\overrightarrow{x_1}, \overrightarrow{x_2}}{|\overrightarrow{x_1}||\overrightarrow{x_2}|}$$



See also NetworkCommunityStruct and NetworkModularity.

AvgExp

Purpose

Calculates the exponential average.

Syntax

AvgExp(*rangeTS*, *realAlpha*, *boolAscend*)

Input Arguments

rangeTS	Cell range of a univariate time series. It must be a single column array.
realAlpha	The parameter alpha (α). The value must lie between 0 and 1.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.

Output

If the input range contains $N+1$ values $X_t, \cdots, X_{t-N}$, the output is obtained by applying the following formula.

$$AvgExp = \frac{X_t + (1-\alpha) \times X_{t-1} + (1-\alpha)^2 \times X_{t-2} + \cdots + (1-\alpha)^N \times X_{t-N}}{1 + (1-\alpha) + (1-\alpha)^2 + \cdots + (1-\alpha)^N}$$

Remarks

This function is useful for calculating the moving averages of a time series. See also AvgRunning, AvgWeighted, AvgWeightedGeneral.

AvgRunning

Purpose

Calculates the running average.

Syntax

AvgRunning(*rangeTS*, *boolAscend*)

Input Arguments

rangeTS	Cell range of a univariate time series. It must be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.

Output

If the input range contains $N+1$ values $X_t, \cdots, X_{t-N}$, the running average is equal to the exponential average with the alpha parameter $\alpha = \frac{1}{N+1}$.

Remarks

This function is useful for calculating the moving averages of a time series. See also AvgExp, AvgWeighted, AvgWeightedGeneral.

AvgWeighted

Purpose

Calculates the weighted average. The weights are fixed.

Syntax

AvgWeighted(*rangeTS*, *boolAscend*)

Input Arguments

rangeTS	Cell range of a univariate time series. It must be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.

Output

If the input range contains $N+1$ values $X_t, \cdots, X_{t-N}$, the output is obtained by applying the following formula.

$$AvgWeighted = w_t \times X_t + w_{t-1} \times X_{t-1} + w_{t-2} \times X_{t-2} + \cdots + w_{t-N} X_{t-N}$$

where each weight w_{t-i} is calculated as

$$w_{t-i} = \frac{(N+1-i)}{1+2+3+\cdots+N+(N+1)}$$

Remarks

The weights w_{t-i} are normalized. This means the following:

$$\sum_{i=0}^{N} w_{t-i} = 1$$

This function is useful for calculating the moving averages of a time series. See also AvgExp, AvgRunning, AvgWeightedGeneral.

AvgWeightedGeneral

Purpose

Calculates the weighted average. The weights are provided by the user.

Syntax

AvgWeightedGeneral(*rangeTS*, *rangeWeight*)

Input Arguments

rangeTS	Cell range of a univariate time series. It must be a single column array.
rangeWeight	Cell range of the weights. It must a single column array with the same length as *rangeTS*. The weights can be of any positive value and need not be normalized.

Output

If the input range contains $N+1$ values $X_t, \cdots, X_{t-N}$ and correspondingly the weight range contains values $V_t, \cdots, V_{t-N}$, the output is obtained by applying the following formula.

$$AvgWeightedGeneral = w_t \times X_t + w_{t-1} \times X_{t-1} + w_{t-2} \times X_{t-2} + \cdots + w_{t-N} X_{t-N}$$

where the normalized weight w_{t-i} is given as

$$w_{t-i} = \frac{V_{t-i}}{\sum_{i=0}^{N} V_{t-i}}$$

Remarks

The weights w_{t-i} are normalized. This means the following:

$$\sum_{i=0}^{N} w_{t-i} = 1$$

.

This function is useful for calculating the moving averages of a time series. See also AvgExp, AvgRunning, AvgWeighted. One useful example is the so called VWMA (Volume Weighted Moving Average) where X_t = *stock price* and V_t = *trading volume*.

ClusterCenters

Purpose

Calculates the cluster centers by Lloyd's algorithm.

Syntax

ClusterCenters(*rangeData*, *intN*, *intSeed*, *intRepeat*)

Input Arguments

rangeData	Cell range containing the data coordinates. These coordinates are row ordered. Different columns mean different features.
intN	Total number of clusters that is targeted (denoted as $N_{clusters}$).
intSeed	Seed for initializing the random number generator.
intRepeat	Number of times the algorithm is repeated with random initialization. The final pick is the one with optimal intra cluster distance.

Output

If the *rangeData* has dimension $N_{data} \times N_{features}$ and $N_{clusters}$ is the number of clusters that is targeted, the output has dimension $N_{clusters} \times N_{features}$. Different rows of the output correspond to the coordinates of the cluster centers.

Remarks

The cluster assignment of the input data is not explicitly provided by this function. One should assign a coordinate to the cluster of which center is the closest. One can perform this task easily from the PrimaXL ribbon menu.

CorrelatedSamples

Purpose

Generates correlated samples from uncorrelated ones.

Syntax

CorrelatedSamples(*rangeX*, *rangeCorrMat*)

Input Arguments

rangeX	Cell range of the multiple random variables organized in contiguous columns. Each column contains samples of different random variable X_i. In principle, the samples belonging to different columns are not correlated. The dimension of this range is $N \times M$ (N rows and M columns).
rangeCorrMat	Cell range containing the correlation matrix $\tilde{\mathrm{P}}$. The dimension should be $M \times M$.

Output

If the input range has dimension $N \times M$, then the output also has dimension $N \times M$. The correlations between the columns are determined by the correlation matrix $\tilde{\mathrm{P}}$. The transformation is such that the means and the variances of the columns are preserved.

Remarks

First, let's denote as $\tilde{X}$ the matrix of the uncorrelated standardized samples. In fact, this is just a matrix notation of the *rangeX*. We can apply Cholesky decomposition to the correlation matrix $\tilde{\mathrm{P}}$ and express it in terms of the transformation matrix $\tilde{U}$.

$$\tilde{\mathrm{P}} = \tilde{U}\tilde{U}^t$$

Then, the matrix containing the correlated samples ($\tilde{X}_c$) can be obtained by the following linear transformation. See also CorrMatrix.

$$\tilde{X}_c = \tilde{X}\tilde{U}^t$$

CorrMatrix

Purpose

Calculates the correlation matrix of multiple random variables.

Syntax

CorrMatrix(*rangeX*)

Input Arguments

rangeX	Cell range of the multiple random variables organized in contiguous columns. Each column contains samples of different random variable X_i. The samples belonging to different columns may or may not be correlated.

Output

If the input range has *M* columns, the output is a square matrix of dimension $M \times M$ of the following form.

$$\begin{pmatrix} 1 & \rho_{12} & \cdots & \rho_{1M} \\ \rho_{21} & 1 & \cdots & \rho_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{M1} & \rho_{M2} & \cdots & 1 \end{pmatrix}$$

Each element of the matrix is calculated as

$$\rho_{ij} = Corr(X_i, X_j) = \frac{Cov(X_i, X_j)}{\sqrt{Var(X_i)Var(X_j)}}$$

Remarks

A valid correlation matrix must be symmetric ($\rho_{ij} = \rho_{ji}$) and the diagonal elements must be equal to the unity ($\rho_{ii} = 1$). Also, a valid correlation matrix must be a positive semi-definite matrix, meaning that the eigenvalues are all non-negative. See also CovMatrix.

CovMatrix

Purpose

Calculates the covariance matrix of multiple random variables.

Syntax

CovMatrix(*rangeX*)

Input Arguments

rangeX	Cell range of the multiple random variables organized in contiguous columns. Each column contains samples of different random variable X_i. The samples belonging to different columns may or may not be correlated.

Output

If the input range has *M* columns, the output is a square matrix of dimension $M \times M$ of the following form.

$$\begin{pmatrix} \sigma_{11} & \sigma_{12} & \cdots & \sigma_{1M} \\ \sigma_{21} & \sigma_{22} & \cdots & \sigma_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{M1} & \sigma_{M2} & \cdots & \sigma_{MM} \end{pmatrix}$$

Each element of the matrix is calculated as

$$\sigma_{ij} = Cov(X_i, X_j)$$

Remarks

A valid covariance matrix must be symmetric ($\sigma_{ij} = \sigma_{ji}$). Also, a valid covariance matrix must be a positive semi-definite matrix, meaning that the eigenvalues are all non-negative. See also CorrMatrix.

DataCompleteRows

Purpose

Eliminates the rows with missing data and keeps only the complete rows.

Syntax

DataCompleteRows(*rangeData*)

Input Arguments

rangeData	Cell range of the data. Different features are organized as columns while different observations are organized as rows.

Output

The number of columns in the output range is the same as that of the input range. However, the number of rows may be reduced if there were any missing data in the input range.

Remarks

See also DataCountCompleteRows and DataCompleteRows2.

DataCompleteRows2

Purpose

Eliminates the rows with missing data and keeps only the simultaneously complete rows.

Syntax

DataCompleteRows2(*rangeDataX*, *rangeDataY*)

Input Arguments

rangeDataX	Cell range of the data set *X*. Different features are organized as columns while different observations are organized as rows.
rangeDataY	Cell range of the data set *Y*. The number of rows must match between *rangeDataX* and *rangeDataY*, while the number of columns may or may not match.

Output

The sets *X* and *Y* are first combined by columns (*X* on the left and *Y* on the right) and then the equivalent of DataCompleteRows is applied. The rows with missing data either from the set *X* or set *Y* are eliminated. The number of columns in the output range is the sum of the column numbers of the sets *X* and *Y*. The number of rows may be reduced if there were any missing data in the input ranges.

Remarks

See also DataCountCompleteRows2 and DataCompleteRows.

DataCountCompleteRows

Purpose

Counts the number of complete rows in the data range.

Syntax

DataCountCompleteRows(*rangeData*)

Input Arguments

rangeData	Cell range of the data. Different features are organized as columns while different observations are organized as rows.

Output

The output is an integer value.

Remarks

See also DataCountCompleteRows2 and DataCompleteRows.

DataCountCompleteRows2

Purpose

Counts the number of simultaneously complete rows in the data ranges.

Syntax

DataCountCompleteRows2(*rangeDataX*, *rangeDataY*)

Input Arguments

rangeDataX	Cell range of the data set *X*. Different features are organized as columns while different observations are organized as rows.
rangeDataY	Cell range of the data set *Y*. The number of rows must match between *rangeDataX* and *rangeDataY*, while the number of columns may or may not match.

Output

The output is an integer value.

The sets *X* and *Y* are first combined by columns (*X* on the left and *Y* on the right) and then the equivalent of DataCountCompleteRows is applied. The rows with missing data either from the set *X* or set *Y* are not included in the counting.

Remarks

See also DataCountCompleteRows and DataCompleteRows2.

DataTransform

Purpose

Transforms the data by log, standardization and normalization.

Syntax

DataTransform(*rangeData*, *boolLog*, *intChoice*)

Input Arguments

rangeData	Cell range of the data. Different features are organized as columns while different observations are organized as rows.
boolLog	If *TRUE*, log function is applied to the data set. The values in the data set must be larger than 0.
intChoice	If 0, no further transformation is applied. If 1, standardization by column is applied. If 2, normalization by column is applied.

Output

The output range has the same size as the input data range.

Remarks

The values are standardized as following. μ_{column} is the column mean and σ_{column} is the column standard deviation.

$$x_{standardized} = \frac{x - \mu_{column}}{\sigma_{column}}$$

The values are normalized as following. x_{max} is the column maximum and x_{min} is the column minimum.

$$x_{normalized} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

See also DataTransformTest.

DataTransformTest

Purpose

Transforms the test data set taking into account the train data set.

Syntax

DataTransformTest(*rangeTrain*, *rangeTest*, *boolLog*, *intChoice*)

Input Arguments

rangeTrain	Cell range of the train data. Different features are organized as columns while different observations are organized as rows.
rangeTest	Cell range of the test data. The number of columns must match between the train and the test data sets.
boolLog	If *TRUE*, log function is applied to the test data set. The values in the data set must be larger than 0.
intChoice	If 0, no further transformation is applied. If 1, standardization by column is applied. If 2, normalization by column is applied.

Output

The output range has the same size as the test data range.

Remarks

The μ_{column}, σ_{column}, x_{max} and x_{min} are calculated from the row joined set of the train and test data.

See also DataTransform.

DetrendPoly

Purpose

Given *X* vs *Y* data sets, produces a table of residuals after subtracting polynomial fit from the *Y* data.

Syntax

DetrendPoly(*rangeX, rangeY, intPower*)

Input Arguments

rangeX	Cell range of the data set *X*. It must be a single column array of a given length (denoted as *N*).
rangeY	Cell range of the data set *Y*. It must be a single column array of the same length as *rangeX*.
intPower	Maximum power of the fitting polynomial. Possible values are: 1 for the linear fit, 2 for the quadratic fit and 3 for the cubic fit.

Output

If we denote as $\hat{Y}$ the fitted values of *Y*, the output is the residual $Y - \hat{Y}$. It is presented as a single column array of the same length as *rangeX* and *rangeY*.

Remarks

The polynomial fit is as explained in the functions FitPoly and FitPolyCoeff. "Fitting" is different from "interpolating" in that a fitting curve usually does not necessarily pass through the data points. The fitting line or curve shows the general trend. Thus, getting the residuls is equivalent to the detrending.

FitBezier

Purpose

Given *X* vs *Y* data sets, produces a table of Bezier fit on the grid.

Syntax

FitBezier(*rangeX*, *rangeY*, *intGrids*)

Input Arguments

rangeX	Cell range of the data set *X*. It must be a single column array of a given length (denoted as *N*).
rangeY	Cell range of the data set *Y*. It must be a single column array of the same length as *rangeX*.
intGrids	Number of grid points between the first and the last coordinate pairs. The value of $intGrid$ has to be larger than 0.

Output

The output is a two column array. Each column contains the fitted values of *X* and *Y* respectively.

Remarks

Here, we assume *N* measurements for *X* and *Y*. We denote as P_i a pair of data points of *X* and *Y*: $P_i = (x_i, y_i)$.

Bezier curve is an example of the parametric fitting with the fixed initial and the final coordinates: P_0 and P_N. The Bezier curve must start at P_0 and must end at P_N. However, it does not necessarily have to pass through the intermediate data coordinates: P_i with $i = 1, \cdots, N-1$. The difference between the fit and the interpolation has been remarked in the context of polynomial fit vs polynomial interpolation (see FitPoly and InterpolatePoly).

A conceptual difference from the polynomial fit has to be noted. In the polynomial fit

only the values of *Y* are calculated by the procedure, while in the Bezier fit the values of both *X* and *Y* are calculated.

The number of grid points can be controlled by *intGrids*. If $intGrids = g$, the total number of grid points is $g + 2$ because the terminal points are already counted as +2. These grid points are generated with evenly spaced intermediate parameter *t* values. However, this does not mean that the coordinates of *X* and *Y* are evenly spaced. This is another difference from the polynomial fit, where the grid points of *X* can easily be made evenly spaced.

Bezier curve $B(t)$ is calculated using the following formula. Here, *t* is the intermediate parameter confined to the range $[0,1]$. $B(t)$ as well as P_i can be regarded as 2 dimensional vectors on the *X*-*Y* plane.

$$B(t) = \sum_{i=0}^{N} \binom{N}{i} (1-t)^{N-i} t^i P_i$$

FitPoly

Purpose

Given *X* vs *Y* data sets, produces a table of polynomial fit on the grid. The maximum power of the fitting polynomial can be controlled.

Syntax

FitPoly(*rangeX, rangeY, intPower, intGrids*)

Input Arguments

rangeX	Cell range of the data set *X*. It must be a single column array of a given length (denoted as *N*).
rangeY	Cell range of the data set *Y*. It must be a single column array of the same length as *rangeX*.
intPower	Maximum power of the fitting polynomial. Possible values are: 1 for the linear fit, 2 for the quadratic fit and 3 for the cubic fit.
intGrids	Number of grid points between the maximum and the minimum values of *X*. When $intGrid = 0$, the same data points of *X* are used as grid points.

Output

When $intGrids = 0$, the output is a single column array containing the fitted values of *Y* (denoted as $\hat{Y}$). When $intGr\square\square s > 0$, the output is a two column array: the first column contains the grid coordinates of *X* while the second column contains the fitted values $\hat{Y}$.

Remarks

Here, we assume *N* measurements of *X* and *Y*: we denote as x_i the values of *X* and as y_i the values of *Y*, where the sub-index $i = 1,2,\cdots,N$. The following dependency is assumed. Instead of *intPower*, we use the notation "*K*".

$$\begin{gathered}\beta_0 + \beta_1 x_1 + \beta_2 {x_1}^2 + \cdots + \beta_K {x_1}^K \approx y_1 \\ \beta_0 + \beta_1 x_2 + \beta_2 {x_2}^2 + \cdots + \beta_K {x_2}^K \approx y_2 \\ \vdots \\ \beta_0 + \beta_1 x_N + \beta_2 {x_N}^2 + \cdots + \beta_K {x_N}^K \approx y_N\end{gathered}$$

The number of measurements *N* should be at least equal or larger than *K*. The coefficients $\beta_0, \beta_1, \cdots, \beta_K$ are determined by the ordinary least square method. See also OLSCoeff.

When $intGrids = 0$, the interpolation $\hat{Y}$ has the same dimension as *X* and Y (that is *N*). However, the number of grid points can also be controlled by *intGrids*. If $intGrids = g$, the total number of grid points is $g + 2$ because the maximum and the minimum of the data set *X* are already counted as +2 points.

"Fitting" is different from "interpolating" in that a fitting curve usually does not necessarily pass through the data points. It only shows the general trend. While an interpolating curve must necessarily pass through all of the data points. See also DetrendPoly and InterpolatePoly.

FitPolyCoeff

Purpose

Given *X* vs *Y* data sets, calculates the coefficients of the polynomial fit. The maximum power of the fitting polynomial can be controlled.

Syntax

FitPolyCoeff(*rangeX*, *rangeY*, *intPower*)

Input Arguments

rangeX	Cell range of the data set *X*. It must be a single column array of a given length (denoted as *N*).
rangeY	Cell range of the data set *Y*. It must be a single column array of the same length as *rangeX*.
intPower	Maximum power of the fitting polynomial. Possible values are: 1 for the linear fit, 2 for the quadratic fit and 3 for the cubic fit.

Output

The output is a $(K+1)\times 4$ matrix, where *K*=*intPower* is the maximum power of the fitting polynomial. In this case, the total number of coefficients is $K+1$. The first column corresponds to the coefficients of the polynomial. The sub-index 0 indicates the constant intercept. The second column corresponds to the errors. The third and the fourth columns show the t-statistics and the *p*-values.

$$\begin{matrix} \beta_0 & error_0 & t-stat_0 & p-value_0 \\ \beta_1 & error_1 & t-stat_1 & p-value_1 \\ \vdots & \vdots & \vdots & \vdots \\ \beta_K & error_K & t-stat_K & p-value_K \end{matrix}$$

Remarks

Here, we assume *N* measurements of *X* and *Y*. Thus, we denote as x_i the values of *X* and as y_i the values of *Y*, where the sub-index $i = 1,2,\cdots,N$. The following dependency is assumed. Also here instead of *intPower*, we use the notation "*K*".

$$\begin{gathered}\beta_0 + \beta_1 x_1 + \beta_2 {x_1}^2 + \cdots + \beta_K {x_1}^K \approx y_1 \\ \beta_0 + \beta_1 x_2 + \beta_2 {x_2}^2 + \cdots + \beta_K {x_2}^K \approx y_2 \\ \vdots \\ \beta_0 + \beta_1 x_N + \beta_2 {x_N}^2 + \cdots + \beta_K {x_N}^K \approx y_N\end{gathered}$$

The number of measurements *N* should be at least equal or larger than *K*. The coefficients $\beta_0, \beta_1, \cdots, \beta_K$ are determined by the ordinary least square method. See also OLSCoeff.

"Fitting" is different from "interpolating" in that a fitting curve usually does not necessarily pass through the data points. It only shows the general trend. While an interpolating curve must necessarily pass through all of the data points. See also FitPoly and InterpolatePoly.

FitPolyPt

Purpose

Given *X* vs *Y* data sets, applies the polynomial fit at one specific point. The maximum power of the fitting polynomial can be controlled.

Syntax

FitPoly(*rangeX, rangeY, intPower, realX*)

Input Arguments

rangeX	Cell range of the data set *X*. It must be a single column array of a given length (denoted as *N*).
rangeY	Cell range of the data set *Y*. It must be a single column array of the same length as *rangeX*.
intPower	Maximum power of the fitting polynomial. Possible values are: 1 for the linear fit, 2 for the quadratic fit and 3 for the cubic fit.
realX	Value of *X* at which the fitting is applied.

Output

This function returns a fitted value of *Y*.

