Package 'bimets'

May 15, 2023

Type Package

Title Time Series and Econometric Modeling

Version 3.0.1

Date 2023-05-09

Maintainer Andrea Luciani <andrea.luciani@bancaditalia.it>

Author Andrea Luciani [aut, cre] (<https://orcid.org/0000-0002-7372-358X>),
Roberto Stok [aut],
Bank of Italy [cph]

ByteCompile no

Description Time series analysis, (dis)aggregation and manipulation, e.g. time series extension, merge, projection, lag, lead, delta, moving and cumulative average and product, selection by index, date and year-period, conversion to daily, monthly, quarterly, (semi)annually. Simultaneous equation models definition, estimation, simulation and forecasting with coefficient restrictions, error autocorrelation, exogenization, add-factors, impact and interim multipliers analysis, conditional equation evaluation, endogenous targeting and model renormalization, structural stability, stochastic simulation and forecast, optimal control.

Depends R (>= 4.0), xts, zoo

Imports stats

LazyData true

License GPL-3

Encoding UTF-8

BugReports https://github.com/andrea-luciani/bimets/issues

URL https://github.com/andrea-luciani/bimets

NeedsCompilation no

Repository CRAN

Date/Publication 2023-05-15 16:20:02 UTC

R topics documented:

`bimets-package` *bimets :: Time Series And Econometric Modeling In R*

Description

BIMETS is a software framework developed by using R language and designed for time series analysis and econometric modeling, which allows creating and manipulating time series, specifying simultaneous equation models of any size by using a kind of high-level description language, and performing model estimation and structural stability analysis, deterministic and stochastic simulation and forecasting, optimal control.

Besides, BIMETS computational capabilities provide many tools to pre-process data and post-process results, designed for statisticians and economists. These operations are fully integrated with the R environment.

The package can be installed and loaded in R with the following commands (with R> as the R command prompt):

```
R> install.packages('bimets')
R> library(bimets)
```

If you have general questions about using BIMETS, or for bug reports, please use the git issue tracker or write to the maintainer.

TIME SERIES

BIMETS supports daily, weekly, monthly, quarterly, semiannual and yearly time series. Time series with a frequency of 24 and 36 periods per year are also supported. Time series are created by the `TIMESERIES` function.

Example:

```
R> #yearly time series
R> myTS <- TIMESERIES(1:10,START=as.Date('2000-01-01'),FREQ=1)

R> #monthly time series
R> myTS <- TIMESERIES(1:10,START=c(2002,3),FREQ='M')
```

The main BIMETS time series capabilities are:

- *Indexing*
- *Aggregation / Disaggregation*
- *Manipulation*

Time Series Indexing

The BIMETS package extends R indexing capabilities in order to ease time series analysis and manipulation. Users can access and modify time series data:

- *by date*: users can select and modify a single observation by date by using the syntax `ts['Date']`, or multiple observations by using `ts['StartDate/EndDate']`;

- *by year-period*: users can select and modify observations by providing the year and the period requested, i.e. `ts[[Year,Period]]`;

- *by observation index*: users can select and modify observations by simply providing the array of requested indices, i.e. `ts[indices]`;

Example:

```
R> #create a daily time series
R> myTS <- TIMESERIES((1:100),START=c(2000,1),FREQ='D')

R> myTS[1:3]                      #get first three obs.
R> myTS['2000-01-12']             #get Jan 12, 2000 data
R> myTS['2000-02-03/2000-02-14']  #get Feb 3 up to Feb 14
R> myTS[[2000,14]]                #get year 2000 period 14

R> myTS['2000-01-15'] <- NA        #assign to Jan 15, 2000
R> myTS[[2000,42]] <- NA           #assign to Feb 11, 2000
R> myTS[[2000,100]] <- c(-1,-2,-3) #extend time series starting from period 100
```

Time Series Aggregation / Disaggregation

The BIMETS package provides advanced (dis)aggregation capabilities, having linear interpolation capabilities in disaggregation, and aggregation functions (e.g. STOCK, SUM, AVE, etc.) while reducing the time series frequency.