Remarks

Here, we assume *N* measurements of *X* and *Y*. The fitting curve follows the "general trend" of the given data. Thus, "fitting" is different from "interpolating" in that a fitting curve usually does not pass through the data points while an interpolating curve must necessarily pass through all of the data points. See the remarks of FitPoly, FitPolyCoeff and InterpolatePoly.

Unlike the interpolation, the values of *realX* are not restricted to the interval defined by the maximum and the minimum of the data set *X*.

InterpolateLinear

Purpose

Given *X* vs *Y* data sets, produces a table of linear interpolation on the grid.

Syntax

InterpolateLinear(*rangeX*, *rangeY*, *intGrids*)

Input Arguments

rangeX	Cell range of the data set *X*. It must be a single column array of a given length.
rangeY	Cell range of the data set *Y*. It must be a single column array of the same length as *rangeX*.
intGrids	Number of the evenly spaced grid points between the maximum and the minimum values of the data set *X*. It must be a positive integer.

Output

The output is a two column array. The first column corresponds to the grid points of *X*, while the second column is the result of interpolating on these grid points.

The number of grid points can be controlled by *intGrids*. If $intGrids = g$, the total number of grid points is $g + 2$ because the maximum and the minimum of *X* are already counted as +2 points.

Remarks

The interpolating points always lie on one of the linear segments that join the adjacent data points. The interpolation is continuous throughout the whole range. However, the continuity of the derivative of any order (first, second, or higher) is usually not guaranteed. Thus, this interpolation does not usually provide a “smooth appearance”.

InterpolateLinearPt

Purpose

Given *X* vs *Y* data sets, applies the linear interpolation at one specific point.

Syntax

InterpolateLinearPt(*rangeX*, *rangeY*, *realX*)

Input Arguments

rangeX	Cell range of the data set *X*. It must be a single column array of a given length.
rangeY	Cell range of the data set *Y*. It must be a single column array of the same length as *rangeX*.
realX	Value of *X* at which the interpolation is applied.

Output

This function returns a single interpolated value of *Y*.

Remarks

This function is analogous to InterpolateLinear. However, instead of producing a interpolation table with several grid points, now the interpolation is applied to one specific point. The value of *realX* must lie within the interval defined by the minimum and the maximum of the data set *X*.

InterpolatePoly

Purpose

Given *X* vs *Y* data sets, produces a table of polynomial interpolation on the grid. The maximum power of the polynomial is $N - 1$, where *N* is the number of the data points.

Syntax

InterpolatePoly(*rangeX*, *rangeY*, *intGrids*)

Input Arguments

rangeX	Cell range of the data set *X*. It must be a single column array of a given length.
rangeY	Cell range of the data set *Y*. It must be a single column array of the same length as *rangeX*.
intGrids	Number of the evenly spaced grid points between the maximum and the minimum values of the data set *X*.

Output

The output is a two column array. The first column corresponds to the grid points of *X* while the second column is the result of interpolating on these grid points (we denote them as $\hat{Y}$).

The number of grid points can be controlled by *intGrids*. If $intGrids = g$, the total number of grid points is $g + 2$ because the maximum and the minimum of *X* are already counted as +2 points.

Unlike the fitting, for the interpolation it is almost pointless to use the original data points of *X* as the grid. Because, by definition the interpolating curve must pass through all of the given data points. Thus, if we want to see how the interpolating curve behaves away from the data points, the grid points must be different from these. Here, we always assume evenly spaced grid points.

Remarks

The polynomial interpolation is also known as Lagrange interpolation. Given a set of data points for *X* and *Y*, the interpolating polynomial $f(x)$ is constructed by the following method. Notice that here we are denoting as x_i each data point of *X* and as y_i each data point of *Y*. Then, the interpolation at any other value of *x* is $\hat{y} = f(x)$.

$$f(x) = \sum_{i=1}^{N} L_i(x) y_i$$

with

$$L_i(x) = \prod_{j=1, j\neq i}^{N} \frac{x - x_j}{x_i - x_j}$$

The polynomial interpolation is continuous throughout the whole range. Also the derivatives of order up to $N - 1$ are continuous. Thus, the polynomial interpolation is "smooth".

InterpolatePolyPt

Purpose

Given *X* vs *Y* data sets, applies the polynomial interpolation at one specific point.

Syntax

InterpolatePolyPt(*rangeX*, *rangeY*, *realX*)

Input Arguments

rangeX	Cell range of the data set *X*. It must be a single column array of a given length.
rangeY	Cell range of the data set *Y*. It must be a single column array of the same length as *rangeX*.
realX	Value of *X* at which the interpolation is applied.

Output

This function returns a single interpolated value of *Y*.

Remarks

The polynomial interpolation is also known as Lagrange interpolation. This function is analogous to InterpolatePoly. However, instead of producing a table with several grid points, now the interpolation is applied only to one specific point. The value of *realX* must lie within the interval defined by the minimum and the maximum of the data set *X*.

InterpolateSpline

Purpose

Given *X* vs *Y* data sets, produces a table of interpolation on the grid using the cubic spline curve.

Syntax

InterpolateSpline(*rangeX*, *rangeY*, *intGrids*)

Input Arguments

rangeX	Cell range of the data set *X*. It must be a single column array of a given length.
rangeY	Cell range of the data set *Y*. It must be a single column array of the same length as *rangeX*.
intGrids	Number of the evenly spaced grid points between the maximum and the minimum values of *X*.

Output

The output is a two column array. The first column corresponds to the grid points of *X*, while the second column is the result of interpolating at these grid points.

The number of grid points can be controlled by *intGrids*. If $intGrids = g$, the total number of grid points is $g + 2$ because the maximum and the minimum of *X* are already counted as +2 points.

Unlike the fitting, for the interpolation it is almost pointless to use the original data points of *X* as the grid. Because, by definition the interpolating curve must pass through all of the given data points. Thus, if we want to see how the interpolating curve behaves away from the data points, the grid points must be different from these. Here, we always assume evenly spaced grid points.

Remarks

The cubic spline interpolation consists in joining the neighboring data points by cubic polynomials such that the continuities of the first and the second derivatives are maintained. Thus, the spline interpolation results in a “smooth” appearance. Also, the spline interpolation does not suffer from the “overshooting” as the polynomial interpolation.

InterpolateSplinePt

Purpose

Given *X* vs *Y* data sets, applies the cubic spline interpolation at one specific point.

Syntax

InterpolateSplinePt(*rangeX*, *rangeY*, *realX*)

Input Arguments

rangeX	Cell range of the data set *X*. It must be a single column array of a given length.
rangeY	Cell range of the data set *Y*. It must be a single column array of the same length as *rangeX*.
realX	Value of *X* at which the interpolation is applied.

Output

This function returns a single interpolated value of *Y*.

Remarks

This function is analogous to InterpolateSpline. However, instead of producing a table with several grid points, now the interpolation is applied only to one specific point. The value of *realX* must lie within the interval defined by the minimum and the maximum of the data set *X*.

Kalman

Purpose

Applies the Kalman algorithm to a time series.

Syntax

Kalman(*rangeZ, boolAscend, doubleX00, doubleP00, doubleA, doubleH, doubleQ, doubleR*)

Input Arguments

rangeZ	Cell range of the measured time series. It must be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
doubleX00	Initialization of the "posterior estimate" of the process: denoted as $\hat{X}_{0\|0}$.
doubleP00	Initialization of the "posterior estimate" of the process variance: denoted as $P_{0\|0}$. It must be a non-zero positive value.
doubleA	Transition coefficient.
doubleH	Transformation coefficient.
doubleQ	Process variance. It must be a non-zero positive value.
doubleR	Measurement variance. It must be a non-zero positive value.

Output

The output consists of 3 columns: showing the posterior estimate of the state, the posterior estimate of the variance and the Kalman gain. In the output, descending time order is assumed regardless of the ordering of the measured time series.

Remarks

The Kalman filter is one of the so-called "data fusion algorithms". Given a series of noisy measurements and a underlying model, the Kalman filter combines them optimally such that the outcome is more accurate and smoother estimate of the measured series (called "posterior estimate").

Here, we assume a discrete linear system. The system is assumed to evolve according to the underlying model.

$$X_t = AX_{t-1} + \omega_t$$

Here, ω_t is the so-called "process noise" that is characterized by its variance denoted as *Q* (called "process variance"). The coefficient *A* is the "transition coefficient". The connection between the underlying model and the measured series is established by the following expression.

$$Z_t = HX_t + \nu_t$$

Here, Z_t represents the measured time series. *H* is the "transformation coefficient" and ν_t is the "measurement noise". The measurement noise is characterized by its variance *R* (called "measurement variance").

It is important to emphasize that the time series X_t is not directly measured. It can only be "estimated" by the Kalman filter. The estimated values of X_t are also called the "posterior estimates".

The Kalman algorithm is recursive. Starting from the initial time step it repeats the cycles of the prediction and update throughout the whole extension of the time series.

The prediction step can be summarized by the following equations.

$$\hat{X}_{t|t-1} = A\,\hat{X}_{t-1|t-1}$$

$$P_{t|t-1} = A^2\,P_{t-1|t-1} + Q$$

Here, the quantities such as $\hat{X}_{t|t-1}$ and $P_{t|t-1}$ with the sub-index "$t|t-1$" are the "prior estimates" (predictions). $\hat{X}_{t|t-1}$ is the prior estimate of the process and $P_{t|t-1}$ is the prior estimate of the process variance.

On the other hand, the quantities such as $\hat{X}_{t-1|t-1}$ and $P_{t-1|t-1}$ with the sub-index "$t-1|t-1$" are the "posterior estimates" (consolidated quantities from the previous step) of the process and the variance respectively.

Then, the update step where the prior estimates are "fused" with the actual measurement consists in applying the following equations.

$$\hat{X}_{t|t} = \hat{X}_{t|t-1} + K_t(Z_t - H\hat{X}_{t|t-1})$$

$$P_{t|t} = P_{t|t-1}(1 - K_t H)$$

$$K_t = \frac{P_{t|t-1}H}{H^2\, P_{t|t-1} + R}$$

Here, again the quantities such as $\hat{X}_{t|t}$ and $P_{t|t}$ with the sub-index "$t|t$" are the "posterior estimates" (consolidated or "fused" quantities) at the current time step (sub-index *t*). Please, note that the quantity K_t is the so-called "Kalman gain" that determines how the predicted model value and the actual measured value are combined.

Among the required input arguments of this function, $\hat{X}_{0|0}$ and $P_{0|0}$ are the initial values of the posterior estimate provided by the user. As long as they are "reasonable", the recursive process will work.

When the transition coefficient *A* is close to 1, the underlying model acquires the characteristics of the random walk. When the coefficient *A* is close to 0 (small enough to be negligible), the underlying model becomes white noise-like.

The transformation coefficient *H* accounts for the possible change of unit between the measurement and the model process. When there is no need of such adjustment, we can set $H = 1$.

The process variance *Q* quantifies the uncertainties and imperfections of the underlying model, while the measurement variance *R* quantifies the noise in the "measuring device". Reasonable values must be assigned to *Q* and *R*.

When, $Q >> R$ the posterior estimates follow closely the measurements and the smoothing effect is weakened. On the other hand, when $Q << R$, the model process prevails and greater smoothing effect can be expected.

LogisticCoeff

Purpose

Calculates the coefficients of logistic regression.

Syntax

LogisticCoeff(*rangeX*, *rangeY*)

Input Arguments

rangeX	Cell range of the independent variable(s) X_j, also called regressor(s) or predictor(s). It may encompass one or more columns. Each column represents one variable.
rangeY	Cell range of the dependent variable *Y*, also called response. It must be a single column array of length (rows) equal to that of *rangeX*. Acceptable values in this range are 1 (true) or 0 (false).

Output

Here, we assume the following notation.

β_i	Regression coefficient.
$error_i$	Standard error.
$z-stat_i$	z-statistic.
$p-value_i$	*p*-value.

If we denote the number of independent variables as *K*, the output has a dimension equal to $(K+1)\times 4$. The sub-index equal to 0 represents the constant intercept.

$$\beta_0 \qquad error_0 \qquad z-stat_0 \qquad p-value_0$$

$$\begin{matrix} \beta_1 & error_1 & z-stat_1 & p-value_1 \\ \vdots & \vdots & \vdots & \vdots \\ \beta_K & error_K & z-stat_K & p-value_K \end{matrix}$$

Remarks

Here, we consider N observations of each one of the K independent variables X_j denoted as x_{ij} and correspondingly N observations of the dependent variable Y denoted as y_i. The sub-index $i = 1,2,\cdots,N$ and $j = 1,2,\cdots,K$. The dependent variable Y can take values either 1 (true) or 0 (false).

The purpose of the logistic regression is to establish a relationship between the independent variable(s) X_j and the probability that the dependent variable Y has value equal to 1 (true). This is achieved by first introducing a new "intermediate dependent variable" s_i as linear combination of the independent variables.

$$s_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots \beta_K x_{iK} = \vec{\beta}^{\,t}\,\vec{x}_i$$

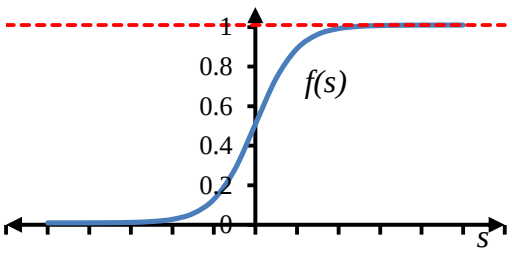
Here, we assumed the following definitions.

$$\vec{x}_i = \begin{pmatrix} 1 \\ x_{i1} \\ \vdots \\ x_{iK} \end{pmatrix} \quad , \quad \vec{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_K \end{pmatrix}$$

Then the probability that $y_i = 1$ can be calculated using the following expression.

$$f(s_i) = \frac{e^{\,s_i}}{1 + e^{\,s_i}}$$

The function $f(s_i)$ above is called "logistic function" and has the following shape.



The β_i coefficients are calculated by maximizing the log likelihood function and the standard errors are calculated from the Fisher information matrix.

LogisticCoeffOneVsAll

Purpose

Calculates the coefficients of logistic regression “One Vs All”.

Syntax

LogisticCoeffOneVsAll(*rangeX*, *rangeY*, *intY*)

Input Arguments

rangeX	Cell range of the independent variable(s) X_j, also called regressor(s) or predictor(s). It may encompass one or more columns. Each column represents one variable.
rangeY	Cell range containing the class labels. It must be a single column array of length (rows) equal to that of *rangeX*. The possible values are integer numbers *0,1,⋯, M*. Each one of these numbers would encode a class or category. There should be no gap meaning that the numbers *0,1,⋯, M* must occur at least once.
intY	Class label against which the rest of classes are lumped together as “another”.

Output

If we denote the number of independent variables as *K*, the output has a dimension equal to $(K+1)\times 4$. The sub-index equal to 0 represents the constant intercept.

$$\begin{matrix} \beta_0 & error_0 & z-stat_0 & p-value_0 \\ \beta_1 & error_1 & z-stat_1 & p-value_1 \\ \vdots & \vdots & \vdots & \vdots \\ \beta_K & error_K & z-stat_K & p-value_K \end{matrix}$$

Remarks

The chosen class (specified by *intY*) is taken as “1” of the simple logistic regression and the rest is taken as “0”.

See also LogisticCoeff and LogisticForecastMulti.

LogisticConfMatrix

Purpose

Calculates the confusion matrix by logistic regression.

Syntax

LogisticConfMatrix(*rangeX1*, *rangeY1*, *rangeX2*, *rangeY2*, *realCutoff*)

Input Arguments

rangeX1	Cell range of the independent variable(s). It may have one or more columns. This range forms part of the training data set.
rangeY1	Cell range of the dependent variable. It must be a single column array of length equal to that of *rangeX1*. Acceptable values in this range are 1 or 0. This range also forms part of the training data set.
rangeX2	Cell range of the independent variable(s) of the test data set. The number of columns must be equal to that of *rangeX1*. However, the number of rows (length) does not need to match.
rangeY2	Cell range of the dependent variable of the test data set. It must be a single column array of length equal to that of *rangeX2*. Acceptable values in this range are 1 or 0.
realCutoff	Cutoff probability of the positive (1 or true) identification. It must be a number between 0 and 1.

Output

The output is a matrix with two rows and two columns that reports the number of false positives, false negatives, true positives, and true negatives.

	Actual 0	*Actual 1*
Predicted 0	True Negative	False Negative
Predicted 1	False Positive	True Positive

True Positive (TP): Count of actual 1s that were correctly classified.

True Negative (TN): Count of actual 0s that were correctly classified.

False Positive (FP): Count of actual 0s that were wrongly classified as 1.

False Negative (FN): Count of actual 1s that were wrongly classified as 0.

Remarks

Using the confusion matrix, we can calculate the following quantities:

$$Sensitivity = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{TN + FP}$$

$$Positive\ Predictive\ Rate = \frac{TP}{TP + FP}$$

$$Negative\ Predictive\ Rate = \frac{TN}{TN + FN}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

See also LogisticROC and LogisticConfMatrixMulti.

LogisticConfMatrixMulti

Purpose

Calculates the confusion matrix by multiclass logistic regression: one vs all.

Syntax

LogisticConfMatrixMulti(*rangeX1, rangeY1, rangeX2, rangeY2*)

Input Arguments

rangeX1	Cell range of the independent variable(s). It may have one or more columns. This range forms part of the training data set.
rangeY1	Cell range containing the class labels. It must be a single column array of length (rows) equal to that of *rangeX1*. This range also forms part of the training data set. The possible values are integer numbers *0,1,⋯, M*. Each one of these numbers would encode a class or category. There should be no gap meaning that the numbers *0,1,⋯, M* must occur at least once.
rangeX2	Cell range of the independent variable(s) of the test data set. The number of columns must be equal to that of *rangeX1*. However, the number of rows (length) does not need to match.
rangeY2	Cell range of the dependent variable of the test data set. It must be a single column array of length equal to that of *rangeX2*. Acceptable values are the same as those present in the *rangeY1*.

Output

The output is a matrix with *M* rows and *M* columns that reports the classification

accuracy. It is structured as following:

	Actual 0	*Actual 1*	⋯	*Actual M*
Predicted 0	Correctly classified			
Predicted 1		Correctly classified		
⋮			⋱	⋮
Predicted M			⋯	Correctly classified

The diagonal elements count the correct classifications while the off diagonal elements count the miss classifications.

Remarks

Using the confusion matrix, we can calculate the following:

$$Accuracy = \frac{Sum\ of\ the\ diagonal\ elements}{Sum\ of\ all\ the\ elements}$$

It can be noted that the test set may or may not overlap (or coincide) with the training set. One can train with one set and then test on another set and estimate the classification accuracy.

See also LogisticROC , LogisticForecastMulti and LogisticConfMatrix.

LogisticForecast

Purpose

Forecasts the probability of 1 (true) in the dependent variable by applying the logistic regression method.

Syntax

LogisticForecast(*rangeX1*, *rangeY1*, *rangeX2*)

Input Arguments

rangeX1	Cell range of the independent variable(s), also called regressor(s) or predictor(s). It may encompass one or more columns. Each column represents one variable. This range forms part of the training data set.
rangeY1	Cell range of the dependent variable, also called response. It must be a single column array of length (rows) equal to that of *rangeX1*. Acceptable values in this range are 1 (true) or 0 (false). This range also forms part of the training data set.
rangeX2	Cell range of the independent variable(s), used as the test data set. The number of columns must be equal to that of *rangeX1*. However, the number of rows (length) does not need to match.

Output

The output is a column array of which length (number of rows) is equal to that of *rangeX2*. The output is the probability of value 1 (true) in the dependent variable in response to the test data of the predictor(s) contained in *rangeX2*.

Remarks

See also LogisticCoeff and LogisticForecastMulti.

LogisticForecastMulti

Purpose

Forecast by “one vs all” multiclass logistic regression method.

Syntax

LogisticForecast(*rangeX1*, *rangeY1*, *rangeX2*, *boolFull*)

Input Arguments

rangeX1	Cell range of the independent variable(s), also called feature(s) or predictor(s). It may encompass several rows and columns. Different columns represent different features. This range forms part of the training data set.
rangeY1	Cell range containing the class labels. It must be a single column array of length (rows) equal to that of *rangeX1*. This range also forms part of the training data set. The possible values are integer numbers *0,1,⋯, M*. Each one of these numbers would encode a class or category. There should be no gap meaning that the numbers *0,1,⋯, M* must occur at least once.
rangeX2	Cell range of the independent variable(s) used as the test data set. The number of columns must be equal to that of *rangeX1*. However, the number of rows (length) does not need to match.
boolFull	If *TRUE*, the probabilities of each class are shown. If *FALSE*, only the most probable class is shown.

Output

When *boolFull* is *FALSE*, the output range has just 1 column. In this case, the number of rows is equal to that of *rangeX2* . However, when *boolFull* is *TRUE*, the number

of columns in the output is equal to the number of different class labels in the *rangeY* and the number of rows is equal to that of *rangeX2* + 1.

When *boolFull* is *FALSE*, the class with the largest forecast probability is picked and the corresponding label is shown at each row.

When *boolFull* is *TRUE*, at each row the forecast probabilities of the different classes are shown in detail. The probabilities are normalized (row sum equal to 1). The first row of the output shows the labels: they are ordered ascendingly from left to right.

Remarks

See also LogisticForecast and LogisticCoeffOneVsAll.

LogisticROC

Purpose

Estimates the forecast performance of the logistic regression. The generated table can be used for plotting the so-called Receiver Operating Characteristic (ROC) curve.

Syntax

LogisticROC(*rangeX1*, *rangeY1*, *rangeX2*, *rangeY2*, *intIntervals*)

Input Arguments

rangeX1	Cell range of the independent variable(s). It may have one or more columns. This range forms part of the training data set.
rangeY1	Cell range of the dependent variable. It must be a single column array of length equal to that of *rangeX1*. Acceptable values in this range are 1 or 0. This range also forms part of the training data set.
rangeX2	Cell range of the independent variable(s) of the test data set. The number of columns must be equal to that of *rangeX1*. However, the number of rows (length) does not need to match.
rangeY2	Cell range of the dependent variable of the test data set. It must be a single column array of length equal to that of *rangeX2*. Acceptable values in this range are 1 or 0.
intIntervals	Number of the intervals for the cutoff probability of the positive (1 or true) identification. It must be at least 10 or larger.

Output

The output is a matrix where the number of rows is equal to $intIntervals + 1$ and the number of columns is equal to 3. The columns correspond to cutoff, false positive rate

and true positive rate. For example, if $intInterval = 10$ the cutoffs would be 0, 0.1, 0.2, etc up to 1, such that there are 10 intervals.

Remarks

The "cutoff" refers to the forecast probability above which positive identification is made. The true positive rate is the ratio of the correctly identified positive values divided by the total of actual positive values of the dependent variable in the test set. The false positive rate can be calculated as $1 - true\ negative\ rate$. Here, the true negative rate is the ratio of the correctly identified negative values divided by the total of actual negative values of the dependent variable in the test set.

It can be noted that the test set may or may not overlap (or coincide) with the training set. One can train with one set and then test on another set and estimate the forecast performance (with ROC curve).

One can easily draw the ROC curve in Excel by inserting a "scatter plot" with the second and the third columns of the output table.

See also LogisticCoeff , LogisticConfMatrix and LogisticForecast.

MissingDataFill

Purpose

Replaces the missing or incorrect data using one of the three methods.

Syntax

MissingDataFill(*rangeX* , *intMode*)

Input Arguments

rangeX	Cell range of one or more columns containing data in which missing data (blank cells) or incorrect type values might be present. Although multi-column range is allowed, each column is considered as independent data set.
intMode	If equal to 0, the blank is replaced by the average of the valid data (within the same column). If equal to 1, the blank is replaced by a value randomly drawn from the valid data (within the same column). If equal to 2, the blank is replaced by the average of the neighboring cells (one cell above and once cell below in the same column).

Output

The output has the same dimension as the input *rangeX*. However, the invalid cells are now filled according to one of the three criteria as explained above.

Remarks

This function is useful for dealing with the missing data (blank cells) or incorrect type values. When *intMode* is 2, the invalid data should not occur neither in the first nor in the last row. See also MissingDataMask.

MissingDataMask

Purpose

Provides a mask for the missing or incorrect data.

Syntax

MissingDataMask(*rangeX*)

Input Arguments

rangeX	Cell range of one or more columns containing data in which missing data (blank cells) or incorrect type value might be present.

Output

The output has the same dimension as the input *rangeX*. For a blank cell, value 1 is placed. For a cell containing a number, value 0 is placed. For all the other cases (such as string, date, etc) value -1 is given.

Remarks

This function is useful for changing the format (color or font) of the cells according to the occurrence of the missing data (blank cells) or incorrect type values. See also MissingDataFill.

NetworkCommunityStruct

Purpose

Shows the community structure of a network by modularity analysis. Implements the simulated annealing algorithm for maximizing the modularity. Does not require the adjacency matrix.

Syntax

NetworkCommunityStruct(*rangeData*, *realFraction*, *intCommunity*, *intSeed*, *intRepeat*)

Input Arguments

rangeData	Cell range containing the data. The data can be interpreted as coordinates (vectors) of the nodes. These coordinates should be ordered as rows.
realFraction	When the "strength" of edges between different nodes is ranked according to the cosine similarity, this value specifies the upper fraction of the edges that are kept in the adjacency matrix.
intCommunity	Total number of communities that is targeted $N_{community}$.
intSeed	Seed for initializing the random number generator.
intRepeat	Determines the number of trial steps in the simulated annealing algorithm. At each given "temperature", there are $intRepeat \times N_{nodes}$ trial steps.

Output

The output is a column range of length N_{nodes}. Each element indicates the community (1 through $N_{community}$) to which the corresponding node belongs.

Remarks

This function implements the simulated annealing algorithm and maximizes the modularity (*M*). The modularity can be written in a compact form as following.

$$M = \frac{1}{2\,N_{edges}} Tr\left(\tilde{S}^t \tilde{B} \tilde{S}\right)$$

Here, the matrix $\tilde{S}$ has dimension $N_{nodes} \times N_{community}$ and contains the information on the community structure. The rows of $\tilde{S}$ represent the different nodes. For a given row, the community assignment is given by the column position of the element 1. All the other elements of the same row should be 0s.

The matrix $\tilde{B}$ has dimension $N_{nodes} \times N_{nodes}$. It is built from the adjacency matrix $\tilde{A}$. The elements of $\tilde{B}$ are given as following. Here, k_i denotes the node degree.

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2\,N_{edges}}$$

See also AdjMatrix , NetworkCommunityStructAdj and NetworkModularity.

NetworkCommunityStructAdj

Purpose

Shows the community structure of a network by modularity analysis. Implements the simulated annealing algorithm for maximizing the modularity. Requires the adjacency matrix.

Syntax

NetworkCommunityStructAdj(*rangeAdjMat*, *intCommunity*, *intSeed*, *intRepeat*)

Input Arguments

rangeAdjMat	Cell range containing the simple adjacency matrix of a network. It must be a transpose symmetric matrix of 0s and 1s.
intCommunity	Total number of communities that is targeted $N_{community}$.
intSeed	Seed for initializing the random number generator.
intRepeat	Determines the number of trial steps in the simulated annealing algorithm. At each given "temperature", there are $intRepeat \times N_{nodes}$ trial steps.

Output

The output is a column range of length N_{nodes}. Each element indicates the community (1 through $N_{community}$) to which the corresponding node belongs.

Remarks

This function implements the simulated annealing algorithm and maximizes the modularity (*M*). The modularity can be written in a compact form as following.

$$M = \frac{1}{2\,N_{edges}} Tr\left(\tilde{S}^t \tilde{B} \tilde{S}\right)$$

Here, the matrix $\tilde{S}$ has dimension $N_{nodes} \times N_{community}$ and contains the information on the community structure. The rows of $\tilde{S}$ represent the different nodes. For a given row, the community assignment is given by the column position of the element 1. All the other elements of the same row should be 0s.

The matrix $\tilde{B}$ has dimension $N_{nodes} \times N_{nodes}$. It is built from the adjacency matrix $\tilde{A}$. The elements of $\tilde{B}$ are given as following. Here, k_i denotes the node degree.

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2\, N_{edges}}$$

See also NetworkCommunityStruct.

NetworkModularity

Purpose

Calculates the maximum modularity of a network. Implements the simulated annealing algorithm for maximizing the modularity. Does not require the adjacency matrix.

Syntax

NetworkModularity(*rangeData*, *realFraction*, *intCommunity*, *intSeed*, *intRepeat*)

Input Arguments

rangeData	Cell range containing the data. The data can be interpreted as coordinates (vectors) of the nodes. These coordinates should be ordered as rows.
realFraction	When the "strength" of edges between different nodes is ranked according to the cosine similarity, this value specifies the upper fraction of the edges that are kept in the adjacency matrix.
intCommunity	Total number of communities that is targeted $N_{community}$.
intSeed	Seed for initializing the random number generator.
intRepeat	Determines the number of trial steps in the simulated annealing algorithm. At each given "temperature", there are $intRepeat \times N_{nodes}$ trial steps.

Output

The modularity is explained in the remarks section below. The output is the maximized modularity for a fixed number of possible communities. One should repeat the execution of this function varying the number of communities in order to obtain the global maximum.

Remarks

This function implements the simulated annealing algorithm and maximizes the modularity (*M*). The modularity can be written in a compact form as following.

$$M = \frac{1}{2\,N_{edges}} Tr\left(\tilde{S}^t \tilde{B} \tilde{S}\right)$$

Here, the matrix $\tilde{S}$ has dimension $N_{nodes} \times N_{community}$ and contains the information on the community structure. The rows of $\tilde{S}$ represent the different nodes. For a given row, the community assignment is given by the column position of the element 1. All the other elements of the same row should be 0s.

The matrix $\tilde{B}$ has dimension $N_{nodes} \times N_{nodes}$. It is built from the adjacency matrix $\tilde{A}$. The elements of $\tilde{B}$ are given as following. Here, k_i denotes the node degree.

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2\,N_{edges}}$$

See also AdjMatrix , NetworkModularityAdj and NetworkCommunityStruct.

NetworkModularityAdj

Purpose

Calculates the maximum modularity of a network represented by an adjacency matrix. Implements the simulated annealing algorithm for maximizing the modularity. Requires the adjacency matrix.

Syntax

NetworkModularityAdj(*rangeAdjMat*, *intCommunity*, *intSeed*, *intRepeat*)

Input Arguments

rangeAdjMat	Cell range containing the simple adjacency matrix of a network. It must be a transpose symmetric matrix of 0s and 1s.
intCommunity	Total number of communities that is targeted $N_{community}$.
intSeed	Seed for initializing the random number generator.
intRepeat	Determines the number of trial steps in the simulated annealing algorithm. At each given "temperature", there are $intRepea\square \times N_{nodes}$ trial steps.

Output

The modularity is explained in the remarks section below. The output is the maximized modularity for a fixed number of possible communities. One should repeat the execution of this function varying the number of communities in order to obtain the global maximum.

Remarks

This function implements the simulated annealing algorithm and maximizes the modularity (*M*). The modularity can be written in a compact form as following.

$$M = \frac{1}{2\,N_{edges}} Tr\left(\tilde{S}^t \tilde{B} \tilde{S}\right)$$

Here, the matrix $\tilde{S}$ has dimension $N_{nodes} \times N_{community}$ and contains the information on the community structure. The rows of $\tilde{S}$ represent the different nodes. For a given row, the community assignment is given by the column position of the element 1. All the other elements of the same row should be 0s.

The matrix $\tilde{B}$ has dimension $N_{nodes} \times N_{nodes}$. It is built from the adjacency matrix $\tilde{A}$. The elements of $\tilde{B}$ are given as following. Here, k_i denotes the node degree.

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2\,N_{edges}}$$

See also NetworkModularity.

OLSANOVA

Purpose

Performs the analysis of variance (ANOVA) of the linear regression. The ordinary least square method (OLS) is applied.

Syntax

OLSANOVA(*rangeX*, *rangeY*, *boolConst*)

Input Arguments

rangeX	Cell range of the independent variable(s), also called regressor(s) or predictor(s). It may encompass one or more columns. Each column represents one variable.
rangeY	Cell range of the dependent variable, also called response. It must be a single column array of length (rows) equal to that of *rangeX*.
boolConst	Whether to include the constant intercept or not. Its value can be either *TRUE* or *FALSE*.

Output

The output is a column array of length 7. Each element is as explained below.

1	Sum of squares of regression (SSR).
2	Sum of squares of errors (SSE).
3	Total sum of squares (SST).
4	Mean squares of regression (MSR).
5	Mean square of errors (MSE).
6	*F-statistic*.
7	*p*-value of the F distribution.

Remarks

Formulas used to calculate are as the following. Here, we use the notation: *N = number of the observations, K = number of the independent variables* and *P = number of the coefficients*. If the constant intercept is included, $P = K + 1$. If the constant intercept is not included, $P = K$.

$SSR = \sum_{i=1}^{N}(\bar{y} - \hat{y}_i)^2$.

$SSE = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{N}(e_i)^2$.

$SST = \sum_{i=1}^{N}(y_i - \bar{y})^2$.

$MSR = \frac{SSR}{K}$.

$MSE = \frac{SSE}{N-P}$.

$F - statistic = \frac{MSR}{MSE}$.

p-value = *F(F-statistic, K, N-P).*

OLSCoeff

Purpose

Calculates the regression coefficients by the ordinary least square method (OLS).

Syntax

OLSCoeff(*rangeX*, *rangeY*, *boolConst*)

Input Arguments

rangeX	Cell range of the independent variable(s), also called regressor(s) or predictor(s). It may encompass one or more columns. Each column represents one variable.
rangeY	Cell range of the dependent variable, also called response. It must be a single column array of length (rows) equal to that of *rangeX*.
boolConst	Whether to include the constant intercept or not. Its value can be either *TRUE* or *FALSE*.

Output

Here, we assume the following notation.

β_i	Regression coefficient.
$error_i$	Standard error.
$t-stat_i$	t-statistic.
$p-value_i$	*p*-value.

If *boolConst* is *FALSE*, the output is a $K \times 4$ array, where K is the number of independent variables. K is equal to the number of columns of *rangeX*.

$$\beta_1 \qquad error_1 \qquad t-stat_1 \qquad p-value_1$$

$$\begin{matrix} \beta_2 & error_2 & t-stat_2 & p-value_2 \\ \vdots & \vdots & \vdots & \vdots \\ \beta_K & error_K & t-stat_K & p-value_K \end{matrix}$$

If *boolConst* is *TRUE*, the output is a $(K+1)\times 4$ array, where $(K+1)$ is the number of independent variables plus one. The sub-index 0 represents the constant intercept.

$$\begin{matrix} \beta_0 & error_0 & t-stat_0 & p-value_0 \\ \beta_1 & error_1 & t-stat_1 & p-value_1 \\ \vdots & \vdots & \vdots & \vdots \\ \beta_K & error_K & t-stat_K & p-value_K \end{matrix}$$

Remarks

The linear regression assumes the following relationship between the independent variables and the dependent variable. Here, we consider *N* observations for each one of the *K* independent variables denoted as x_{ij} and correspondingly *N* observations of the dependent variable denoted as y_j. The sub-index $i = 1,2,\cdots,N$ and $j = 1,2,\cdots,K$. ε_i denote the errors.

$$\begin{matrix} \beta_0 + \beta_1 x_{11} + \beta_2 x_{12} + \cdots + \beta_K x_{1K} + \varepsilon_1 = y_1 \\ \beta_0 + \beta_1 x_{21} + \beta_2 x_{22} + \cdots + \beta_K x_{2K} + \varepsilon_2 = y_2 \\ \vdots \\ \beta_0 + \beta_1 x_{N1} + \beta_2 x_{N2} + \cdots + \beta_K x_{NK} + \varepsilon_N = y_N \end{matrix}$$

When we use a more compact notation with $\tilde{X} =$ *matrix of the independent variables*, $\vec{Y} =$ *column vector of the dependent variable*, $\vec{\beta} =$ *column vector of the coefficients* and $\vec{\varepsilon} =$ *disturbance vector*, we can express the above set of equations as the following.

$$\tilde{X}\vec{\beta} + \vec{\varepsilon} = \vec{Y}$$

$$\tilde{X} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1K} \\ 1 & x_{21} & \cdots & x_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & \cdots & x_{NK} \end{pmatrix} \quad , \quad \vec{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} \quad , \quad \vec{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_N \end{pmatrix} \quad , \quad \vec{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_K \end{pmatrix}$$

By applying the ordinary least square method (OLS), the solution is given by

$$\vec{\beta} = \left(\tilde{X}^t \tilde{X}\right)^{-1} \tilde{X}^t \tilde{Y}$$

OLSForecast

Purpose

Forecasts the values of the dependent variable using the linear regression method.

Syntax

OLSForecast(*rangeX1*, *rangeY1*, *boolConst*, *rangeX2*)

Input Arguments

rangeX1	Cell range of the independent variable(s), also called regressor(s) or predictor(s). It may encompass one or more columns. Each column represents one variable. This range forms part of the training data set.
rangeY1	Cell range of the dependent variable, also called response. It must be a single column array of length (rows) equal to that of *rangeX1*. This range also forms part of the training data set.
boolConst	Whether to include the constant intercept or not. Its value can be either TRUE or FALSE.
rangeX2	Cell range of the independent variable(s), used as the test data set. The number of columns must be equal to that of *rangeX1*. However, the number of rows does not need to match.

Output

The output is a column array of length (number of rows) equal to that of *rangeX2*. The output contains the predicted values of the dependent variable *Y* in response to the test data of the independent variable(s) contained in the *rangeX2*.

Remarks

See also OLSCoeff.

OLSLeverage

Purpose

Calculates the leverage and the Cook's distance of each observation by applying the linear regression method.

Syntax

OLSLeverage(*rangeX*, *rangeY*, *boolConst*, *intSelect*)

Input Arguments

rangeX	Cell range of the independent variable(s), also called regressor(s) or predictor(s). It may encompass one or more columns. Each column represents one variable.
rangeY	Cell range of the dependent variable, also called response. It must be a single column array of length (rows) equal to that of *rangeX*.
boolConst	Whether to include the constant intercept or not. Its value can be either *TRUE* or *FALSE*.
intSelect	If equal to 1, returns leverage. If equal to 2, returns Cook's distance.

Output

The output is a column array of which length is equal to that of *rangeY* (or the number of rows of *rangeX*). If *intSelect=1*, the output corresponds to the leverage while if *intSelect=2*, the output corresponds to the Cook's distance.

Remarks

The leverage of the *i-th* observation is defined as the diagonal element H_{ii} of the "Hat matrix" $\tilde{H} = \tilde{X}\left(\tilde{X}^t\tilde{X}\right)^{-1}\tilde{X}^t$. The leverage measures how much influence the observed predictor values have. As a requirement, the following sum rule should be satisfied. We denote *P = number of the coefficients* of the linear regression.

$$\sum_{i=1}^{N} H_{ii} = P$$

Thus, the leverage has, on the average, value$\approx \frac{P}{N}$. The leverage above $\frac{P}{N}$ means strong influence and the leverage below $\frac{P}{N}$ means weak influence.

The Cook's distance is another measure of the influence of each data point. The Cook's distance D_i is calculated using the following expression.

$$D_i = \frac{{e_i}^2}{P \times MSE}\left[\frac{H_{ii}}{(1-H_{ii})^2}\right]$$

For the definition of the mean square of errors (*MSE)*, refer to the function OLSANOVA.

OLSResiduals

Purpose

Calculates the residuals (errors) and the standardized (studentized) residuals of the linear regression.

Syntax

OLSResiduals(*rangeX*, *rangeY*, *boolConst*, *intSelect*)

Input Arguments

rangeX	Cell range of the independent variable(s), also called regressor(s) or predictor(s). It may encompass one or more columns. Each column represents one variable.
rangeY	Cell range of the dependent variable, also called response. It must be a single column array of length (rows) equal to that of *rangeX*.
boolConst	Whether to include the constant intercept or not. Its value can be either *TRUE* or *FALSE*.
intSelect	If equal to 1, returns residuals. If equal to 2, returns standardized residuals.

Output

The output is a column array of which length is equal to that of *rangeY* (or the number of rows of *rangeX*). If *intSelect=1*, the output is the simple residual while if *intSelect=2*, the output is the standardized residual.

Remarks

Residual (or error) of the *i-th* observation is simply $e_i = y_i - \hat{y}_i$, where y_i is the observed value of the dependent variable and $\hat{y}_i$ is the estimation by linear regression.

Standardized or studentized residual (internally studentized residual) is calculated using the following expression.

$$Standardized\ e_i = \frac{e_i}{\sigma_e\sqrt{1 - H_{ii}}}$$

Here, H_{ii} is a diagonal element of the "Hat matrix" $\tilde{H} = \tilde{X}\left(\tilde{X}^t\tilde{X}\right)^{-1}\tilde{X}^t$ and $\sigma_e = \sqrt{MSE}$. For the definition of *MSE*, refer to the function OLSANOVA.

OLSStat

Purpose

Calculates various measures of the linear regression by applying the ordinary least square method (OLS).