Example:

```
R> #create a monthly time series
R> myMonthlyTS <- TIMESERIES(1:100,START=c(2000,1),FREQ='M')

R> #convert to yearly time series by using the average as aggregation fun
R> myYearlyTS <- YEARLY(myMonthlyTS,'AVE')

R> #convert to daily by using central interpolation as disaggregation fun
R> myDailyTS <- DAILY(myMonthlyTS,'INTERP_CENTER')
```

Time Series Manipulation

The BIMETS package provides, among others, the following time series manipulation capabilities:

- Time series extension TSEXTEND
- Time series merging TSMERGE
- Time series projection TSPROJECT
- Lag TSLAG
- Lead TSLEAD
- Lag differences: standard, percentage, and logarithmic TSDELTA TSDELTAP TSDELTALOG
- Cumulative product CUMPROD
- Cumulative sum CUMSUM
- Moving average MOVAVG
- Moving sum MOVSUM
- Time series data presentation TABIT

Example:

```
R> #define two time series
R> myTS1 <- TIMESERIES(1:100,START=c(2000,1),FREQ='M')
R> myTS2 <- TIMESERIES(-(1:100),START=c(2005,1),FREQ='M')

R> #extend time series up to Apr 2020 with quadratic formula
R> myExtendedTS <- TSEXTEND(myTS1,UPTO=c(2020,4),EXTMODE='QUADRATIC')

R> #merge two time series with sum
R> myMergedTS <- TSMERGE(myExtendedTS,myTS2,fun='SUM')
```

```
R> #project time series on arbitrary time range
R> myProjectedTS <- TSPROJECT(myMergedTS,TSRANGE=c(2004,2,2006,4))

R> #lag and delta% time series
R> myLagTS <- TSLAG(myProjectedTS,2)
R> myDeltaPTS <- TSDELTAP(myLagTS,2)

R> #moving average
R> myMovAveTS <- MOVAVG(myDeltaPTS,5)

R> #print data
R> TABIT(myMovAveTS,
      myTS1,
      TSRANGE=c(2004,8,2004,12)
      )

    Date, Prd., myMovAveTS , myTS1
Aug 2004, 8   ,            , 56
Sep 2004, 9   ,            , 57
Oct 2004, 10  , 3.849002   , 58
Nov 2004, 11  , 3.776275   , 59
Dec 2004, 12  , 3.706247   , 60
```

ECONOMETRIC MODELING

BIMETS econometric modeling capabilities comprehend:

- *Model Definition Language*
- *Estimation*
- *Structural Stability*
- *Simulation*
- *Stochastic Simulation*
- *Multipliers Analysis*
- *Endogenous Targeting*
- *Optimal Control*

We will go through each item of the list with a simple Klein model example (ref: *"Economic Fluctuations in the United States 1921-1941"* by L. R. Klein, Wiley and Sons Inc., New York, 1950).

Model Definition Language

BIMETS provides a language to specify an econometric model unambiguously. This section describes how to create a model and its general structure. The specification of an econometric model is translated and identified by keyword statements which are grouped in a model file, i.e. a plain

text file or a character variable with a specific syntax. Collectively, these keyword statements constitute the BIMETS Model Description Language (from now on MDL). The model specifications consist of groups of statements. Each statement begins with a keyword. The keyword classifies the component of the model which is being specified.

Below is an example of Klein's model, which can either be stored in an R variable of class character or in a plain text file with an MDL compliant syntax.

The content of the *klein1.txt* variable is:

```
R> klein1.txt <- "
MODEL

COMMENT> Consumption
BEHAVIORAL> cn
TSRANGE 1921 1 1941 1
EQ> cn =  a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)
COEFF> a1 a2 a3 a4

COMMENT> Investment
BEHAVIORAL> i
TSRANGE 1921 1 1941 1
EQ> i = b1 + b2*p + b3*TSLAG(p,1) + b4*TSLAG(k,1)
COEFF> b1 b2 b3 b4

COMMENT> Demand for Labor
BEHAVIORAL> w1
TSRANGE 1921 1 1941 1
EQ> w1 = c1 + c2*(y+t-w2) + c3*TSLAG(y+t-w2,1)+c4*time
COEFF> c1 c2 c3 c4

COMMENT> Gross National Product
IDENTITY> y
EQ> y = cn + i + g - t

COMMENT> Profits
IDENTITY> p
EQ> p = y - (w1+w2)

COMMENT> Capital Stock
IDENTITY> k
EQ> k = TSLAG(k,1) + i

END
"
```

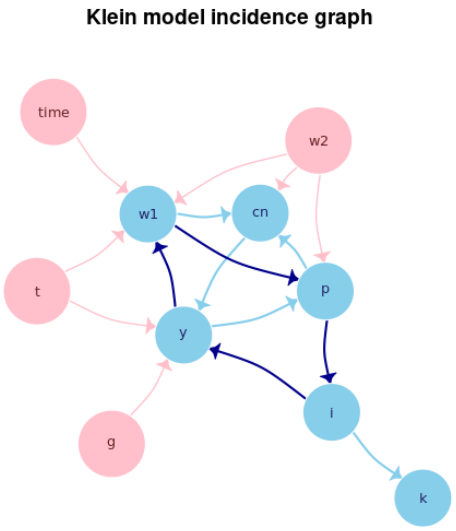
Given:

- cn as *Private Consumption Expenditure*;

- `i` as *Investment*;
- `w1` as *Wage Bill of the Private Sector (Demand for Labor)*;
- `p` as *Profits*;
- `k` as *Stock of Capital Goods*;
- `y` as *Gross National Product*;
- `w2` as *Wage Bill of the Government Sector*;
- `time` as an annual index of the passage of time;
- `g` as *Government Expenditure plus Net Exports*;
- `t` as *Business Taxes*.

`a1, a2, a3, a4, b1, b2, b3, b4, c1, c2, c3, c4` are coefficients to be estimated.

This system has only six equations, three of which must be fitted to assess the coefficients. It may not seem challenging to solve this system. However, the objective complexity emerges if you look at the incidence graph in the following figure, wherein endogenous variables are plotted in blue and exogenous variables are plotted in pink.



Each edge states a simultaneous dependence from a variable to another, e.g. the `w1` equation depends on the current value of the `time` time series; complexity arises because in this model there are several circular dependencies, one of which is plotted in dark blue.

A circular dependency in the incidence graph of a model implies that the model is a *simultaneous* equations model. It must be estimated using ad-hoc procedures; moreover, it can be simulated, e.g. performing a forecast, only using an iterative algorithm.

As shown in the code, the model definition is quite intuitive. The first keyword is `MODEL`, while at the end of the model definition we can find the `END` keyword. Available tags in the definition of a generic BIMETS model are:

- **EQUATION>** or **BEHAVIORAL>** indicate the beginning of a series of keyword statements describing a behavioral equation;

- **IDENTITY>** indicates the beginning of a series of keyword statements describing an identity or technical equation;

- **EQ>** specifies the mathematical expression for a behavioral equation or an identity equation;

- **COEFF>** specifies the coefficient names used in the EQ> keyword statement of a behavioral equation;

- **ERROR>** specifies an autoregressive process of a given order for the regression error;

- **PDL>** defines an Almon polynomial distributed lag;

- **RESTRICT>** is a keyword that can be used to specify linear coefficient restrictions;

- **IF>** is used to conditionally evaluate an identity during a simulation, depending on a logical expression's value. Thus, it is possible to have a model alternating between two or more identity specifications for each simulation period, depending upon results from other equations;

- **IV>** specifies the mathematical expression for an instrumental variable used in a behavioral equation;

- **COMMENT>** can be used to insert comments into a model;

The mathematical expression in the EQ> and IF> definitions can include the standard arithmetic operators, parentheses, and the following MDL time series functions:

- `TSLAG(ts,i)`: lag the `ts` time series by `i`-periods;

- `TSDELTA(ts,i)`: `i`-periods difference of the `ts` time series;

- `TSDELTAP(ts,i)`: `i`-periods percentage difference of the `ts` time series;

- `TSDELTALOG(ts,i)`: `i`-periods logarithmic difference of the `ts` time series;

- `MOVAVG(ts,i)`: `i`-periods moving average of the `ts` time series;

- `MOVSUM(ts,i)`: `i`-periods moving sum of the `ts` time series;

- `LOG(ts)`: log of the `ts` time series.;

- EXP(ts): exponential of the ts time series;

- ABS(ts): absolute values of the ts time series;

More details are available in MDL and LOAD_MODEL help pages. LOAD_MODEL() is the BIMETS function that reads an MDL model file and creates an equivalent R data structure.