Syntax

OLSStat(*rangeX*, *rangeY*, *boolConst*)

Input Arguments

rangeX	Cell range of the independent variable(s), also called regressor(s) or predictor(s). It may encompass one or more columns. Each column represents one variable.
rangeY	Cell range of the dependent variable, also called response. It must be a single column array of length (rows) equal to that of *rangeX*.
boolConst	Whether to include the constant intercept or not. Its value can be either *TRUE* or *FALSE*.

Output

The output is a column array of length 7. Each element is as explained below.

1	R^2.
2	*Adjusted* R^2.
3	*Standard Error of Regression (SER).*
4	*Durbin-Watson statistic.*
5	*Log likelihood.*
6	*Akaike Information Criterion (AIC).*
7	*Bayesian Information Criterion (BIC).*

Remarks

Formulas used to calculate are as the following. Here, we denote with N = *number of the observations*, K = *number of the independent variables* and P = *number of the coefficients*. If the constant intercept is included, $P = K + 1$. If the constant intercept is not included, $P = K$. Bayesian Information Criterion (BIC) is also known as Schwarz Information Criterion (SIC).

When the constant intercept is included, then the "centered" R^2 is used.

$R^2 = 1 - \frac{SSE}{SST}$, with $SSE = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{N}(e_i)^2$

and $SST = \sum_{i=1}^{N}(y_i - \bar{y})^2$.

When the constant intercept is not included, then the "uncentered" R^2 is used.

$R^2 = \frac{\sum_{i=1}^{N} \hat{y}_i{}^2}{\sum_{i=1}^{N} y_i{}^2}$

Adjusted $R^2 = 1 - \frac{N-1}{N-K-1}(1 - R^2)$.

Standard Error of Regression (SER)$= \frac{SSE}{N-P}$.

Durbin-Watson statistic$= \frac{\sum_{i=2}^{N}(e_i - e_{i-1})^2}{\sum_{i=1}^{N}(e_i)^2}$.

Log likelihood$= -\frac{N}{2}\left(1 + Ln(2\pi) + Ln\left(\frac{SSE}{N}\right)\right)$.

Akaike Information Criterion (AIC)$= -2\frac{Log\ likelihood}{N} + 2\frac{P}{N}$.

Bayesian Information Criterion (BIC)$= -2\frac{Log\ likelihood}{N} + p\frac{Ln(N)}{N}$.

OLSTest

Purpose

Tests the forecast of linear regression by the ordinary least square method (OLS). Implements the validation model.

Syntax

OLSTest(*rangeX1*, *rangeY1*, *boolConst*, *rangeX2*, *rangeY2*)

Input Arguments

rangeX1	Cell range of the independent variable(s), also called regressor(s) or predictor(s). It may encompass one or more columns. Each column represents one variable.
rangeY1	Cell range of the dependent variable, also called response. It must be a single column array of length (rows) equal to that of *rangeX1*.
boolConst	Whether to include the constant intercept or not. Its value can be either *TRUE* or *FALSE*.
rangeX2	Cell range of the independent variable(s) of the test data set. The number of columns must be equal to that of *rangeX1*. However, the number of rows (length) does not need to match.
rangeY2	Cell range of the dependent variable of the test data set. It must be a single column array of length equal to that of *rangeX2*.

Output

The output is a column array of length 3. Each element is as explained below.

1	*SSE*

2	*MSE.*
3	R^2.

Remarks

Formulas used to calculate are as the following. Here, we denote with *N = number of the observations*, *K = number of the independent variables* and *P = number of the coefficients*. If the constant intercept is included, $P = K + 1$. If the constant intercept is not included, $P = K$.

$$SSE = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{N}(e_i)^2$$

$$MSE = \frac{SSE}{N-P}$$

When the constant intercept is included, then the "centered" R^2 is used.

$$R^2 = 1 - \frac{SSE}{SST}, \quad \text{with } SSE = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{N}(e_i)^2$$

$$\text{and } SST = \sum_{i=1}^{N}(y_i - \bar{y})^2.$$

When the constant intercept is not included, then the "uncentered" R^2 is used.

$$R^2 = \frac{\sum_{i=1}^{N}\hat{y}_i{}^2}{\sum_{i=1}^{N}y_i{}^2}$$

It can be noted that the test set may or may not overlap (or coincide) with the training set. One can train with one set and then test on another set.

OutliersGet

Purpose

Extracts the outliers from the sample.

Syntax

OutliersGet(*rangeX*, *intMode*, *intN*)

Input Arguments

rangeX	A single column cell range containing the random samples.
intMode	Set to 0 in order to get the outliers from the extreme low. Set to 1 in order to get the outliers from the extreme high. Set to 2 in order to get the outliers from both extremes.
intN	Equal to the number of the outliers for extraction when *intMode* = *0* or *1*. When *intMode* = *2* the number of the extracted outliers is actually $2 \times intN$, because equal number of outliers is extracted from both extremes.

Output

Extracted outliers are presented as a column array. The ordering follows the original occurrence (not sorted).

Remarks

As the number of outliers for extraction is specified by *intN*, in order to specify in percentage (denoted as α) we should calculate first $intN = \alpha \times N_s$ where N_s is the total number of the random samples. See also OutliersTrim.

OutliersTrim

Purpose

Trims the outliers from the sample.

Syntax

OutliersTrim(*rangeX*, *intMode*, *intN*)

Input Arguments

rangeX	A single column cell range containing the random samples.
intMode	Set to 0 in order to trim the outliers from the extreme low. Set to 1 in order to trim the outliers from the extreme high. Set to 2 in order to trim the outliers from both extremes.
intN	Equal to the number of the outliers for trimming when *intMode* = *0* or *1*. When *intMode* = *2* the number of the trimmed outliers is actually $2 \times intN$, because equal number of outliers is trimmed from both extremes.

Output

The output consists of the original samples excluding the trimmed ones. The ordering follows the original occurrence (not sorted).

Remarks

As the number of the outliers for trimming is specified by *intN*, in order to specify in percentage (denoted as α) we should calculate first $intN = \alpha \times N_s$ where N_s is the total number of the random samples. See also OutliersGet.

PCCommunality

Purpose

Calculates the final communalities for a reduced dimension by principal component analysis.

Syntax

PCCommunality(*rangeX*, *intN*)

Input Arguments

rangeX	Cell range containing the data. The data can be interpreted as coordinates that are ordered as rows. Columns correspond to different variables or features.
intN	Number of the principal components in the basis set. It must be an integer number equal or larger than 1 and not exceed the number of variables (features).

Output

The output is a row vector of length equal to $N_{features}$, the number of variables (columns). The final communality is the fraction of variance of each input variable that is accounted when the dimension of the basis set is reduced to *intN*. When *intN* is less than $N_{features}$, the final communality is less than 1 as there is loss of "fidelity".

Remarks

The input is always standardized. See also PCInputReduced, PCLoads, PCLoadsReduced and PCScores.

PCInputReduced

Purpose

Represents the input data with a reduced set of the principal axes.

Syntax

PCInputReduced(*rangeX*, *intN*)

Input Arguments

rangeX	Cell range containing the data. The data can be interpreted as coordinates that are ordered as rows. Columns correspond to different variables or features.
intN	Number of the principal components in the basis set. It must be an integer number equal or larger than 1 and not exceed the number of variables (features).

Output

The output cell range has the same dimension as the input cell range. However, each row of the output range is a representation of the input data point using a reduced set of the principal axes. When *intN* is equal to the total number of variables (features), the original data points are represented without loss of "fidelity". For any other value of *intN* (less than the number of variables), some fidelity is lost.

Remarks

Although the principal axes are calculated from the standardized input, the output is in the original non-standardized format.

See also PCCommunality, PCLoads, PCLoadsReduced and PCScores.

PCLoads

Purpose

Calculates the principal axes (loadings) for the standardized input.

Syntax

PCLoads(*rangeX*, *boolStd*)

Input Arguments

rangeX	Cell range containing the data. The data can be interpreted as coordinates that are ordered as rows. Columns correspond to different variables or features.

Output

If we denote as $N_{features}$ the number variables (columns) in the input data, the output is a matrix of dimension $N_{features} \times N_{features}$. The principal axes (loadings) are given as column vectors. When *boolStd* is *FALSE*, the principal axes are the eigenvectors of the covariance matrix. They are ordered according to their variances (eigenvalues) in a descending order from left to right. When *boolStd* is *TRUE*, the principal axes are the eigenvectors of the correlation matrix.

Remarks

The principal axes as given here are normalized vectors (modulus = 1). The input is always standardized.

See also PCLoadsReduced, PCScores and PCVariance.

PCLoadsReduced

Purpose

Calculates the reduced dimension principal axes (loadings) for the standardized input.

Syntax

PCLoadsReduced(*rangeX*, *intN*)

Input Arguments

rangeX	Cell range containing the data. The data can be interpreted as coordinates that are ordered as rows. Columns correspond to different variables or features.
intN	Number of the principal components in the basis set. It must be an integer number equal or larger than 1 and not exceed the number of variables (features).

Output

This function is analogous to the PCLoads. However, this function allows the calculation of a reduced set of principal axes. Here, the input is always standardized.

The dimension of the output is $intN \times N_{features}$. Thus, the principal axis coordinates are given as row vectors (not as column vectors as in PCLoads). They are ordered according to their variance in a descending order from the top to the bottom.

Remarks

The principal axes as given here are normalized vectors (modulus = 1).

See also PCLoads.

PCScores

Purpose

Calculates the transformed scores: projections of the original coordinates onto the principal component axes. The input is always standardized.

Syntax

PCScores(*rangeX*, *boolStd*)

Input Arguments

rangeX	Cell range containing the data. The data can be interpreted as coordinates that are ordered as rows. Columns correspond to different variables or features.

Output

The output cell range has the same dimension as the input cell range. Now, each row of the output range corresponds to the rotated vector of which elements are the projections of the original coordinates onto the principal axes.

Remarks

For a vector representing the coordinates of the standardized input data point $\vec{x}$ the projection onto the principal axis $\vec{y_i}$ is given by the following inner product (dot product). Here, we remind ourselves that the modulus of $\vec{y_i}$ is equal to 1: $|\vec{y_i}| = 1$.

$$Projection_i = \vec{x}, \vec{y_i}$$

PCVariance

Purpose

Calculates the variances associated with the principal axes. The input is always standardized.

Syntax

PCVariance(*rangeX*, *boolStd*)

Input Arguments

rangeX	Cell range containing the data. The data can be interpreted as coordinates that are ordered as rows. Columns correspond to different variables or features.

Output

If we denote as $N_{features}$ the number variables (columns) in the input data, the output is a matrix of dimension $3 \times N_{features}$. The first row of the output contains the variances associated with the principal axes ordered in a descending order from left to right. The second row contains the proportions with respect to the total sum. The third row contains the cumulative proportions.

Remarks

The input is always standardized. See also PCLoads and PCScores.

RandomBernoulli

Purpose

Generates Bernoulli random numbers.

Syntax

RandomBernoulli(*realP, intSample, intVariable, intSeed*)

Input Arguments

realP	Parameter p (success probability) of the Bernoulli random variable. It's value must lie in the interval $[0,1]$.
intSample	Number of the samples per random variable. This is equal to the number of rows in the output.
intVariable	Number of the random variables. This is equal to the number of columns in the output.
intSeed	Seed for initializing the random number generator.

Output

The output cell range has dimension equal to $intSample \times intVariable$.

Remarks

Bernoulli random variable is an example of the discrete random variable. A Bernoulli random number can be a value either 1 or 0. The probability of occurrence of 1 (success) is equal to the parameter p. Conversely, the probability of occurrence of 0 (failure) is equal to $(1-p)$. The parameter p must lie in the interval $[0,1]$. See also RandomBinomial.

RandomBinomial

Purpose

Generates binomial random numbers.

Syntax

RandomBinomial(*realP, intN, intSample, intVariable, intSeed*)

Input Arguments

realP	Success probability (p) of each trial. It's value must lie in the interval $[0,1]$.
intN	Number of trials *N*.
intSample	Number of the samples per random variable. This is equal to the number of rows in the output.
intVariable	Number of the random variables. This is equal to the number of columns in the output.
intSeed	Seed for initializing the random number generator.

Output

The output cell range has dimension equal to $intSample \times intVariable$.

Remarks

Binomial random variable is the sum of the Bernoulli trials. Let's consider a Binomial random variable composed of *N* Bernoulli trials X_i. Then

$$X_{Binomial} = \sum_{i=1}^{N} X_i$$

Each Bernoulli trial can take a value 1 or 0. The probability of occurrence of 1 (success) is equal to the parameter p. Conversely, the probability of occurrence of 0 (failure) is equal to $(1-p)$. See also RandomBernoulli.

RandomCauchy

Purpose

Generates standard Cauchy random numbers.

Syntax

RandomCauchy(*intSample*, *intVariable*, *intSeed*)

Input Arguments

intSample	Number of the samples per random variable. This is equal to the number of rows in the output.
intVariable	Number of the random variables. This is equal to the number of columns in the output.
intSeed	Seed for initializing the random number generator.

Output

The output cell range has dimension equal to $intSample \times intVariable$.

Remarks

Cauchy random variable (Z_{Cauchy}) is equal to the ratio of two normal random variables *X* and *Y*, both of mean zero and of a given shape parameter (standard deviation).

$$Z_{Cauchy} = \frac{X}{Y}$$

The standard Cauchy random variable coincides with the Student's t random variable of degree 1. See also RandomStudentT.

RandomChiSqr

Purpose

Generates Chi-squared (χ^2) random numbers.

Syntax

RandomChiSqr(*intDegree, intSample, intVariable, intSeed*)

Input Arguments

intDegree	Degree of freedom *k* of the Chi-squared random variable.
intSample	Number of the samples per random variable. This is equal to the number of rows in the output.
intVariable	Number of the random variables. This is equal to the number of columns in the output.
intSeed	Seed for initializing the random number generator.

Output

The output cell range has dimension equal to $intSample \times intVariable$.

Remarks

Chi-squared random variable (X_{ChiSqr}) of degree *k* is equal to the sum of squares of *k* standard normal variables X_i.

$$X_{ChiSqr} = \sum_{i=1}^{k} (X_i)^2$$

See also RandomNormal.

RandomLognormal

Purpose

Generates lognormal random numbers.

Syntax

RandomLognormal(*realDrift*, *realSigma*, *intSample*, *intVariable*, *intSeed*)

Input Arguments

realDrift	Drift parameter (μ).
realSigma	Volatility parameter (σ). It must be a positive number.
intSample	Number of the samples per random variable. This is equal to the number of rows in the output.
intVariable	Number of the random variables. This is equal to the number of columns in the output.
intSeed	Seed for initializing the random number generator.

Output

The output cell range has dimension equal to $intSample \times intVariable$.

Remarks

Lognormal random variable X can be related to a normal random variable Y as following.

$$X = e^Y$$

Here, Y is characterized by μ and σ.

RandomNormal

Purpose

Generates normal (Gauss) random numbers.

Syntax

RandomNormal(*realMean, realSigma, intSample, intVariable, intSeed*)

Input Arguments

realMean	Value of the mean (μ). Equal to 0 for the standard normal.
realSigma	Value of the shape parameter or standard deviation (σ). Equal to 1 for the standard normal. It must be a positive number.
intSample	Number of the samples per random variable. This is equal to the number of rows in the output.
intVariable	Number of the random variables. This is equal to the number of columns in the output.
intSeed	Seed for initializing the random number generator.

Output

The output cell range has dimension equal to $intSample \times intVariable$.

Remarks

Two parameters μ and σ fully describe this random variable. Normal distribution is symmetric (zero skewness) with zero excess kurtosis. See also RandomChiSqr.

RandomPoisson

Purpose

Generates Poisson random numbers.

Syntax

RandomPoisson(*realLambda*, *intSample*, *intVariable*, *intSeed*)

Input Arguments

realLambda	Parameter lambda (λ). It is equal to the mean as well as the variance of the distribution. It must be a positive number.
intSample	Number of the samples per random variable. This is equal to the number of rows in the output.
intVariable	Number of the random variables. This is equal to the number of columns in the output.
intSeed	Seed for initializing the random number generator.

Output

The output cell range has dimension equal to $intSample \times intVariable$.

Remarks

Poisson random variable is a discrete random variable. Possible values are restricted to the non-zero integers.

The only parameter is lambda (λ) which is equal to the mean as well as the variance of the distribution. For large values of λ, the Poisson distribution converges to the normal distribution with mean $\mu = \lambda$ and standard deviation $\sigma = \sqrt{\lambda}$.

RandomStudentT

Purpose

Generates Student's t random numbers.

Syntax

RandomStudentT(*intDegree*, *intSample*, *intVariable*, *intSeed*)

Input Arguments

intDegree	Degree (ν) of the Student's t random variable. It must be a positive integer number.
intSample	Number of the samples per random variable. This is equal to the number of rows in the output.
intVariable	Number of the random variables. This is equal to the number of columns in the output.
intSeed	Seed for initializing the random number generator.

Output

The output cell range has dimension equal to $intSample \times intVariable$.

Remarks

Student's t distribution is symmetric and bell-shaped, just like the normal distribution. However, it has heavier tails than the normal distribution. When the degree is 1, Student's t coincides with the standard Cauchy. When the degree is infinite, it converges to the standard normal random variable.

RandomUniform

Purpose

Generates uniform random numbers.

Syntax

RandomUniform(*realMin*, *realMax*, *intSample*, *intVariable*, *intSeed*)

Input Arguments

realMin	Lower limit of the random number.
realMax	Upper limit of the random number.
intSample	Number of the samples per random variable. This is equal to the number of rows in the output.
intVariable	Number of the random variables. This is equal to the number of columns in the output.
intSeed	Seed for initializing the random number generator.

Output

The output cell range has dimension equal to $intSample \times intVariable$.

Remarks

This function generates continuous random numbers bounded by the lower and the upper limits.

RangeFlip

Purpose

Flips a cell range if the condition is true.

Syntax

RangeFlip(*rangeX*, *boolFlip*)

Input Arguments

rangeX	Cell range containing numbers without blanks. It could be a multi-column range.
boolFlip	If equal to *TRUE*, flips the cell range.

Output

The output cell range has the same dimension as *rangeX*. The *rangeX* must contain numbers without blanks.

Remarks

A time series can be presented on a spreadsheet in such a way that the top most value is the most recent one. This is the case of ascending time ordering. In the reverse case, we say that the time series is in descending time ordering.

This function is useful for reverting the time ordering by flipping the chosen cell range such that the previously top most values now goes to the bottom of the column (and vice versa).

TableHisto

Purpose

Generates a table of frequencies for the purpose of plotting a histogram.

Syntax

TableHisto(*rangeX*, *intBins*, *intCurveSelect*, *strLabelMask*)

Input Arguments

rangeX	A single or a multi-column cell range containing data.
intBins	Number of the bins. All the bins have the same size.
intCurveSelect	Type of the curve overlay. If equal to 0, there is no curve overlay. If equal to 1, "cumulative probability overlay" and if equal to 2, "cumulative percentage overlay" is shown.
strLabelMask	Mask for the label. For example "F3" means float with 3 decimal digits like 0.123. Another example "E2" is scientific notation with 2 digit precision like 1.23 E+1234.

Output

The output can be a two or three column matrix. The first column shows the bin labels. These labels are numbers formatted according to *strLabelMask* and then converted to the string type. The second column corresponds to the frequencies. Finally, the optional third column is shown when the value of *intCurveSelect* is *1* or *2*. When *intCurveSelect=1*, the "cumulative probability" overlay is shown. When *intCurveSelect=2*, the "cumulative percentage" overlay is shown.

Remarks

Each bin is defined by its upper and lower limits. The labels' values coincide with the upper limits of the bins.

TableQQGaussPlot

Purpose

Generates a table of ordered statistics for the purpose of making Quantile-Quantile plot (Q-Q Plot) that compares data vs Gaussian distribution.

Syntax

TableQQGaussPlot(*rangeX*)

Input Arguments

rangeX	Cell range of the sample data set. It must be a single column array.

Output

The output is a two column matrix. The first column corresponds to the ordered statistics of the input data set. The second column is the ordered statistics of the Gaussian distribution. For the Gaussian, zero mean and unit standard deviation are assumed (standard normal distribution). Q-Q Plot of "data vs theory" can be made from this output.

Remarks

For other types of Q-Q Plot see TableQQPlot and TableQQSTTPlot.

TableQQPlot

Purpose

Generates a table of ordered statistics for the purpose of making Quantile-Quantile plot (Q-Q Plot) from the data samples.

Syntax

TableQQPlot(*rangeX1*, *rangeX2*)

Input Arguments

rangeX1	Cell range of the sample data set #1. It must be a single column array.
rangeX2	Cell range of the sample data set #2. It must be a single column array. The length must coincide with that of *rangeX1*.

Output

The output is a two column matrix. The columns correspond to the ordered statistics of the input data set. Q-Q Plot of “data vs data” can be made from this output.

Remarks

For Q-Q Plot of “data vs theory” see TableQQGaussPlot and TableQQSTTPlot.