Back to Klein's model example, the BIMETS LOAD_MODEL function reads the *klein1.txt* model as previously defined:

```
R> kleinModel <- LOAD_MODEL(modelText = klein1.txt)

Analyzing behaviorals...
Analyzing identities...
Optimizing...
Loaded model "klein1.txt":
    3 behaviorals
    3 identities
    12 coefficients
...LOAD MODEL OK
```

As shown in the output, BIMETS counted 3 behavioral equations, 3 identities and 12 coefficients. Now in the R session there is a variable named *kleinModel* that contains the model structure defined in the *klein1.txt* variable. From now on, users can ask BIMETS about any details of this model.

For example, to gather information on the "cn" *Consumption* behavioral equation:

```
R> kleinModel$behaviorals$cn

$eq
[1] "cn=a1+a2*p+a3*TSLAG(p,1)+a4*(w1+w2)"

$eqCoefficientsNames
[1] "a1" "a2" "a3" "a4"

$eqComponentsNames
[1] "cn" "p" "w1" "w2"

$tsrange
[1] 1921 1 1941 1

$eqRegressorsNames
[1] "1" "p" "TSLAG(p,1)" "(w1+w2)"

$eqSimExp
expression(cn[2, ] = cn__ADDFACTOR[2, ] + +cn__a1 * 1 + cn__a2 *
p[2, ] + cn__a3 * (p[1, ]) + cn__a4 * (w1[2, ] + w2[2, ]))
```

```
etc...
```

Users can always read (or carefully change) any model parameters. The LOAD_MODEL function parses behavioral and identity expressions of the MDL definition, but it also does a significant optimization. Properly reordering the model equations is a key preparatory step in the later phase of the simulation, in order to guarantee performance and convergence, if any, with the aim of minimizing the number of *feedback* endogenous variables (see SIMULATE).

The LOAD_MODEL function builds the model's incidence matrix, and uses this matrix to calculate the proper evaluation order of the model equations during the simulation.

Back to the Klein's model example, the incidence matrix and the reordering of the equations are stored in the following variables:

```
R> kleinModel$incidence_matrix

   cn i w1 y p k
cn 0  0 1  0 1 0
i  0  0 0  0 1 0
w1 0  0 0  1 0 0
y  1  1 0  0 0 0
p  0  0 1  1 0 0
k  0  1 0  0 0 0

R> kleinModel$vpre

NULL

R> kleinModel$vsim

[1] "w1" "p" "cn" "i" "y"

R> kleinModel$vfeed

[1] "y"

R> kleinModel$vpost

[1] "k"
```

While simulating the Klein's model, BIMETS will iterate on the computation of, in order, `w1 -> p -> cn -> i -> y` (the `vsim` variables), by looking for convergence on `y` (the `vfeed` variable, only one in this example) that is the feedback variable. If the convergence is achieved, it will calculate `k` (the `vpost` variable). The `vpre` array in this example is empty; therefore, no equation has to be evaluated before the iterative algorithm.

More details on the equations reordering are available in SIMULATE and LOAD_MODEL help pages.