TableQQSTTPlot

Purpose

Generates a table of ordered statistics for the purpose of making Quantile-Quantile plot (Q-Q Plot) that compares data vs Student's t distribution.

Syntax

TableQQSTTPlot(*rangeX*, *intDegree*)

Input Arguments

rangeX	Cell range of the sample data set. It must be a single column array.
intDegree	Degree of freedom of the Student's t distribution.

Output

The output is a two column matrix. The first column corresponds to the ordered statistics of the input data set. The second column contains the ordered statistics of the Student's t distribution of degree *intDegree*. *intDegree* must be an integer value equal or larger than 1. Q-Q Plot of "data vs theory" can be made from this output.

Remarks

For other types of Q-Q Plot see TableQQPlot and TableQQGaussPlot.

TSACF

Purpose

Calculates the Auto-Correlation Function (ACF) of a stationary time series.

Syntax

TSACF(*rangeTS*, *intL*)

Input Arguments

rangeTS	Cell range of a stationary time series. It must be a single column array.
intL	Lag which can be 0 or larger integer number.

Output

The output is a column array with the elements corresponding to the ACF. Here, the ACF is denoted as $\rho(\ell)$, where ℓ represents positive lag (including 0).

Remarks

A time series is a set of time ordered observations of a random process. Here, we assume weak stationarity. This means that the mean and the standard deviation are constant.

The ACF, in principle, can depend on the specific positions in time as $\rho(t_1, t_2)$. However, under the assumption of the weak stationarity $\rho(t_1, t_2) = \rho(|t_1 - t_2|) = \rho(\ell)$ with the definition $\ell = |t_1 - t_2|$. Thus, ACF is function only of the relative distance in time. The value of $\rho(\ell)$ must lie between -1 and 1. ACF equal to 1 is the perfect correlation and ACF equal to -1 is the perfect anti-correlation. ACF equal to 0 means no correlation. $\rho(0) = 1$ because, a data sample is always perfectly correlated with itself. The following formula is used, where *T* is the total length of the time series and μ is the mean of the time series.

$$\rho(\ell) = \frac{\sum_{t=\ell+1}^{T}(x_t - \mu)(x_{t-\ell} - \mu)}{\sum_{t=1}^{T}(x_t - \mu)^2}$$

TSACFAR

Purpose

Calculates the Auto-Correlation Function (ACF) using the Auto-Regressive (AR) parameters.

Syntax

TSACFAR(*rangeAR*)

Input Arguments

rangeAR	Cell range containing the AR parameters: $\phi_0, \phi_1, \ldots, \phi_p$. It must be a single column or a single row array of length $p + 1$.

Output

The output is a column array with the elements corresponding to the ACF. In principle, the ACF of an AR model extends to the infinite lag. However, here the first 30 lags are calculated (starting from the zero lag). When the parameters do not satisfy the stationarity condition, an array containing zeroes is returned.

Remarks

Internally, this function first converts the AR model to the Moving Average (MA) model and then calculates the auto-correlations. Thus, the parameters $\phi_0, \phi_1, \ldots, \phi_p$ must represent stationary time series. See TSARCharRoots about the characteristic roots and the stationarity condition. See also TSARtoMA and TSARIMAFit.

TSACFARMA

Purpose

Calculates the Auto-Correlation Function (ACF) using the Auto-Regressive Moving Average (ARMA) parameters.

Syntax

TSACFARMA(*rangeAR*, *rangeMA*)

Input Arguments

rangeAR	Cell range containing the AR parameters: $\phi_0, \phi_1, \ldots, \phi_p$. It must be a single column or a single row array of length $p + 1$.
rangeMA	Cell range containing the MA parameters: $\theta_1, \theta_2, \ldots, \theta_q$. It must be a single column or a single row array of length q.

Output

The output is a column array with the elements corresponding to the ACF. In principle, the ACF of an ARMA model extends to the infinite lag. However, here the first 30 lags are calculated (starting from the zero lag). When the parameters do not satisfy the stationarity condition, an array containing zeroes is returned.

Remarks

Internally, this function first converts the ARMA model to the Moving Average (MA) model and then calculates the auto-correlations. Thus, the AR parameters $\phi_0, \phi_1, \ldots, \phi_p$ must represent stationary time series. See TSARCharRoots about the characteristic roots and the stationarity condition. See also TSARMAtoMA and TSARIMAFit.

TSACFMA

Purpose

Calculates the Auto-Correlation Function (ACF) using the Moving Average (MA) parameters.

Syntax

TSACFMA(*rangeMA*)

Input Arguments

rangeMA	Cell range containing the MA parameters: $\mu, \theta_1, \theta_2, \ldots, \theta_q$. It must be a single column or a single row array of length $q+1$. Notice that the parameter list starts from the constant average μ.

Output

The output is a column array with the elements corresponding to the ACF. Unlike the AR and the ARMA, the MA has a sharp cutoff in the ACF: for $\ell > q$, the auto-correlation $\rho(\ell) = 0$. Thus, the same cutoff is imposed to the output of this function. Including the zero lag, the total length of the output is *q+1*.

Remarks

There is no need to check for the stationarity of the MA model as long as the parameters θ_i are square summable ($\sum_{i=0}^{\infty} {\theta_i}^2 < \infty$). Strictly speaking, the mean μ is not required for calculating the ACF. However, it is required in the input list in order to maintain the consistency in the format. The ACF is calculated using the following formula, where we introduce a new constant defined as $\theta_0 = 1$.

$$\rho(\ell) = \frac{\sum_{k=0}^{q-\ell} \theta_k \theta_{k+\ell}}{\sum_{k=0}^{q} {\theta_k}^2}$$

TSACFTest

Purpose

Tests the significance of the components of the Auto-Correlation Function (ACF) of a stationary time series.

Syntax

TSACFTest(*rangeTS*, *intMaxLag*)

Input Arguments

rangeTS	Cell range of a stationary time series. It must be a single column array.
intMaxLag	Maximum lag.

Output

The output consists of the test z-statistics and the *p*-values of the ACF (denoted as $\rho(\ell)$) starting from $\ell = 1$ up to $\ell = intMaxLag$. Thus, the output is a two column matrix of length (rows) equal to $intMaxLag$.

$$\begin{matrix} z-statistic_1 & p-value_1 \\ z-statistic_2 & p-value_2 \\ \vdots & \vdots \\ z-statistic_{Max} & p-value_{Max} \end{matrix}$$

Remarks

The test statistics are calculated using the following formula. Here, *T* denotes the total length of the time series. *P*-values are obtained from the normal distribution.

$$z-statistic_\ell = \frac{\rho(\ell)}{\sqrt{\frac{1+2\sum_{m=1}^{\ell-1}\rho(m)^2}{T}}}$$

TSARCharRoots

Purpose

Calculates the characteristic roots using the AR parameters.

Syntax

TSARCharRoots(*rangeAR*)

Input Arguments

rangeAR	Cell range of the AR parameters $\phi_0, \phi_1, \phi_2, \cdots, \phi_p$. It must be a single column or a single row array. There is no need to specify the number of parameters (denoted as p) since it is implicitly given by the range size. ϕ_i are the parameters of the AR(p) model. Notice that the parameter range must start from the constant intercept ϕ_0 (and not from ϕ_1).

Output

The output consists of the roots of the characteristic polynomial and their moduli. Since the roots can be complex valued, the first column shows the real part while the second column shows the imaginary part. The third column corresponds to the modulus. Thus the output is a matrix with 3 columns and the number of rows equal to the maximum power of the characteristic polynomial.

Remarks

AR(p) assumes the following model for the time series.

$$x_t = \varepsilon_t + \phi_0 + \sum_{i=1}^{p} \phi_i x_{t-i}$$

The characteristic roots are the inverse of the solutions of the following equation.

$$1 - \sum_{i=1}^{p} \phi_i x^i = 0$$

where the left hand side of the equation ($1 - \sum_{i=1}^{p} \phi_i x^i$) is the so-called "characteristic polynomial".

Notice that the parameter ϕ_0, while necessary for specifying the AR(*p*) model, is not really necessary for the calculation of the characteristic roots. However, in order to maintain consistency, this function requires full parameter list (*rangeAR*) including ϕ_0.

A stationary time series has the characteristic roots that are less than 1 in modulus. Complex roots, if present, come in pairs: if z_0 is a complex root, then its complex conjugate $\bar{z_0}$ is also a root. The presence of the complex roots signals the cyclic behavior of the time series.

TSARCHFit

Purpose

Calculates the parameters of Auto-Regressive Conditional Heteroskedasticity ARCH(*p*) model.

Syntax

TSARCHFit(*rangeTS*, *boolAscend*, *intP*)

Input Arguments

rangeTS	Cell range of a univariate time series (example: financial return). It must be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
intP	Number of the α_i parameters (*p*). It must be equal or larger than 1.

Output

The output consists of the parameter estimates as well as the corresponding errors ($error_i$). Thus, the output is a two column matrix of length (rows) equal to $p+1$. The ARCH parameters are denoted as $\alpha_0, \alpha_1, \cdots, \alpha_p$.

$$\begin{matrix} \alpha_0 & error_0 \\ \alpha_1 & error_1 \\ \vdots & \vdots \\ \alpha_p & error_P \end{matrix}$$

Remarks

This is the so-called symmetric normal ARCH model. Hence, the errors (ε_t) are assumed to obey the normal distribution of mean zero and time varying conditional variance: $\varepsilon_t \sim N(0, \sigma_t^2)$.

ARCH(p) model assumes the following serial dependence of the volatility.

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2$$

The ARCH parameters must obey the following constraints.

$$\alpha_0 > 0, \quad \alpha_i \geq 0, \quad \sum_{i=1}^{p} \alpha_i < 1$$

The unconditional mean of ε_t^2 (long run prediction of σ_t^2) is given by the following expression.

$$E[\varepsilon^2] = \frac{\alpha_0}{1 - \left(\sum_{i=1}^{p} \alpha_i\right)}$$

For the GARCH(p,q) model refer to TSGARCHFit.

TSARCHForecast

Purpose

Applies ARCH(*p*) model to a time series and forecasts volatility.

Syntax

TSARCHForecast(*rangeTS, boolAscend, intP, intN, intInterval, intSamples*)

Input Arguments

rangeTS	Cell range of a univariate time series (example: financial return). It must be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
intP	Number of the α_i parameters (*p*).
intN	Number of the forecast steps (*N*).
intInterval	Choice of the confidence interval. 0 for no interval, 1 for 68% interval, 2 for 95% interval and 3 for 99.7% interval.
intSamples	Number of samples for estimating the confidence interval by Monte Carlo method.

Output

When *intInterval = 0*, the output is a single column array of length $N + 1$. When *intInterval* is non zero, the output is a three column matrix of size $(N + 1) \times 3$. The columns correspond to the lower limit, middle point and the upper limit of the forecast. The step "0" of the output is equal to the most recent volatility of the input data. The output time ordering is always descendent.

Remarks

Suppose we have the time series data (X_t) up to the step t with the corresponding ε_t^2 and σ_t^2. If we would like to forecast the step $t+1$ (we denote the predicted value as $\hat{\sigma}_{t+1}^2$), we apply the following expression.

$$\hat{\sigma}_{t+1}^2 = \alpha_0 + \alpha_1\, \varepsilon_t^2 + \alpha_2\, \varepsilon_{t-1}^2 + \alpha_3\, \varepsilon_{t-2}^2 + \cdots + \alpha_p\, \varepsilon_{t-p+1}^2$$

In order to predict the next step $t+2$, notice that the expected values $\hat{\varepsilon}_{t+1} = E[\varepsilon_{t+1}] = 0$ and $\hat{\varepsilon}_{t+1}^2 = E[\varepsilon_{t+1}^2] = \hat{\sigma}_{t+1}^2$. Thus, we have the following.

$$\hat{\sigma}_{t+2}^2 = \alpha_0 + \alpha_1 \hat{\sigma}_{t+1}^2 + \alpha_2\, \varepsilon_t^2 + \alpha_3\, \varepsilon_{t-1}^2 + \cdots + \alpha_p\, \varepsilon_{t-p+2}^2$$

Analogously, for the step $t+3$, we have,

$$\hat{\sigma}_{t+3}^2 = \alpha_0 + \alpha_1 \hat{\sigma}_{t+2}^2 + \alpha_2\, \hat{\sigma}_{t+1}^2 + \alpha_3\, \varepsilon_t^2 + \cdots + \alpha_p\, \varepsilon_{t-p+3}^2$$

And so on. For a large number of steps $h \to \infty$, the prediction converges to the unconditional mean of ε_t^2 given by the following expression.

$$\hat{\sigma}_{t+h}^2 \to E[\varepsilon^2] = \frac{\alpha_0}{1 - \left(\sum_{i=1}^{p} \alpha_i\right)}$$

It should be remarked that this result is not annualized. For example, if the forecast is based on the daily return data, then an annualization coefficient of 250 must be multiplied to get the yearly value ($\sigma_{yearly}^2 = 250\, \sigma_{daily}^2$).

Notice also that the volatility is the square root of the above result: $\hat{\sigma} = \sqrt{\hat{\sigma}^2}$.

For more details of the ARCH(p) model, refer to the remarks on the function TSARCHFit.

Finally, the confidence intervals are estimated by Monte Carlo simulation. The larger the number of samples (*intSamples*), the more precise the interval estimates one can get. However, this comes at the expense of time. A few thousands of samples would suffice in most situations.

TSARCHSimul

Purpose

Simulates the volatilities of the Auto-Regressive Conditional Heteroskedasticity ARCH(*p*) process.

Syntax

TSARCHSimul(*rangeAlpha*, *intSeed*, *intN*)

Input Arguments

rangeAlpha	Cell range containing the alpha parameters: $\alpha_0, \alpha_1, \ldots, \alpha_p$. It must be a single column or a single row array of length $p+1$. It is required that $p \geq 1$.
intSeed	Seed for the random number generator. It should be an integer number such as *1234*.
intN	Number of the simulation steps (*N*).

Output

The output is a column array of length *N*. The time ordering is always descendent. Unlike the GARCH simulation (see TSGARCHSimul), the starting volatility cannot be specified. In fact, the first *p* steps are sampled randomly.

Remarks

ARCH(*p*) model assumes the following serial dependence. Notice that the error terms (also called "innovations") have variable σ_t^2, that is $\varepsilon_t \sim N(0, \sigma_t^2)$.

$$\sigma_t = \sqrt{\alpha_0 + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2}$$

It should be remarked that this result is not annualized. For example, if the daily time unit is assumed, an annualization coefficient of $\sqrt{250}$ must be multiplied to get the yearly value ($\sigma_{yearly} = \sqrt{250}\,\sigma_{daily}$).

The ARCH parameters must obey the following constraints.

$$\alpha_0 > 0, \quad \alpha_i \geq 0, \quad \sum_{i=1}^{p} \alpha_i < 1$$

In case any of the above constraints is violated, value 0 is returned by this function.

TSARCHVol

Purpose

Calculates the volatilities of the Auto-Regressive Conditional Heteroskedasticity ARCH(p) process.

Syntax

TSARCHVol(*rangeTS*, *boolAscend*, *rangeAlpha*)

Input Arguments

rangeTS	Cell range of a univariate time series (example: financial return). It must be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
rangeAlpha	Cell range containing the alpha parameters: $\alpha_0, \alpha_1, \ldots, \alpha_p$. It must be a single column or a single row array of length $p+1$. It is required that $p \geq 1$.

Output

The output is a column array of length equal to that of *rangeTS*. The time ordering is always descendent.

Remarks

In order to obtain the parameters (α_i) required for this function, execute first the function TSARCHFit.

ARCH(p) model assumes the following serial dependence of the volatility.

$$\sigma_t = \sqrt{\alpha_0 + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2}$$

See also TSGARCHVol for the GARCH volatility.

TSARCHVol2

Purpose

Calculates the volatilities of the Auto-Regressive Conditional Heteroskedasticity ARCH(*p*) process. Version 2.

Syntax

TSARCHVol2(*rangeTS*, *boolAscend*, *intP*)

Input Arguments

rangeTS	Cell range of a univariate time series (example: financial return). It must be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
intP	Number of the α_i parameters (*p*).

Output

The output is a column array of length equal to that of *rangeTS*. The time ordering is always descendent.

Remarks

Unlike TSARCHVol, this function does not require the α_i parameters as input. ARCH(*p*) model assumes the following serial dependence of the volatility.

$$\sigma_t = \sqrt{\alpha_0 + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2}$$

See also TSGARCHVol2 for the GARCH equivalent.

TSARIMAFit

Purpose

Calculates the parameters of the Auto-Regressive Integrated Moving Average ARIMA(*p*,*d*,*q*) model.

Syntax

TSARIMAFit(*rangeTS*, *boolAscend*, *boolLog*, *intP*, *intD*, *intQ*)

Input Arguments

rangeTS	Cell range of a time series. It must be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
boolLog	Whether to convert the input time series into another of logarithmic rate of change.
intP	Number of the Auto-Regression (AR) parameters (*p*).
intD	Order of integration (*d*) of the input time series. Conversely, it is same as the order of difference operations required to reduce the input time series to a stationary one. It should be an integer equal or larger than 0.
intQ	Number of the Moving Average (MA) parameters (*q*).

Output

The output consists of the parameter estimates as well as the corresponding errors ($error_i$). Thus, the output is a two column matrix of length (rows) equal to $intP + intQ + 1$. ϕ_0 is the constant intercept and $\phi_1, \cdots, \phi_p$ are the "proper" AR parameters. However, most of the time we will lump them together as $\phi_0, \phi_1, \cdots, \phi_p$ and refer to them simply as AR parameters. $\theta_1, \cdots, \theta_q$ are the MA parameters.

ϕ_0	$error_0$
ϕ_1	$error_1$
$\vdots$	$\vdots$
ϕ_p	$error_P$
θ_1	$error_{p+1}$
θ_2	$error_{p+2}$
$\vdots$	$\vdots$
θ_q	$error_{p+q}$

Remarks

When *intLog* is set to *TRUE*, the input time series is first converted into another one of logarithmic rate of change. This conversion operation is the same as what the function TSLogRate does. This is a useful feature especially when the input is a non-stationary price time series. The logarithmic rate of change (return) series is more likely to be a stationary time series.

When denoted as ARIMA(*p*,*d*,*q*), it means that there are *p* auto-regression parameters and q moving average parameters in addition to the constant intercept: thus, there are $p + q + 1$ parameters in total. d means the order of the difference operation. Here, the difference operation is the same as by the function TSDifference.

When $d = 0$, ARIMA is reduced to the Auto-Regressive Moving Average (ARMA). Furthermore, when $q = 0$ ARMA is reduced to the Auto-Regressive (AR), while when $p = 0$ it is reduced to the Moving Average (MA).

ARMA(*p*,*q*) model assumes the following serial dependence. The terms denoted as ε_t represent the errors (also called "innovations") of the time series. They are assumed to be independent and identically distributed (IID). Also, the errors are assumed to obey the normal distribution of mean zero and fixed variance.

$$x_t = \varepsilon_t + \phi_0 + \sum_{i=1}^{p} \phi_i x_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}$$

From this, it is straightforward to see that AR(*p*) assumes serial dependence of the following form because $q = 0$.

$$x_t = \varepsilon_t + \phi_0 + \sum_{i=1}^{p} \phi_i x_{t-i}$$

When $p = 0$, we have MA model for which instead of the constant intercept "ϕ_0" we introduce "μ" to represent the mean of the time series. Thus, MA(q) is as following.

$$x_t = \mu + \varepsilon_t + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}$$

TSARMAForecast

Purpose

Applies ARMA(*p*,*q*) to a time series and forecasts *N* time steps. Please, notice that the order of integration is assumed zero ($d = 0$).

Syntax

TSARMAForecast(*rangeTS*, *boolAscend*, *boolLog*, *intP*, *intQ*, *intN*, *intInterval*, *intSamples*)

Input Arguments

rangeTS	Cell range of a stationary time series. It must be a single column array. This is the training set.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
boolLog	Whether to convert the input time series into another of logarithmic rate of change.
intP	Number of the Auto-Regression (AR) parameters (*p*).
intQ	Number of the Moving Average (MA) parameters (*q*).
intN	Number of the forecast steps (*N*).
intInterval	Choice of the confidence interval. 0 for no interval, 1 for 68% interval, 2 for 95% interval and 3 for 99.7% interval.
intSamples	Number of samples for estimating the confidence interval by Monte Carlo method.

Output

When *intInterval = 0*, the output is a single column array of length $N + 1$. When *intInterval* is non zero, the output is a three column matrix of size $(N + 1) \times 3$. The columns correspond to the lower limit, middle point and the upper limit of the forecast. The step "0" of the forecast is taken from the most recent value of the input time series. The output time

ordering is always descendent.

Remarks

When *intLog* is set to *TRUE*, the input time series is first converted into another one of logarithmic rate of change. This conversion operation is the same as what the function TSLogRate does. This is a useful feature especially when the input is a non-stationary price time series. The logarithmic rate of change (return) series is more likely to be a stationary time series.

One important aspect of this function is that when *intLog* is set to *TRUE*, the input series is converted to its logarithmic rate of change for ARMA forecast (requires stationarity) and then then the result is converted back (by exponential function). All this conversion and reversal happen internally and the user sees only the forecast of the original input time series.

This makes sense in situations where the user wants to forecast non-stationary price series while internally ARMA is applied to the returns.

If we have the time series data up to the step t (that is up to x_t) and then would like to forecast the step $t+1$ (we denote the predicted value as $\hat{x}_{t+1}$), we apply the following expression. Notice that ϕ_i are the AR model parameters and θ_i are the MA model parameters.

$$\hat{x}_{t+1} = \phi_0 + \phi_1 x_t + \phi_2 x_{t-1} + \cdots + \phi_p x_{t-p+1} +$$

$$\theta_1 \varepsilon_t + \theta_2 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q+1}$$

In order to predict the next step $t+2$, we would feed back $\hat{x}_{t+1}$ and apply the following. Notice that we are assuming that the expected value of ε_{t+1} is zero ($E[\varepsilon_{t+1}] = 0$), thus the θ_1 term is no longer present.

$$\hat{x}_{t+2} = \phi_0 + \phi_1 \hat{x}_{t+1} + \phi_2 x_t + \cdots + \phi_p x_{t-p+2} +$$

$$\theta_2 \varepsilon_t + \cdots + \theta_q \varepsilon_{t-q+2}$$

Then, we can proceed analogously to advance any number of steps into the future. After q steps, the contributions from the MA terms are gone and only the AR terms remain.

For a large number of steps $h \to \infty$, the prediction converges to the unconditional mean of x_t given by the following expression.

$$\hat{x}_{t+h} \to \mu = \frac{\phi_0}{1 - \sum_{i=1}^{p} \phi_i}$$

When $q = 0$ ARMA is reduced to the Auto-Regressive (AR), while when $p = 0$ it

is reduced to the Moving Average (MA).

For more details of the ARMA(*p*,*q*) model, refer to the remarks on the function TSARIMAFit. Here, the order of integration is assumed 0. If difference operation is needed, refer to the function TSDifference.

Finally, the confidence intervals are estimated by Monte Carlo simulation. The larger the number of samples (*intSamples*), the more precise the interval estimates one can get. However, this comes at the expense of time. A few thousands of samples would suffice in most situations.

TSARMASimul

Purpose

Simulates *N* steps of the stationary Auto-Regressive Moving Average ARMA(*p*,*q*) process.

Syntax

TSARMASimul(*rangeAR, rangeMA, realSigma, intSeed, intN*)

Input Arguments

rangeAR	Cell range containing the AR parameters: $\phi_0, \phi_1, \ldots, \phi_p$. It must be a single column or a single row array of length $p+1$. Notice that even when the $p = 0$ (strictly MA process), still the value of ϕ_0 must be specified as a minimum requirement. In this case, ϕ_0 becomes the unconditional mean μ and you may type in the value directly instead of referencing a cell.
rangeMA	Cell range containing the MA parameters: $\theta_1, \cdots, \theta_q$. It must be a single column or a single row array of length q. In case the specified range is just one cell, you can also directly type in the value instead of referencing the cell. When $q = 0$ (there is no MA parameter), it becomes a strictly AR process. In this case, it is suggested to use the function TSARSimul instead.
realSigma	Standard deviation of the Gaussian innovation (σ_ε).
intSeed	Seed for the random number generator. It should be an integer number such as *1234*.
intN	Number of the simulation steps (*N*).

Output

The output is a column array of length *N*. The time ordering is always descendent.

Remarks

The first step is always the unconditional mean of the time series denoted as μ. For an AR or ARMA model, the unconditional mean can be calculated using the parameters $\phi_0, \phi_1, \ldots, \phi_p$:

$$\mu = \frac{\phi_0}{1 - \sum_{i=1}^{p} \phi_i}$$

For a purely MA model (except for ϕ_0, all the other ϕ_i are zero), the unconditional mean of the time series is $\mu = \phi_0$.

The innovations (ε_t) are normally distributed (that is, $\varepsilon_t \sim N(0, \sigma_\varepsilon^2)$).

For more details of the ARMA(*p*,*q*) model, refer to the remarks on the function TSARIMAFit.

TSARSimul

Purpose

Simulates *N* steps of the stationary Auto-Regressive AR(*p*) process.

Syntax

TSARSimul(*rangeAR*, *realSigma*, *intSeed*, *intN*)

Input Arguments

rangeAR	Cell range containing the AR parameters: $\phi_0, \phi_1, \ldots, \phi_p$. It must be a single column or a single row array of length $p+1$. In case the specified range is just one cell, you may type in the corresponding value instead. For the full ARMA process, use TSARMASimul instead.
realSigma	Standard deviation of the Gaussian innovation (σ_ε).
intSeed	Seed for the random number generator. It should be an integer number such as *1234*, *4321*, … etc.
intN	Number of the simulation steps (*N*).

Output

The output is a column array of length *N*. The time ordering is always descendent.

Remarks

The first step is always the unconditional mean of the time series calculated as $\mu = \frac{\phi_0}{1-\sum_{i=1}^{p}\phi_i}$. The innovations ($\varepsilon_t$) are normally distributed (that is, $\varepsilon_t \sim N(0, \sigma_\varepsilon^2)$). For more details of the ARMA(*p*,*q*) model, refer to the remarks on the function TSARIMAFit.

TSARStat

Purpose

Calculates various diagnostic measures such as PACF, AIC, BIC by applying Auto-Regressive AR(*p*) model.

Syntax

TSARStat(*rangeTS*, *intPmax*, *boolAscend*)

Input Arguments

rangeTS	Cell range of a stationary time series. It must be a single column array.
intPmax	Maximum number of the Auto-Regression parameters (*p*).
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.

Output

The output is a matrix of dimension $intPmax \times 3$. Each element is as explained below. *PACF = Partial Auto Correlation Function*, *AIC = Akaike Information Content*, and *BIC = Bayes Information Content*. *BIC* is identical to *SIC* (Schwarz Information Content).

$$\begin{matrix} PACF_1 & AIC_1 & BIC_1 \\ PACF_2 & AIC_2 & BIC_2 \\ \vdots & \vdots & \vdots \\ PACF_{Pmax} & AIC_{Pmax} & BIC_{Pmax} \end{matrix}$$

Remarks

The lag-1 sample PACF is the estimate ϕ_1 of the AR(*1*) model, The lag-2 sample

PACF is the estimate ϕ_2 of the AR(2) model and so on and so forth. Thus, the lag-n sample PAFC is the estimate ϕ_n of the AR(*n*) model.

AIC is calculated using the following expression

$$AIC = -2\frac{Log\ likelihood}{T} + 2\frac{p}{T}$$

where $Log\ likelihood = -\frac{T}{2}\left(1 + Ln(2\pi) + Ln\left(\frac{SSE}{T}\right)\right)$ and $SSE = \sum_{i=1}^{T}(e_i)^2 = \sum_{i=1}^{T}(y_i - \hat{y}_i)^2$. *T* denotes the total length of the time series.

BIC is calculated using the following expression.

$$BIC = -2\frac{Log\ likelihood}{T} + p\frac{Ln(T)}{T}$$

TSConvARMAtoMA

Purpose

Converts the Auto-Regressive Moving Average (ARMA) model parameters to the Moving Average (MA) model parameters.

Syntax

TSConvARMAtoMA(*rangeAR*, *rangeMA*)

Input Arguments

rangeAR	Cell range containing the AR parameters: $\phi_0, \phi_1, \ldots, \phi_p$. It must be a single column or a single row array of length $p+1$.
rangeMA	Cell range containing the MA parameters: $\theta'_1, \theta'_2, \ldots, \theta'_q$. It must be a single column or a single row array of length q. Here, we used the primed notation in order to distinguish these MA parameters from the output.

Output

The output is a column array containing the MA parameters ordered in the following way: $\mu, \theta_1, \theta_2, \theta_3, \ldots$ where μ is the unconditional mean of the time series. When converting from the ARMA parameters, in principle there will be an infinite number of the MA parameters. However, this function imposes an artificial cutoff at the 30-eth component. When the input parameters do not satisfy the stationarity condition, an array containing zeroes is returned.

Remarks

The ARMA(p,q) model can be expressed as following.

$$x_t = \varepsilon_t + \phi_0 + \sum_{i=1}^{p} \phi_i x_{t-i} + \sum_{i=1}^{q} \theta'_i \varepsilon_{t-i}$$

Now, let us define operator $\Phi(L) = 1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_p L^p$ using the lag operator L. The lag operator is such that $L^i x_t = x_{t-i}$. In the same fashion, let us also define operator $\Theta(L) = 1 + \theta'_1 L + \theta'_2 L^2 + \cdots + \theta'_q L^q$. Then, the ARMA($p$,$q$) model can be rewritten as

$$\Phi(L)(x_t - \mu) = \Theta(L)\varepsilon_t$$

Here ϕ_0 and μ are connected: $\phi_0 = \mu(1 - \phi_1 - \phi_2 - \cdots - \phi_p)$. The conversion consists in applying the inversion operator $\Phi(L)^{-1}$.

$$x_t = \mu + \Phi(L)^{-1}\Theta(L)\varepsilon_t$$

$$\Rightarrow \quad x_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots$$

This conversion (inversion) is feasible if the input parameters satisfy the stationarity condition. See TSARCharRoots about the characteristic roots and the stationarity condition. Please notice that the stationarity condition depends only on the AR parameters: $\phi_0, \phi_1, \ldots, \phi_p$.

See also TSConvARtoMA and TSConvMAtoAR.

TSConvARtoMA

Purpose

Converts the Auto-Regressive (AR) model parameters to the Moving Average (MA) model parameters.

Syntax

TSConvARtoMA(*rangeAR*)

Input Arguments

rangeAR	Cell range containing the AR parameters: $\phi_0, \phi_1, \ldots, \phi_p$. It must be a single column or a single row array of length $p+1$.

Output

The output is a column array containing the MA parameters ordered in the following way: $\mu, \theta_1, \theta_2, \theta_3, \ldots$ where μ is the unconditional mean of the time series. When converting from the AR parameters, in principle there will be an infinite number of the MA parameters. However, this function imposes an artificial cutoff at the 30-eth component. When the input parameters do not satisfy the stationarity condition, an array containing zeroes is returned.

Remarks

The AR(p) model can be expressed as following.

$$x_t = \varepsilon_t + \phi_0 + \sum_{i=1}^{p} \phi_i x_{t-i}$$

Now, let us define operator $\Phi(L) = 1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_p L^p$ using the lag operator L. The lag operator is such that $L^i x_t = x_{t-i}$. Then, the AR(p) model can be rewritten as

$$\Phi(L)(x_t - \mu) = \varepsilon_t$$

Here ϕ_0 and μ are connected: $\phi_0 = \mu(1 - \phi_1 - \phi_2 - \cdots - \phi_p)$. The conversion consists in applying the inversion operator $\Phi(L)^{-1}$.

$$x_t = \mu + \Phi(L)^{-1}\varepsilon_t$$

$$\Rightarrow \quad x_t = \mu + \varepsilon_t + \theta_1\varepsilon_{t-1} + \theta_2\varepsilon_{t-2} + \cdots$$

This conversion (inversion) is feasible if the input parameters satisfy the stationarity condition. See TSARCharRoots about the characteristic roots and the stationarity condition. See also TSIsStationaryAR, TSConvARMAtoMA and TSConvMAtoAR.

TSConvMAtoAR

Purpose

Converts the Moving Average (MA) model parameters to the Auto-Regressive (AR) model parameters.

Syntax

TSConvMAtoAR(*rangeMA*)

Input Arguments

rangeMA	Cell range containing the MA parameters: $\mu, \theta_1, \theta_2, \ldots, \theta_q$. Thus, it must be a single column or a single row array of length $q + 1$.

Output

The output is a column array containing the AR parameters ordered in the following way: $\phi_0, \phi_1, \phi_2, \ldots$ where ϕ_0 is the constant intercept. When converting from the MA parameters, in principle there will be an infinite number of the AR parameters. However, this function imposes an artificial cutoff at the 30-eth component. When the input parameters do not satisfy the inversion condition, an array containing zeroes is returned.

Remarks

The MA(q) model can be expressed as following.

$$x_t = \varepsilon_t + \mu + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}$$

Now, let us define operator $\Theta(L) = 1 + \theta_1 L + \theta_2 L^2 + \cdots + \theta_q L^q$ using the lag operator L. The lag operator is such that $L^i \varepsilon_t = \varepsilon_{t-i}$. Then, the MA($q$) model can be rewritten as

$$x_t - \mu = \Theta(L)\varepsilon_t$$

The conversion consists in applying the inversion operator $\Theta(L)^{-1}$ to both sides of the equality.

$$\Theta(L)^{-1}(x_t - \mu) = \varepsilon_t$$

$$\Rightarrow \quad x_t = \varepsilon_t + \phi_0 + \phi_1 x_{t-1} + \phi_2 x_{t-2} + \cdots$$

Here, ϕ_0 and μ are connected: $\phi_0 = \mu\left(1 - \phi_1 - \phi_2 - \cdots - \phi_p\right)$.

See also TSIsInvertibleMA, TSConvARtoMA and TSConvARMAtoMA.

TSDickeyFuller

Purpose

Performs the Dickey-Fuller unit root test as well as the Engle-Granger cointegration test.

Syntax

TSDickeyFuller(*rangeTS*, *boolAscend* , *intN*, *intCritical*)

Input Arguments

rangeTS	Cell range of a univariate time series. It must be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
intN	When $intN = 1$, it is Dickey-Fuller unit root test. When $intN > 1$, the test is on the residual of the linear combination of more than one univariate time series. In this case, *intN* must be equal to the number of the univariate time series of which the portfolio is composed. When $intN > 6$, the criticality of $intN = 6$ is assumed.
intCritical	Sets the critical value. It can take values 0, 1 or 2 which correspond to 90%, 95% and 99% criticality levels.

Output

The output is formatted as following (row array of 5 components). For the meanings of ϕ_1, *err*, *test statistic* and *critical value*, see the remarks section below.

ϕ_1	*err*	*test statistic*	*critical value*	*hypothesis*

The output component "*hypothesis*" can be either 0 or 1.

If *hypothesis* is 0, it means that the criticality is not reached and the null hypothesis (H_0) is kept. When $intN = 1$, the H_0 is that the unit root is present. When $intN > 1$, the H_0 is that the linear combination of the univariate time series is not cointegrating.

If *hypothesis* is 1, the test statistic is larger in the negative direction than the critical value. In this case, the null hypothesis is rejected in favor of the alternative hypothesis (H_1). If $intN = 1$, the H_1 is that the time series is likely stationary. Accordingly, if $intN > 1$, the H_1 is that the linear combination of the univariate time series is likely cointegrating. This last case forms part of the Engle-Granger test for cointegration. See also TSEngleGranger for more details.

Remarks

To the given time series X_t the auto-regressive model of the following form can be applied.

$$X_t = \phi_0 + \phi_1 X_{t-1} + \varepsilon_t$$

Here ϕ_0 and ϕ_1 are the parameters of the AR(1) model. In particular, when the value of ϕ_1 is equal to 1 it is said that the time series has a unit root. This means that the time series is non-stationary and non-reverting.

For the null hypothesis (H_0) that assumes the existence of the unit root, the test statistic is calculated as following. Notice that *err* is the error of the parameter ϕ_1.

$$test\ statistic = \frac{\phi_1 - 1}{err}$$

The value of the test statistic is usually negative. When the value of the test statistic crosses the given criticality level (larger in the negative direction than the critical value), the null hypothesis can be rejected.

The critical values depend on the number of the test samples as well as on whether it is a simple unit root test or the cointegration test. The critical values do not follow any simple distribution function and must be obtained by a simulation method. According to J. G. MacKinnon[*], the critical values (denoted as β) can be parameterized in the following way.

$$\beta(T) = \beta_\infty + \frac{\beta_1}{T} + \frac{\beta_2}{T^2}$$

Here, T is the length of the time series (number of samples) and $\beta_\infty, \beta_1, \beta_2$ are tabulated parameters(refer to [*]).

See also TSDickeyFullerAugmented.

[*] *"Critical Values for Cointegration Tests"*, James G. MacKinnon in R. F. Engle and C. W. J. Granger (editors) *"Long-Run Economic Relationships"*, Oxford University Press, p267-276.

TSDickeyFullerAugmented

Purpose

Performs the augmented Dickey-Fuller unit root test.

Syntax

TSDickeyFullerAugmented(*rangeTS*, *boolAscend* , *intMaxLag*, *intCritical*)

Input Arguments

rangeTS	Cell range of a univariate time series. It must be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
intMaxLag	It must be at least 0 or larger. Maximum lag included in the model. Here, we denote by M = *intMaxLag*.
intCritical	Sets the critical value. It can take values 0, 1 or 2 which correspond to 90%, 95% and 99% criticality levels.

Output

The output is formatted as following (row array of 5 components). For the meanings of π, *err*, *test statistic* and *critical value*, see the remarks section below.

π	*err*	*test statistic*	*critical value*	*hypothesis*

The output component "*hypothesis*" can be either 0 or 1.

If *hypothesis* is 0, it means that the criticality is not reached and the null hypothesis (H_0) is kept. Here, the H_0 is that the unit root is present. If *hypothesis* is 1, the test statistic is

larger in the negative direction than the critical value. In this case, the null hypothesis is rejected in favor of the alternative hypothesis (H_1). The H_1 is that the time series is likely stationary.

Remarks

To the given time series X_t the following model is applied. Notice that ΔX_t means the difference: $\Delta X_t = X_t - X_{t-1}$. M denotes the maximum lag (*intMaxLag*).

$$\Delta X_t = \alpha + \pi X_{t-1} + \left(\sum_{i=1}^{M} \phi_i^* \, \Delta X_{t-i} \right) + \varepsilon_t$$

Here, α, π and ϕ_i^* are the parameters of the model. In particular, when the value of π is equal to 0 it is said the time series has a unit root. This means that the time series is non-stationary and non-reverting.

The parameters α, π and ϕ_i^* can be related to those of the auto-regressive model. In fact, if we consider AR(p) model with $p = M + 1$ and the parameters $\phi_0, \phi_1, \cdots, \phi_p$ (AR(p) model assumes $X_t = \varepsilon_t + \phi_0 + \sum_{i=1}^{p} \phi_i X_{t-i}$), we have the following relations.

$$\alpha = \phi_0$$

$$\pi = \left(\sum_{i=1}^{p} \phi_i \right) - 1$$

$$\phi_j^* = - \sum_{i=j+1}^{p} \phi_i$$

For the null hypothesis (H_0) that assumes the unit root, the test statistic is calculated as following. Here, *err* is the error of the parameter π.

$$test\ statistic = \frac{\pi}{err}$$

The value of the test statistic is usually negative. When the value of the test statistic crosses the given criticality level (larger in the negative direction than the critical value), the null hypothesis is rejected. For the calculation of the critical values refer to the remarks on the function TSDickeyFuller.

TSDifference

Purpose

Applies difference operator to a time series.

Syntax

TSDifference(*rangeTS*, *boolAscend*, *intD*)

Input Arguments

rangeTS	Cell range of a univariate time series. It must be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
intD	Order or power of the difference operator. It must be an integer equal or larger than 0. When *intD=0*, this function returns the original time series.

Output

The output is another time series shown as a column array in descending order. It shows the result after differencing *d* times the input time series. Here, $d \equiv intD$. The length of the output is $N - d$ where *N = length of the input time series*. Thus, the input time series must have length *N* larger than *d*.

Remarks

Let us define difference operator $D \equiv 1 - L$ using the lag operator L. The lag operator is such that $L^i x_t = x_{t-i}$. The output time series y_t and the input time series x_t are related as following.

$$y_t = D^d x_t = (1 - L)^d x_t$$

The difference operations can be explicitly written as following.

d=1	$y_t = x_t - x_{t-1}$
d=2	$y_t = x_t - 2x_{t-1} + x_{t-2}$
d=3	$y_t = x_t - 3x_{t-1} + 3x_{t-2} - x_{t-3}$
d=4	$y_t = x_t - 4x_{t-1} + 6x_{t-2} - 4x_{t-3} + x_{t-4}$
d=5	$y_t = x_t - 5x_{t-1} + 10x_{t-2} - 10x_{t-3} + 5x_{t-4} - x_{t-5}$
⋮	⋮
⋮	⋮

TSEngleGranger

Purpose

Performs the Engle-Granger cointegration test.