Once the model has been parsed, users need to load the data of all the time series involved in the model, by using the LOAD_MODEL_DATA function. In the following example, the code defines a list of time series and loads this list into the Klein's model previously defined:

```
R> kleinModelData <- list(
    cn  =TIMESERIES(39.8,41.9,45,49.2,50.6,52.6,55.1,56.2,57.3,57.8,
                    55,50.9,45.6,46.5,48.7,51.3,57.7,58.7,57.5,61.6,65,69.7,
                    START=c(1920,1),FREQ=1),
    g   =TIMESERIES(4.6,6.6,6.1,5.7,6.6,6.5,6.6,7.6,7.9,8.1,9.4,10.7,
                    10.2,9.3,10,10.5,10.3,11,13,14.4,15.4,22.3,
                    START=c(1920,1),FREQ=1),
    i   =TIMESERIES(2.7,-.2,1.9,5.2,3,5.1,5.6,4.2,3,5.1,1,-3.4,-6.2,
                    -5.1,-3,-1.3,2.1,2,-1.9,1.3,3.3,4.9,
                    START=c(1920,1),FREQ=1),
    k   =TIMESERIES(182.8,182.6,184.5,189.7,192.7,197.8,203.4,207.6,
                    210.6,215.7,216.7,213.3,207.1,202,199,197.7,199.8,
                    201.8,199.9,201.2,204.5,209.4,
                    START=c(1920,1),FREQ=1),
    p   =TIMESERIES(12.7,12.4,16.9,18.4,19.4,20.1,19.6,19.8,21.1,21.7,
                    15.6,11.4,7,11.2,12.3,14,17.6,17.3,15.3,19,21.1,23.5,
                    START=c(1920,1),FREQ=1),
    w1  =TIMESERIES(28.8,25.5,29.3,34.1,33.9,35.4,37.4,37.9,39.2,41.3,
                    37.9,34.5,29,28.5,30.6,33.2,36.8,41,38.2,41.6,45,53.3,
                    START=c(1920,1),FREQ=1),
    y   =TIMESERIES(43.7,40.6,49.1,55.4,56.4,58.7,60.3,61.3,64,67,57.7,
                    50.7,41.3,45.3,48.9,53.3,61.8,65,61.2,68.4,74.1,85.3,
                    START=c(1920,1),FREQ=1),
    t   =TIMESERIES(3.4,7.7,3.9,4.7,3.8,5.5,7,6.7,4.2,4,7.7,7.5,8.3,5.4,
                    6.8,7.2,8.3,6.7,7.4,8.9,9.6,11.6,
                    START=c(1920,1),FREQ=1),
    time=TIMESERIES(NA,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,
                    1,2,3,4,5,6,7,8,9,10,
                    START=c(1920,1),FREQ=1),
    w2  =TIMESERIES(2.2,2.7,2.9,2.9,3.1,3.2,3.3,3.6,3.7,4,4.2,4.8,
                    5.3,5.6,6,6.1,7.4,6.7,7.7,7.8,8,8.5,
                    START=c(1920,1),FREQ=1)
)

R> kleinModel <- LOAD_MODEL_DATA(kleinModel,kleinModelData)
```

Since time series and other data (e.g. regressor coefficients, error coefficients, constant adjustments, targets, instruments, etc...) are stored in the model object, users can define multiple model objects - each with its own arbitrary data - in the same R session. BIMETS makes it possible to estimate, simulate and compare results from different models with different data sets. Furthermore, users can easily save an estimated or a simulated model as a standard R variable, thus reloading it later, having all available data and time series stored in it, i.e. endogenous and exogenous time series, estimated coefficients, constant adjustments, simulation options, simulated time series, calculated instruments, targets, etc. (see also SIMULATE, STOCHSIMULATE, RENORM, OPTIMIZE)

An advanced MDL model example follows:

```
R> #KLEIN MODEL WITH AUTOCORRELATION, RESTRICTIONS,
R> #CONDITIONAL EVALUATIONS AND LHS FUNCTIONS

R> lhsKlein1.txt <- "
MODEL

COMMENT> Modified Klein Model 1 of the U.S. Economy with PDL,
COMMENT> autocorrelation on errors, restrictions,
COMMENT> conditional evaluations and LHS functions on EQ

COMMENT> Exp Consumption
BEHAVIORAL> cn
TSRANGE 1925 1 1941 1
EQ> EXP(cn) = a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)
COEFF> a1 a2 a3 a4
ERROR> AUTO(2)

COMMENT> Log Investment
BEHAVIORAL> i
TSRANGE 1925 1 1941 1
EQ> LOG(i) = b1 + b2*p + b3*TSLAG(p,1) + b4*TSLAG(k,1)
COEFF> b1 b2 b3 b4
RESTRICT> b2 + b3 = 1

COMMENT> Demand for Labor
BEHAVIORAL> w1
TSRANGE 1925 1 1941 1
EQ> w1 = c1 + c2*(TSDELTA(y)+t-w2) + c3*TSLAG(TSDELTA(y)+t-w2,1)+c4*time
COEFF> c1 c2 c3 c4
PDL> c3 1 3

COMMENT> Delta Gross National Product
IDENTITY> y
EQ> TSDELTA(y) = EXP(cn) + LOG(i) + g - t

COMMENT> Profits
IDENTITY> p
EQ> p = TSDELTA(y) - (w1+w2)

COMMENT> Capital Stock with switches
IDENTITY> k
EQ> k = TSLAG(k,1) + LOG(i)
IF> LOG(i) > 0
IDENTITY> k
EQ> k = TSLAG(k,1)
```

```
IF> LOG(i) <= 0

END"
```

See MDL help page for details.

Estimation

The BIMETS ESTIMATE function estimates equations that are linear in the coefficients, as specified in the behavioral equations of the model object. Coefficients can be estimated for single equations or blocks of simultaneous equations. The estimation function supports:

- *Ordinary Least Squares*;
- *Instrumental Variables*;
- *Deterministic linear restrictions on the coefficients*;
- *Almon Polynomial Distributed Lags*;
- *Autocorrelation of the errors*;
- *Structural stability analysis (Chow tests)*;

Restrictions procedure derives from Lagrange Multipliers' theory, while the Cochrane-Orcutt method allows accounting for residuals autocorrelation.

The estimation of the previously defined Klein's model is shown in the following example:

```
R> kleinModel <- ESTIMATE(kleinModel)
```

Users can also estimate a selection of behavioral equations:

```
R> kleinModel <- ESTIMATE(kleinModel,eqList=c('cn'))

Estimate the Model klein1.txt:
the number of behavioral equations to be estimated is 1.
The total number of coefficients is 4.

----------------------------------------

BEHAVIORAL EQUATION: cn
Estimation Technique: OLS

cn                  = 16.2366
                        T-stat. 12.46382 ***

                    + 0.1929344 p
                        T-stat. 2.115273 *

                    + 0.0898849 TSLAG(p,1)
                        T-stat. 0.9915824
```

```
                    + 0.7962187 (w1+w2)
                        T-stat. 19.93342 ***


STATs:
R-Squared                       : 0.9810082
Adjusted R-Squared              : 0.9776567
Durbin-Watson Statistic         : 1.367474
Sum of squares of residuals     : 17.87945
Standard Error of Regression    : 1.02554
Log of the Likelihood Function  : -28.10857
F-statistic                     : 292.7076
F-probability                   : 7.993606e-15
Akaike's IC                     : 66.21714
Schwarz's IC                    : 71.43975
Mean of Dependent Variable      : 53.99524
Number of Observations          : 21
Number of Degrees of Freedom    : 17
Current Sample (year-period)    : 1921-1 / 1941-1


Signif. codes: *** 0.001 ** 0.01 * 0.05

...ESTIMATE OK
```

A similar output is shown for each estimated regression. Once the estimation is completed, coefficient values, residuals, statistics, etc. are stored in the model object.

```
R> #print estimated coefficients
R> kleinModel$behaviorals$cn$coefficients

   [,1]
a1 16.2366003
a2 0.1929344
a3 0.0898849
a4 0.7962187

R> #print residuals
R> kleinModel$behaviorals$cn$residuals

Time Series:
Start = 1921
End = 1941
Frequency = 1
[1]  -0.323893544 -1.250007790 -1.565741401 -0.493503129  0.007607907
[6]   0.869096295  1.338476868  1.054978943 -0.588557053  0.282311734
[11] -0.229653489 -0.322131892  0.322281007 -0.058010257 -0.034662717
[16]  1.616497310 -0.435973632  0.210054350  0.989201310  0.785077489
[21] -2.173448309
```

```
R> #print a selection of estimate statistics
R> kleinModel$behaviorals$cn$statistics$DegreesOfFreedom

[1] 17

R> kleinModel$behaviorals$cn$statistics$StandardErrorRegression

[1] 1.02554

R> kleinModel$behaviorals$cn$statistics$CoeffCovariance

    a1            a2            a3            a4
a1  1.6970227814  0.0005013886 -0.0177068887 -0.0329172192
a2  0.0005013886  0.0083192948 -0.0052704304 -0.0013188865
a3 -0.0177068887 -0.0052704304  0.0082170486 -0.0006710788
a4 -0.0329172192 -0.0013188865 -0.0006710788  0.0015955167

R> kleinModel$behaviorals$cn$statistics$AdjustedRSquared
[1] 0.9776567

R> kleinModel$behaviorals$cn$statistics$LogLikelihood
[1] -28.10857
```