Syntax

TSEngleGranger(*rangeTS*, *boolAscend*, *boolLog*, *boolSort*)

Input Arguments

rangeTS	Cell range of a multivariate time series (Example: time evolution of multiple asset prices). The range must include at least two columns or more.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
boolLog	If *TRUE*, applies natural logarithm to the values of the time series. In this case, it is required that the times series be of positive non-zero values.
boolSort	If *TRUE*, presents the result sorted according to the Dickey-Fuller statistic of the residual (spread). Larger negative Dickey-Fuller statistic receives priority (from top to bottom rows) as it would mean departure from the unit root (thus, stronger tendency of mean reversion).

Output

First, let us assume that N univariate time series (number of columns in the input range) have been selected. Then the dimension of the output is $N \times (N + 5)$. In fact, the output is composed of two parts. Left portion of dimension $N \times (N + 1)$ shows the

"weights" (or hedge ratios) of each asset, while the right portion of dimension $N \times 4$ shows the results of the Dickey-Fuller test on the residuals (also called "spreads").

Different rows mean different linear combinations. For each linear combination, one of the time series is given a weight equal to 1 while others are given weights such that the residual (spread) is minimized.

Dickey-Fuller test consists in calculating the Dickey-Fuller test statistic and then comparing it with the critical values. When the critical value is not reached (in the negative direction) the null hypothesis is kept. Whereas surpassing of the given criticality means that the alternative hypothesis is chosen. Null hypothesis denoted as H_0 means that the residual series of a given linear combination has unit root (not cointegrating). Alternative hypothesis denoted as H_1 means that the residual series of a given linear combination is mean reverting (cointegrating). The criticality can be set at 90%, 95% and 99% being the higher the more stringent condition of the cointegration.

In summary, the actual output looks like the following (excluding the labels). The left and the right portions are denoted in red and blue colors respectively (not the actual colors, shown for distinction).

	Weight#0	⋯	*Weight #N*	*DF test statistic*	*90%*	*95%*	*99%*
Combo #1							
Combo #2							
⋮							
Combo #M							

Here, the output components under 90%, 95% and 99% labels can have values 0 or 1 that indicate whether null hypothesis (H_0) or the alternative hypothesis (H_1) are chosen.

Remarks

The value of each univariate time series at a given instant *t* can be denoted as $Y_{i,t}$. Here, we assume that there are *N* univariate time series. Thus, the sub-index i runs as $i = 1, \cdots, N$. According to Engle and Granger, the cointegration is tested in two steps.

1. Find the linear combination of the time series such that the resulting ε_t (called residual or spread) is minimized. The weights denoted as w_i are the "hedge ratios". A positive hedge ratio would mean long position while a negative hedge ratio would mean short position.

$$w_0 + w_1 Y_{1,t} + w_2 Y_{2,t} + \cdots + w_N Y_{N,t} = \varepsilon_t$$

In case, *boolLog* is set to *TRUE*, the following linear combination is sought.

$$w_0 + w_1 Ln(Y_{1,t}) + w_2 Ln(Y_{2,t}) + \cdots + w_N Ln(Y_{N,t}) = \varepsilon_t$$

2. Test whether the residual series ε_t is stationary. This is achieved by subjecting the residual series to the Dickey-Fuller test.

See also TSDickeyFuller, TSDickeyFullerAugmented, TSEngleGrangerSpread, TSEngleGrangerSpreadForecast.

TSEngleGrangerSpread

Purpose

Applies the Engle-Granger cointegration test and calculates the spread of the cointegrating portfolio.

Syntax

TSEngleGrangerSpread(*rangeTS*, *boolAscend* , *boolLog*, *boolSort*)

Input Arguments

rangeTS	Cell range of a multivariate time series (Example: time evolution of multiple asset prices). The range must include at least two columns or more.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
boolLog	If *TRUE*, applies natural logarithm to the values of the time series. In this case, it is required that the times series be of positive non-zero values.
boolSort	If *TRUE*, presents the result sorted according to the Dickey-Fuller statistic of the spread. Larger negative Dickey-Fuller statistic receives priority (from left to right columns).

Output

First, let us assume that N univariate time series (number of columns in the input range) of length T (number of rows in the input range) have been selected. Then the dimension of the output is $T \times N$. Results from different linear combinations are ordered in columns from left to right. These linear combinations have weights given by TSEngleGranger (here, ordered from top to bottom).

When *boolSort* is set to *TRUE*, the leftmost column corresponds to the linear

combination with the strongest tendency to mean revert. The criterion for this selection is the Dickey-Fuller statistic of the spread.

Remarks

The value of each univariate time series at a given instant *t* can be denoted as $Y_{i,t}$. Here, we assume that there are *N* univariate time series. Thus, the sub-index i runs as $i = 1, \cdots, N$. According to Engle and Granger, the cointegration is tested in two steps.

1. Find the linear combination of the time series such that the resulting ε_t (called residual or spread) is minimized. The weights denoted as w_i are the "hedge ratios". A positive hedge ratio would mean long position while a negative hedge ratio would mean short position.

$$w_0 + w_1 Y_{1,t} + w_2 Y_{2,t} + \cdots + w_N Y_{N,t} = \varepsilon_t$$

   In case, *boolLog* is set to *TRUE*, the following linear combination is sought.

$$w_0 + w_1 Ln(Y_{1,t}) + w_2 Ln(Y_{2,t}) + \cdots + w_N Ln(Y_{N,t}) = \varepsilon_t$$

2. Test whether the residual series ε_t is stationary. This is achieved by subjecting the residual series to the Dickey-Fuller test.

This function calculates explicitly the residual series ε_t for the purpose of plotting or further testing.

See also TSDickeyFuller, TSDickeyFullerAugmented, TSEngleGranger, TSEngleGrangerSpreadForecast.

TSEngleGrangerSpreadForecast

Purpose

Forecasts the multivariate time series by VAR (Vector Auto-Regressive) model and then calculates the spread by Engle-Granger method.

Syntax

TSEngleGrangerSpreadForecast(*rangeTS, boolAscend, boolLog, boolSort, intP, intN*)

Input Arguments

rangeTS	Cell range of a multivariate time series (Example: time evolution of multiple asset prices). The range must include at least two columns or more.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
boolLog	If *TRUE*, applies natural logarithm to the values of the time series. In this case, it is required that the times series be of positive non-zero values.
boolSort	If *TRUE*, presents the result sorted according to the Dickey-Fuller statistic of the spread. Larger negative Dickey-Fuller statistics receive priority (from left to right columns).
intP	Number of the parameter matrices (p) of the VAR model.
intN	Number of the forecast steps (N).

Output

First, let us assume that M univariate time series (number of columns in the input range) of length T (number of rows in the input range) have been selected. Then the dimension of the output is $(N+1) \times M$. The "step 0" of the forecast is from the most recent value of the input time series. Each column corresponds to a different linear

combination.

When *boolSort* is set to *TRUE*, the leftmost column corresponds to the linear combination with the strongest tendency to mean revert. The criterion for this selection is the Dickey-Fuller statistic of the spread.

While the function TSEngleGrangerSpread calculates the spreads of the past, here the future spreads are predicted.

Remarks

About the forecast of the multivariate time series see TSVARFit.

About the Engle-Granger method for the cointegration test refer to TSDickeyFuller, TSDickeyFullerAugmented, TSEngleGranger, TSEngleGrangerSpread.

TSGARCHFit

Purpose

Calculates the parameters of Generalized Auto-Regressive Conditional Heteroskedasticity GARCH(*p*,*q*) model.

Syntax

TSGARCHFit(*rangeTS*, *boolAscend*, *intP*, *intQ*)

Input Arguments

rangeTS	Cell range of a univariate time series (example: financial return). It must be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
intP	Number of the α_i parameters (*p*). It must be equal or larger than 1.
intQ	Number of the β_i parameters (*q*). Here, *intQ* must be equal or larger than 1. When *intQ* is 0, the model to use would be ARCH. See TSARCHFit instead.

Output

The output consists of the parameter estimates as well as the corresponding errors ($error_i$). Thus, the output is a two column matrix of length (rows) equal to $p+q+1$. $\alpha_0, \alpha_1, \cdots, \alpha_p$ (so-called ARCH parameters) along with $\beta_1, \cdots, \beta_q$ constitute the GARCH parameter set.

α_0	$error_0$
α_1	$error_1$
⋮	⋮

$$\begin{matrix} \alpha_p & error_p \\ \beta_1 & error_{p+1} \\ \beta_2 & error_{p+2} \\ \vdots & \vdots \\ \beta_q & error_{p+q} \end{matrix}$$

Remarks

This is the so-called symmetric normal GARCH model. Hence, the errors (ε_t) are assumed to obey the normal distribution of mean zero and time varying conditional variance: $\varepsilon_t \sim N(0, \sigma_t^2)$.

GARCH(p,q) model assumes the following serial dependence of the volatility.

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2$$

The GARCH parameters must obey the following constraint.

$$\alpha_0 > 0, \quad \alpha_i \geq 0, \quad \beta_i \geq 0, \quad \sum_{i=1}^{p} \alpha_i + \sum_{j=1}^{q} \beta_j < 1$$

The unconditional mean of ε_t^2 (long run prediction of σ_t^2) is given by the following expression.

$$E[\varepsilon^2] = \frac{\alpha_0}{1 - \left(\sum_{i=1}^{p} \alpha_i + \sum_{j=1}^{q} \beta_j\right)}$$

For the ARCH(p) model refer to TSARCHFit.

TSGARCHForecast

Purpose

Applies GARCH(*p*,*q*) model to a time series and forecasts volatility.

Syntax

TSGARCHForecast(*rangeTS*, *boolAscend*, *intP*, *intQ*, *intN*, *intInterval*, *intSamples*)

Input Arguments

rangeTS	Cell range of a univariate time series (example: financial return). It must be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
intP	Number of the α_i parameters (*p*).
intQ	Number of the β_i parameters (*q*).
intN	Number of the forecast steps (*N*).
intInterval	Choice of the confidence interval. 0 for no interval, 1 for 68% interval, 2 for 95% interval and 3 for 99.7% interval.
intSamples	Number of samples for estimating the confidence interval by Monte Carlo method.

Output

When *intInterval = 0*, the output is a single column array of length $N + 1$. When *intInterval* is non zero, the output is a three column matrix of size $(N + 1) \times 3$. The columns correspond to the lower limit, middle point and the upper limit of the forecast. The step "0" of the output is equal to the most recent volatility of the input data. The output time ordering is always descendent.

Remarks

Suppose we have the time series data (X_t) up to the step t with the corresponding ε_t^2 and σ_t^2. If we would like to forecast the step $t+1$ (we denote the predicted value as $\hat{\sigma}_{t+1}^2$), we apply the following expression.

$$\hat{\sigma}_{t+1}^2 = \alpha_0 + \alpha_1\,\varepsilon_t^2 + \alpha_2\,\varepsilon_{t-1}^2 + \alpha_3\,\varepsilon_{t-2}^2 + \cdots + \alpha_p\,\varepsilon_{t-p+1}^2 +$$

$$\beta_1\sigma_t^2 + \beta_2\sigma_{t-1}^2 + \beta_3\sigma_{t-2}^2 + \cdots + \beta_q\sigma_{t-q+1}^2$$

In order to predict the next step $t+2$, we would feed back $\hat{\sigma}_{t+1}^2$. Also, notice that the expected values $\hat{\varepsilon}_{t+1} = E[\varepsilon_{t+1}] = 0$ and $\hat{\varepsilon}_{t+1}^2 = E[\varepsilon_{t+1}^2] = \hat{\sigma}_{t+1}^2$. Thus, we have the following.

$$\hat{\sigma}_{t+2}^2 = \alpha_0 + \alpha_1\hat{\sigma}_{t+1}^2 + \alpha_2\,\varepsilon_t^2 + \alpha_3\,\varepsilon_{t-1}^2 + \cdots + \alpha_p\,\varepsilon_{t-p+2}^2 +$$

$$\beta_1\hat{\sigma}_{t+1}^2 + \beta_2\sigma_t^2 + \beta_3\sigma_{t-1}^2 + \cdots + \beta_q\sigma_{t-q+2}^2$$

Analogously, for the step $t+3$, we have,

$$\hat{\sigma}_{t+3}^2 = \alpha_0 + \alpha_1\hat{\sigma}_{t+2}^2 + \alpha_2\,\hat{\sigma}_{t+1}^2 + \alpha_3\,\varepsilon_t^2 + \cdots + \alpha_p\,\varepsilon_{t-p+3}^2 +$$

$$\beta_1\hat{\sigma}_{t+2}^2 + \beta_2\hat{\sigma}_{t+1}^2 + \beta_3\sigma_t^2 + \cdots + \beta_q\sigma_{t-q+3}^2$$

And so on. For a large number of steps $h \to \infty$, the prediction converges to the unconditional mean of ε_t^2 given by the following expression.

$$\hat{\sigma}_{t+h}^2 \to E[\varepsilon^2] = \frac{\alpha_0}{1 - \left(\sum_{i=1}^{p}\alpha_i + \sum_{j=1}^{q}\beta_j\right)}$$

It should be remarked that this result is not annualized. For example, if the forecast is based on the daily return data, then an annualization coefficient of 250 must be multiplied to get the yearly value ($\sigma_{yearly}^2 = 250\,\sigma_{daily}^2$).

Please, notice also that the volatility is the square root of the above result: $\hat{\sigma} = \sqrt{\hat{\sigma}^2}$.

For more details of the GARCH(*p*,*q*) model, refer to the remarks on the function TSGARCHFit.

Finally, the confidence intervals are estimated by Monte Carlo simulation. The larger the number of samples (*intSamples*), the more precise the interval estimates one can get. However, this comes at the expense of time. A few thousands of samples would suffice in most situations.

TSGARCHSimul

Purpose

Simulates the volatilities of the Generalized Auto-Regressive Conditional Heteroskedasticity GARCH(*p*,*q*) process.

Syntax

TSGARCHSimul(*rangeAlpha*, *rangeBeta*, *realVol*, *intSeed*, *intN*)

Input Arguments

rangeAlpha	Cell range containing the alpha parameters: $\alpha_0, \alpha_1, \ldots, \alpha_p$. It must be a single column or a single row array of length $p+1$. It is required that $p \geq 1$.
rangeBeta	Cell range containing the beta parameters: $\beta_1, \cdots, \beta_q$. It must be a single column or a single row array of length $q \geq 1$. In case the specified range is just one cell, you can also directly type in the value instead.
realVol	Initial volatility (σ_1).
intSeed	Seed for the random number generator. It should be an integer number such as *1234*.
intN	Number of the simulation steps (*N*).

Output

The output is a column array of length *N*. The time ordering is always descendent. The very first step (σ_1) is the same as *realVol*.

Remarks

GARCH(*p*,*q*) model assumes the following serial dependence. Notice that the error

terms (also called "innovations") have variable σ_t^2, that is $\varepsilon_t \sim N(0, \sigma_t^2)$.

$$\sigma_t = \sqrt{\alpha_0 + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2 + \sum_{i=1}^{q} \beta_i \sigma_{t-i}^2}$$

It should be remarked that this result is not annualized. For example, if daily time unit is assumed, then an annualization coefficient of $\sqrt{250}$ must be multiplied to get the yearly value ($\sigma_{yearly} = \sqrt{250}\, \sigma_{daily}$).

The GARCH parameters must obey the following constraint.

$$\alpha_0 > 0, \quad \alpha_i \geq 0, \quad \beta_i \geq 0, \quad \sum_{i=1}^{p} \alpha_i + \sum_{j=1}^{q} \beta_j < 1$$

In case any of the above constraints is violated, value 0 is returned by this function.

TSGARCHVol

Purpose

Calculates the volatilities of the Generalized Auto-Regressive Conditional Heteroskedasticity GARCH(*p*,*q*) process.

Syntax

TSGARCHVol(*rangeTS*, *boolAscend*, *rangeAlpha*, *rangeBeta*)

Input Arguments

rangeTS	Cell range of a univariate time series (example: financial return). It must be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
rangeAlpha	Cell range containing the alpha parameters: $\alpha_0, \alpha_1, \ldots, \alpha_p$. It must be a single column or a single row array of length $p+1$. It is required that $p \geq 1$.
rangeBeta	Cell range containing the beta parameters: $\beta_1, \cdots, \beta_q$. It must be a single column or a single row array of length $q \geq 1$. In case the specified range is just one cell, you can also directly type in the value instead.

Output

The output is a column array of length equal to that of *rangeTS*. The time ordering is always descendent.

Remarks

In order to obtain the parameters (α_i and β_i) required for this function, execute first the function TSGARCHFit.

GARCH(*p*,*q*) model assumes the following serial dependence of the volatility.

$$\sigma_t = \sqrt{\alpha_0 + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2 + \sum_{i=1}^{q} \beta_i \sigma_{t-i}^2}$$

See also TSARCHVol for the ARCH volatility.

TSGARCHVol2

Purpose

Calculates the volatilities of the Generalized Auto-Regressive Conditional Heteroskedasticity GARCH(*p*,*q*) process. Version 2.

Syntax

TSGARCHVol2(*rangeTS, boolAscend, intP, intQ*)

Input Arguments

rangeTS	Cell range of a univariate time series (example: financial return). It must be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
intP	Number of the α_i parameters (*p*).
intQ	Number of the β_i parameters (*q*).

Output

The output is a column array of length equal to that of *rangeTS*. The time ordering is always descendent.

Remarks

Unlike TSGARCHVol this function does not require the α_i and β_i parameters as input. GARCH(*p*,*q*) model assumes the following serial dependence of the volatility.

$$\sigma_t = \sqrt{\alpha_0 + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2 + \sum_{i=1}^{q} \beta_i \sigma_{t-i}^2}$$

See also TSARCHVol2 for the ARCH equivalent.

TSHoltFit

Purpose

Calculates the parameters of the Holt's linear exponential smoothing method.

Syntax

TSHoltFit(*rangeTS*, *boolAscend*)

Input Arguments

rangeTS	Cell range of a time series. It must be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.

Output

The output is a matrix of size 5×2, where the top portion corresponds to the estimates of the parameters α and β with errors. The "Level" is ℓ_T and the "Growth" is b_T where T denotes the last time step of the input series (see Remarks below). σ_e is the standard deviation of the error series ε_t (see Remarks below).

α	$error_\alpha$
β	$error_\beta$
Level	*0*
Growth	*0*
σ_e	*0*

Remarks

For a time series x_t, the following relation is assumed. It should be noted that stationarity is not a required unlike the ARIMA model.

$$x_t = \ell_{t-1} + b_{t-1} + \varepsilon_t$$

ℓ_t denotes an estimate of the level of the time series at time *t* and b_t denotes an estimate of the slope (or growth) of the time series at time *t*. ℓ_t and b_t obey the following serial dependences.

$$\ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t$$

$$b_t = b_{t-1} + \beta\varepsilon_t$$

Here, one step forecast is $\hat{x}_t = \mu_t = \ell_{t-1} + b_{t-1}$ and the error $\varepsilon_t = x_t - \hat{x}_t$. The parameters of this model α and β are constrained as following.

$$0 < \alpha < 1$$

$$0 < \beta < \alpha$$

Forecast of step *t*+*h* with time series data up to step *t* can easily be calculated as following.

$$\hat{x}_{t+h|t} = \ell_t + hb_t$$

See also TSHoltForecast and TSHoltSmooth.

TSHoltForecast

Purpose

Applies the Holt's linear exponential smoothing method and forecasts *N* time steps. This function can also estimate the confidence intervals.

Syntax

TSHoltForecast(*rangeTS*, *boolAscend*, *intN*, *intInterval*, *intSamples*)

Input Arguments

rangeTS	Cell range of a time series. It must be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
intN	Number of the forecast steps (*N*).
intInterval	Choice of the confidence interval. 0 for no interval, 1 for 68% interval, 2 for 95% interval and 3 for 99.7% interval.
intSamples	Number of samples for estimating the confidence interval by Monte Carlo method.

Output

When *intInterval = 0*, the output is a single column array of length $N + 1$. When *intInterval* is non zero, the output is a three column matrix of size $(N + 1) \times 3$. The columns correspond to the lower limit, middle point and the upper limit of the forecast. The step "0" of the forecast is taken from the most recent value of the input time series. The output time ordering is always descendent.

Remarks

For a time series x_t, the following relation is assumed. It should be noted that stationarity is not a required unlike the ARIMA model.

$$x_t = \ell_{t-1} + b_{t-1} + \varepsilon_t$$

ℓ_t denotes an estimate of the level of the time series at time *t* and b_t denotes an estimate of the slope (or growth) of the time series at time *t*. ℓ_t and b_t obey the following serial dependences.

$$\ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t$$

$$b_t = b_{t-1} + \beta\varepsilon_t$$

Here, one step forecast is $\hat{x}_t = \mu_t = \ell_{t-1} + b_{t-1}$ and the error $\varepsilon_t = x_t - \hat{x}_t$. Forecast of step *t*+*h* with time series data up to step *t* can easily be calculated as following.

$$\hat{x}_{t+h|t} = \ell_t + hb_t$$

The confidence intervals are estimated by Monte Carlo simulation. The larger the number of samples (*intSamples*), the more precise the interval estimates one can get. However, this comes at the expense of time. A few thousands of samples would suffice in most situations.