Below is an example of a model estimation that presents coefficient restrictions, PDL, error autocorrelation, and conditional equation evaluations:

```
R> #define model
R> advancedKlein1.txt <-
"MODEL

COMMENT> Modified Klein Model 1 of the U.S. Economy with PDL,
COMMENT> autocorrelation on errors, restrictions and
COMMENT> conditional equation evaluations

COMMENT> Consumption with autocorrelation on errors
BEHAVIORAL> cn
TSRANGE 1923 1 1940 1
EQ> cn =  a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)
COEFF> a1 a2 a3 a4
ERROR> AUTO(2)

COMMENT> Investment with restrictions
BEHAVIORAL> i
TSRANGE 1923 1 1940 1
EQ> i = b1 + b2*p + b3*TSLAG(p,1) + b4*TSLAG(k,1)
COEFF> b1 b2 b3 b4
RESTRICT> b2 + b3 = 1
```

```
COMMENT> Demand for Labor with PDL
BEHAVIORAL> w1
TSRANGE 1923 1 1940 1
EQ> w1 = c1 + c2*(y+t-w2) + c3*TSLAG(y+t-w2,1) + c4*time
COEFF> c1 c2 c3 c4
PDL> c3 1 2

COMMENT> Gross National Product
IDENTITY> y
EQ> y = cn + i + g - t

COMMENT> Profits
IDENTITY> p
EQ> p = y - (w1+w2)

COMMENT> Capital Stock with IF switches
IDENTITY> k
EQ> k = TSLAG(k,1) + i
IF> i > 0
IDENTITY> k
EQ> k = TSLAG(k,1)
IF> i <= 0

END"

R> #load model and data
R> advancedKleinModel <- LOAD_MODEL(modelText=advancedKlein1.txt)

Analyzing behaviorals...
Analyzing identities...
Optimizing...
Loaded model "advancedKlein1.txt":
    3 behaviorals
    3 identities
    12 coefficients
...LOAD MODEL OK

R> advancedKleinModel <- LOAD_MODEL_DATA(advancedKleinModel,kleinModelData)

Load model data "kleinModelData" into model "advancedKlein1.txt"...
...LOAD MODEL DATA OK

R> #estimate model
R> advancedKleinModel <- ESTIMATE(advancedKleinModel)

Estimate the Model advancedKlein1.txt:
the number of behavioral equations to be estimated is 3.
```

```
The total number of coefficients is 13.

----------------------------------------

BEHAVIORAL EQUATION: cn
Estimation Technique: OLS
Autoregression of Order  2  (Cochrane-Orcutt procedure)

Convergence was reached in  6  /  20  iterations.


cn                  =   14.82685
                        T-stat. 7.608453    ***

                    +   0.2589094   p
                        T-stat. 2.959808    *

                    +   0.01423821  TSLAG(p,1)
                        T-stat. 0.1735191

                    +   0.8390274   (w1+w2)
                        T-stat. 14.67959    ***

ERROR STRUCTURE:  AUTO(2)

AUTOREGRESSIVE PARAMETERS:
Rho             Std. Error      T-stat.
 0.2542111       0.2589487       0.9817045
-0.05250591      0.2593578      -0.2024458


STATs:
R-Squared                       : 0.9826778
Adjusted R-Squared              : 0.9754602
Durbin-Watson Statistic         : 2.256004
Sum of squares of residuals     : 8.071633
Standard Error of Regression    : 0.8201439
Log of the Likelihood Function : -18.32275
F-statistic                     : 136.1502
F-probability                   : 3.873514e-10
Akaike's IC                     : 50.6455
Schwarz's IC                    : 56.8781
Mean of Dependent Variable      : 54.29444
Number of Observations          : 18
Number of Degrees of Freedom    : 12
Current Sample (year-period)    : 1923-1 / 1940-1
```

```
Signif. codes:   *** 0.001  ** 0.01  * 0.05



----------------------------------------

BEHAVIORAL EQUATION: i
Estimation Technique: OLS

i                   =   0.5348561
                        T-stat. 0.06197295

                    +   0.6267204   p
                        T-stat. 4.835884    ***

                    +   0.3732796   TSLAG(p,1)
                        T-stat. 2.88029     *

                    -   0.0796483   TSLAG(k,1)
                        T-stat. -1.871304

RESTRICTIONS:
b2+b3=1

RESTRICTIONS F-TEST:
F-value            : 5.542962
F-prob(1,14)       : 0.03368297


STATs:
R-Squared                      : 0.9009016
Adjusted R-Squared             : 0.8876885
Durbin-Watson Statistic        : 1.284709
Sum of squares of residuals    : 23.40087
Standard Error of Regression   : 1.249023
Log of the Likelihood Function : -27.90251
F-statistic                    : 68.18238
F-probability                  : 2.954599e-08
Akaike's IC                    : 63.80501
Schwarz's IC                   : 67.3665
Mean of Dependent Variable     : 1.111111
Number of Observations         : 18
Number of Degrees of Freedom   : 15
Current Sample (year-period)   : 1923-1 / 1940-1


Signif. codes:   *** 0.001  ** 0.01  * 0.05
```

```
----------------------------------------

BEHAVIORAL EQUATION: w1
Estimation Technique: OLS

w1                  =   2.916775
                        T-stat. 1.808594

                    +   0.4229623   (y+t-w2)
                        T-stat. 10.32315    ***

                    +   c3          TSLAG(y+t-w2,1)
                        PDL

                    +   0.1020647   time
                        T-stat. 3.048413    **

PDL:
c3 1 2

Distributed Lag Coefficient: c3
Lag     Coeff.          Std. Error      T-stat.
0        0.1292072      0.06348684      2.035181
1        0.01035948     0.04266269      0.2428229
SUM      0.1395667      0.03801893


STATs:
R-Squared                       : 0.9806112
Adjusted R-Squared              : 0.9746454
Durbin-Watson Statistic         : 2.038182
Sum of squares of residuals     : 6.59422
Standard Error of Regression    : 0.7122132
Log of the Likelihood Function : -16.50329
F-statistic                     : 164.3727
F-probability                   : 5.454803e-11
Akaike's IC                     : 45.00658
Schwarz's IC                    : 50.34881
Mean of Dependent Variable      : 36.41667
Number of Observations          : 18
Number of Degrees of Freedom    : 13
Current Sample (year-period)    : 1923-1 / 1940-1


Signif. codes:   *** 0.001  ** 0.01  * 0.05
```

```
...ESTIMATE OK
```