See also TSHoltFit and TSHoltSmooth.

TSHoltSmooth

Purpose

Applies the Holt's linear exponential smoothing method and shows the smoothed estimates of the time series.

Syntax

TSHoltSmooth(*rangeTS*, *boolAscend*)

Input Arguments

rangeTS	Cell range of a time series. It must be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.

Output

The output is a single column array of length equal to that of *rangeTS*. The output time ordering is always descendent.

Remarks

For a time series x_t, the following relation is assumed. It should be noted that stationarity is not a required unlike the ARIMA model.

$$x_t = \ell_{t-1} + b_{t-1} + \varepsilon_t$$

ℓ_t denotes an estimate of the level of the time series at time *t* and b_t denotes an estimate of the slope (or growth) of the time series at time *t*. ℓ_t and b_t obey the following serial dependences.

$$\ell_t = \ell_{t-1} + b_{t-1} + \alpha\varepsilon_t$$

$$b_t = b_{t-1} + \beta\varepsilon_t$$

Here, one step forecast is $\hat{x}_t = \mu_t = \ell_{t-1} + b_{t-1}$ and the error $\varepsilon_t = x_t - \hat{x}_t$. When applied to the lagged steps of the time series, $\hat{x}_t$ can be considered as the smoothed estimate of the time series (output of this function).

See also TSHoltFit and TSHoltForecast.

TSIsInvertibleMA

Purpose

Checks whether the Moving Average (MA) model could be inverted to the Auto-Regressive (AR) model.

Syntax

TSIsInvertibleMA(*rangeMA*)

Input Arguments

rangeMA	Cell range containing the MA parameters: $\mu, \theta_1, \theta_2, \ldots, \theta_q$. It must be a single column array of length $q + 1$.

Output

The output is 1 when the MA parameters are invertible to the equivalent AR ones. Otherwise, the output is 0.

Remarks

The MA(q) model can be expressed as following.

$$x_t = \varepsilon_t + \mu + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}$$

Now, let us define operator $\Theta(L) = 1 + \theta_1 L + \theta_2 L^2 + \cdots + \theta_q L^q$ using the lag operator L. The lag operator is such that $L^i \varepsilon_t = \varepsilon_{t-i}$. Then, the MA($q$) model can be rewritten as

$$x_t - \mu = \Theta(L)\varepsilon_t$$

The inversion consists in applying the inversion operator $\Theta(L)^{-1}$ to both sides of the equality.

$$\Theta(L)^{-1}(x_t - \mu) = \varepsilon_t$$

$$\Rightarrow \quad x_t = \varepsilon_t + \phi_0 + \phi_1 x_{t-1} + \phi_2 x_{t-2} + \cdots$$

Here, ϕ_0 and μ are connected: $\phi_0 = \mu\left(1 - \phi_1 - \phi_2 - \cdots - \phi_p\right)$.

Now, whether the inversion operator can be applied or not depends on the moduli of the "characteristic roots". Here, the characteristic roots are defined as the reciprocals of the solutions of the following "characteristic equation" (notice the positive sign in front of the sum).

$$1 + \sum_{i=1}^{q} \theta_i x^i = 0$$

There are up to q characteristic roots which we denote as λ_i. Then the operator $\Theta(L)$ can be factorized as

$$\Theta(L) = \prod_{i=1}^{q} (1 - \lambda_i L)$$

and the operator $\Theta(L)^{-1}$ can be expressed as the following.

$$\Theta(L) = \prod_{i=1}^{q} \sum_{n=0}^{\infty} {\lambda_i}^n L^n$$

Thus, the MA model is invertible to the AR model when all of the characteristic roots satisfy condition $|\lambda_i| < 1$. It is reminded that the characteristic roots can be complex numbers.

See also TSConvMAtoAR.

TSIsStationaryAR

Purpose

Checks whether the Auto Regressive (AR) model is stationary (and also whether it could be inverted to the Moving Average (MA) model).

Syntax

TSIsStationaryAR(*rangeAR*)

Input Arguments

rangeAR	Cell range containing the AR parameters: $\phi_0, \phi_1, \ldots, \phi_p$. It must be a single column or a single row array of length $p + 1$.

Output

The output is 1 when the AR parameters are stationary and invertible to the equivalent MA ones. Otherwise, the output is 0.

Remarks

The AR(p) model can be expressed as following.

$$x_t = \varepsilon_t + \phi_0 + \sum_{i=1}^{p} \phi_i x_{t-i}$$

Now, let us define operator $\Phi(L) = 1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_p L^p$ using the lag operator L. The lag operator is such that $L^i x_t = x_{t-i}$. Then, the AR(p) model can be rewritten as

$$\Phi(L)(x_t - \mu) = \varepsilon_t$$

Here, ϕ_0 and μ are connected: $\phi_0 = \mu\left(1 - \phi_1 - \phi_2 - \cdots - \phi_p\right)$. The conversion consists in applying the inversion operator $\Phi(L)^{-1}$.

$$x_t = \mu + \Phi(L)^{-1}\varepsilon_t$$

$$\Rightarrow \quad x_t = \mu + \varepsilon_t + \theta_1\varepsilon_{t-1} + \theta_2\varepsilon_{t-2} + \cdots$$

Now, whether the AR model is stationary and invertible depends on the moduli of the "characteristic roots". Here, the characteristic roots are defined as the reciprocals of the solutions of the following "characteristic equation" (notice the negative sign in front of the sum).

$$1 - \sum_{i=1}^{p} \phi_i x^i = 0$$

There are up to p characteristic roots which we denote as λ_i. Then the operator $\Phi(L)$ can be factorized as

$$\Phi(L) = \prod_{i=1}^{p}(1 - \lambda_i L)$$

and the operator $\Phi(L)^{-1}$ can be expressed as the following.

$$\Phi(L) = \prod_{i=1}^{p}\sum_{n=0}^{\infty} {\lambda_i}^n L^n$$

Thus, the AR model is stationary and invertible to the MA model when all of the characteristic roots satisfy condition $|\lambda_i| < 1$. It is reminded that the characteristic roots can be complex numbers.

See also TSConvARtoMA.

TSLjungBox

Purpose

Calculates the Ljung-Box test statistics and the *p*-values.

Syntax

TSLjungBox(*rangeTS*, *intMaxLag*)

Input Arguments

rangeTS	Cell range of a stationary time series. It must be a single column array.
intMaxLag	Maximum lag.

Output

The output consists of the Ljung-Box *Q-statistics* and the corresponding *p*-values of the selected time series. Thus, the output is a two column array of length equal to $intMaxLag$.

$$\begin{matrix} Q-stat_1 & p-value_1 \\ Q-stat_2 & p-value_2 \\ \vdots & \vdots \\ Q-stat_{Max} & p-value_{Max} \end{matrix}$$

Remarks

The test statistics are calculated using the following formula. Here, *m* ranges from *1* to *intMaxLag*. *T* denotes the total length of the time series. *P*-values are obtained from the Chi-squared (χ^2) distribution.

$$Q-stat_m = T(T+2)\sum_{\ell=1}^{m}\frac{\rho(\ell)^2}{T-\ell}$$

Ljung-Box *Q-statistics* are used to test the null hypothesis $H_0: \rho(0) = \cdots = \rho(m) = 0$ against the alternative hypothesis $H_0: \rho(\ell) \neq 0$ for some $\ell \in \{1,2,\cdots,m\}$. The null hypothesis H_0 is rejected when the *p*-value is less than the significance level α.

TSLogRate

Purpose

Converts a positive time series into another time series of logarithmic rate of change.

Syntax

TSLogRate(*rangeTS*, *boolAscend*)

Input Arguments

rangeTS	Cell range of a time series. It should be a single column array.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.

Output

The length of the output time series is one step shorter than the input. The output is always in descending time order.

Remarks

In order to apply logarithm, the input time series must be of positive numbers. The output time series y_t and the input time series x_t are related as following.

$$y_t = ln(x_{t+1}) - ln(x_t)$$

This operation converts a "price time series" to a "return time series". In many cases, while a price time series might be non-stationary, its return time series would be stationary. Models such as ARIMA and GARCH are usually applied to a return time series.

TSShowLag

Purpose

Shows lagged steps of a time series.

Syntax

TSShowLag(*rangeTS*, *boolAscend*, *intLag*)

Input Arguments

rangeTS	Cell range of a time series. It could have one or more columns (multivariate time series).
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
intLag	Number of lagged steps to be shown, counting from the end of the time series.

Output

The output cell range has the same number of columns as *rangeTS*. The number of rows is the same as *intLag*. The output is always in descending time order.

Remarks

This function is useful in showing past steps of a time series.

TSVARFit

Purpose

Calculates the parameter matrices of the Vector Auto-Regressive (VAR) model applied to a multivariate time series. When denoted as VAR(p), it means that the model has p auto-regression parameter matrices.

Syntax

TSVARFit(*rangeTS*, *boolAscend*, *intP*)

Input Arguments

rangeTS	Cell range of a multivariate time series: it must include at least two columns. Here we denote by M the number of columns. Each column is a univariate time series that makes up the multivariate time series.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
intP	Number of the auto-regression parameters (p).

Output

The output is composed of the estimates of the parameter matrices $\widetilde{\Phi}_i$ as well as a column vector $\overrightarrow{\mu}$. This is in contrast to the ARMA model where the parameters are all scalars.

If M = *number of the univariate time series* (number of columns of *rangeTS*), the parameter matrices denoted as $\widetilde{\Phi}_1, \cdots, \widetilde{\Phi}_p$ are matrices of dimension $M \times M$. Analogously, the vector denoted as $\overrightarrow{\mu}$ is a column vector of dimension M (or a matrix of dimension $M \times 1$) of which elements represent the means of each univariate time series. The output is formatted in the following way.

$$\overrightarrow{\mu} \qquad \widetilde{\Phi}_1 \qquad \cdots \qquad \widetilde{\Phi}_p$$

Thus, the dimension of the output is $M \times (M\,p + 1)$.

Remarks

VAR(p) model assumes the following serial dependence. The error vectors ($\vec{\varepsilon_t}$) are free of the serial correlation.

$$\vec{X}_t = \vec{\varepsilon_t} + \vec{\mu} + \sum_{i=1}^{p} \widetilde{\Phi}_i \left(\vec{X}_{t-i} - \vec{\mu}\right)$$

In order to get the parameters, the least square method (linear regression) is applied. It is important to remark that the vector $\vec{\mu}$ is pre-calculated (before the least square).

See also the functions TSVARForecast and TSVECMFit.

TSVARForecast

Purpose

Applies VAR(*p*) model to a multivariate time series and forecasts *N* time steps.

Syntax

TSVARForecast(*rangeTS*, *boolAscend*, *intP*, *intN*)

Input Arguments

rangeTS	Cell range of a multivariate time series: it must include at least two columns. Here we denote by *M* the number of columns. Each column is a univariate time series that makes up the multivariate time series.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
intP	Number of the parameter marices (*p*).
intN	Number of the forecast steps (*N*).

Output

The dimension of the output is $(N+1) \times M$. The "step 0" of the forecast is from the most recent value of the input time series. The output time ordering is always descendent.

Remarks

If we have the time series data up to the step t (that is up to $\overrightarrow{X}_t$) and then would like to forecast the step $t+1$ (we denote the predicted value as $\hat{X}_{t+1}$), we apply the following expression.

$$\hat{X}_{t+1} = \overrightarrow{\mu} + \tilde{\Phi}_1 \left(\overrightarrow{X}_t - \overrightarrow{\mu}\right) + \tilde{\Phi}_2 \left(\overrightarrow{X}_{t-1} - \overrightarrow{\mu}\right) + \cdots + \tilde{\Phi}_p \left(\overrightarrow{X}_{t-p+1} - \overrightarrow{\mu}\right)$$

In order to predict the next step $t+2$, we would feed back $\hat{X}_{t+1}$ and apply the following.

$$\hat{X}_{t+2} = \vec{\mu} + \widetilde{\Phi}_1 \left(\hat{X}_{t+1} - \vec{\mu}\right) + \widetilde{\Phi}_2 \left(\vec{X}_t - \vec{\mu}\right) + \cdots + \widetilde{\Phi}_p \left(\vec{X}_{t-p+2} - \vec{\mu}\right)$$

Then, we can proceed analogously to advance any number of steps into the future.

For more details of the VAR(*p*) model, refer to the remarks on the function TSVARFit.

TSVARSimul

Purpose

Simulates *N* steps of the VAR(*p*) process. VAR(*p*) models a multivariate time series and has *p* parameter matrices.

Syntax

TSVARSimul(*rangeVAR*, *rangeCov*, *intSeed*, *intN*)

Input Arguments

rangeVAR	Cell range containing the VAR parameter matrices: $\widetilde{\Phi}_1, \cdots, \widetilde{\Phi}_p$ as well as the mean vector $\vec{\mu}$. The content of the range has to be formatted as the output of the function TSVARFit: in a single continuous block, the leftmost portion corresponds to the column vector $\vec{\mu}$ and then the matrices $\widetilde{\Phi}_1, \cdots, \widetilde{\Phi}_p$ should follow. Thus, the total dimension should be $M \times (M\,p + 1)$. There is no need to specify *M* or *p* separately. This function extracts the values of *M* and *p* from the size of the *rangeVAR*.
rangeCov	Cell range containing the covariance matrix $\tilde{\Sigma}$ of the error series $\vec{\varepsilon_t}$. The dimension should be $M \times M$.
intSeed	Seed for the random number generator. It should be an integer number such as *1234*, *4321*, … etc.
intN	Number of the simulation steps (*N*).

Output

The output is contained within a matrix of dimension equal to $N \times M$. The time ordering is always descendent.

Remarks

The starting point (step "1") is the same as the mean vector $\vec{\mu}$.

For more details of the VAR(p) model, refer to the remarks on the function TSVARFit.

The covariance matrix has to be a valid one: symmetric and positive definite.

TSVECMFit

Purpose

Calculates the parameter matrices of the Vector Error Correction Model (VECM) applied to a multivariate time series. When denoted as VECM(p), it means that the model has p parameter matrices.

Syntax

TSVECMFit(*rangeTS*, *boolAscend*, *intP*)

Input Arguments

rangeTS	Cell range of a multivariate time series: it must include at least two columns. Here we denote by M the number of columns. Each column is a univariate time series that makes up the multivariate time series.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
intP	VECM order denoted as p. It has to be 0 or larger. When $p = 0$, the output is the $\vec{\alpha}$ vector and the $\tilde{\Pi}$ matrix only (without $\tilde{\Phi}_i^*$).

Output

The output consists of the estimates of the parameter matrices $\tilde{\Pi}$ and $\tilde{\Phi}_i^*$, as well as the column vector $\vec{\alpha}$. For the VECM, the "parameters" are actually matrices.

If M = *number of the univariate time series* (number of columns of *rangeTS*), the parameter matrices denoted as $\tilde{\Pi}$ and $\tilde{\Phi}_1^*, \cdots, \tilde{\Phi}_p^*$ are all matrices of dimension $M \times M$. Analogously, the vector of the multivariate time series $\vec{\alpha}$ is a column vector of dimension M (or a matrix of dimension $M \times 1$). The output is formatted in the following way.

$$\vec{\alpha} \qquad \tilde{\Pi} \qquad \tilde{\Phi}_1^* \qquad \cdots \qquad \tilde{\Phi}_p^*$$

Thus, the dimension of the output is $M \times (M\,(p+1)+1)$.

Remarks

VECM(p) assumes the following serial dependence. The error vectors ($\vec{\varepsilon_t}$) are free of the serial correlation.

$$\vec{\Delta X}_t = \vec{\varepsilon_t} + \vec{\alpha} + \tilde{\Pi}\,\vec{X}_{t-1} + \sum_{i=1}^{p} \tilde{\Phi}_i^*\,\vec{\Delta X}_{t-i}$$

The parameters of the VECM and those of the VAR model can be related. In fact, if we consider VAR($p+1$) model that assumes $\vec{X}_t = \vec{\varepsilon_t} + \vec{\mu} + \sum_{i=1}^{p+1} \tilde{\Phi}_i \left(\vec{X}_{t-i} - \vec{\mu}\right)$, we can establish the following relations. It can also be noticed that $\vec{\alpha} = -\tilde{\Pi}\vec{\mu}$.

$$\vec{\alpha} = \left(\tilde{1} - \sum_{i=1}^{p+1} \tilde{\Phi}_i\right)\vec{\mu}$$

$$\tilde{\Pi} = \left(\sum_{i=1}^{p+1} \tilde{\Phi}_i\right) - \tilde{1}$$

$$\tilde{\Phi}_j^* = -\sum_{i=j+1}^{p+1} \tilde{\Phi}_i$$

See also the function TSVARFit.

TSVECMForecast

Purpose

Applies VECM(*p*) model to a multivariate time series and forecasts *N* time steps.

Syntax

TSVECMForecast(*rangeTS, boolAscend, intP, intN, boolDifference*)

Input Arguments

rangeTS	Cell range of a multivariate time series: it must include at least two columns. Here we denote by *M* the number of columns. Each column is a univariate time series that makes up the multivariate time series.
boolAscend	Whether the time series is in ascending order or not (descending order). *TRUE* means ascending order: the most recent step at the top.
intP	VECM order denoted as *p*. It has to be 0 or larger.
intN	Number of the forecast steps (*N*).
boolDifference	When it's *TRUE*, the output corresponds to the difference series $\overrightarrow{\Delta X}_t$. When it's *FALSE*, the output is that of the series $\overrightarrow{X}_t$.

Output

The dimension of the output is $(N+1) \times M$. The "step 0" of the forecast is from the most recent value of the input time series. The output time ordering is always descendent.

Remarks

For more details of the VECM(*p*), refer to the remarks on the function TSVECMFit.

TSVECMSimul

Purpose

Simulates *N* steps of a time series process governed by the VECM(*p*). VECM(*p*) is a model of the multivariate time series that has *p* parameter matrices.

Syntax

TSVECMSimul(*rangeVECM, rangeCov, intSeed, intN, boolDifference*)

Input Arguments

rangeVECM	Cell range containing the VECM parameter matrices: $\widetilde{\Pi}$ and $\widetilde{\Phi}_1^*, \cdots, \widetilde{\Phi}_p^*$ as well as the vector $\overrightarrow{\alpha}$. The content of the range has to be formatted as the output of the function TSVECMFit: in a single continuous block, the leftmost portion corresponds to the column vector $\overrightarrow{\alpha}$ and then the matrices $\widetilde{\Pi}$ and $\widetilde{\Phi}_1^*, \cdots, \widetilde{\Phi}_p^*$ should follow. Thus, the total dimension should be $M \times (M\,(p+1)+1)$. There is no need to specify M or p separately. This function extracts the values of M and p from the size of the *rangeVECM*.
rangeCov	Cell range containing the covariance matrix $\tilde{\Sigma}$ of the error series $\overrightarrow{\varepsilon_t}$. The dimension should be $M \times M$.
intSeed	Seed for the random number generator. It should be an integer number such as *1234*, *4321*, … etc.
intN	Number of the simulation steps (*N*).
boolDifference	When it's *TRUE*, the output corresponds to the difference series $\overrightarrow{\Delta X}_t$. When it's *FALSE*, the output is that of the series $\overrightarrow{X}_t$.

Output

The output is contained within a matrix of dimension equal to $N \times M$. The time ordering is always descendent.

Remarks

For more details of the VECM(p), refer to the remarks on the function TSVECMFit. The covariance matrix has to be a valid one: symmetric and positive definite.

Getting Help

PrimaXL comes with 3 manuals, each one with its intended purpose. These manuals should provide most of the help you will need.

1. Quick Start Guide (Free)

   It guides the user through the installation process. It provides a quick introduction and overview of the PrimaXL ribbon menu and functions.

2. User's Reference Manual (Free)

   It provides detailed explanations of the PrimaXL functions. The functions are listed in the alphabetical order for easy consultation.

3. User's Guide (Only with the purchase)

   It explains the use of PrimaXL with examples. It is intended to serve both as an instructional resource and as a practical help. As supplementary material, there are several Excel workbooks with sample data and examples.

If your question is still unanswered even after reading the manuals, please feel free to contact us by sending an email to:

contact@fianresearch.com

Also, our web-site *www.fianresearch.com* will show any updates, fixes of the existing version and the release of new versions.

Thank you !


FIAN
RESEARCH