Structural Stability

One of the main purposes of econometric modeling is its use for forecast and policy evaluation and, to this end, the stability of any behavioral equation parameters over time should be verified. In order to check for structural stability two different procedures, which can be derived from the so-called Chow-tests, are applied.

Given a sample of observations (i.e. the base TSRANGE) and selecting an arbitrary forward extension (i.e. the extended TSRANGE) we can perform the same regression by using these two time ranges.

In general, a stability analysis is carried on in the following ways:
- comparing the parameter estimates arising from the two regressions: this is known as the covariance analysis;
- checking the accuracy of the forecast for the dependent variable in the extended TSRANGE, using the estimates produced in the base TSRANGE: this is known as the predictive power test.

The test statistic follows the F distribution and can be performed during the ESTIMATE() function execution by using the CHOWTEST argument set to TRUE (more details in the ESTIMATE help page).

Example:

```
#chow test for the consumption equation
#base TSRANGE set to 1921/1935
R> kleinModelChow <- ESTIMATE(kleinModel
                      ,eqList='cn'
                      ,TSRANGE=c(1921,1,1935,1)
                      ,forceTSRANGE=TRUE
                      ,CHOWTEST=TRUE)

Estimate the Model klein1.txt:
the number of behavioral equations to be estimated is 1.
The total number of coefficients is 4.

----------------------------------------

BEHAVIORAL EQUATION: cn
Estimation Technique: OLS

cn                  =   13.1275
                        T-stat. 6.5046     ***

                    +   0.16698    p
                        T-stat. 2.18304

                    +   0.0885684  TSLAG(p,1)
```

```
                        T-stat. 0.975042

                    +   0.887964   (w1+w2)
                        T-stat. 12.61      ***


STATs:
R-Squared                      : 0.978728
Adjusted R-Squared             : 0.972926
Durbin-Watson Statistic        : 1.38
Sum of squares of residuals    : 6.9186
Standard Error of Regression   : 0.793072
Log of the Likelihood Function : -15.4803
F-statistic                    : 168.7
F-probability                  : 1.77673e-09
Akaike's IC                    : 40.9606
Schwarz's IC                   : 44.5009
Mean of Dependent Variable     : 50.9133
Number of Observations         : 15
Number of Degrees of Freedom   : 11
Current Sample (year-period)   : 1921-1 / 1935-1


Signif. codes:   *** 0.001  ** 0.01  * 0.05



STABILITY ANALYSIS:
Behavioral equation: cn

Chow test:
Sample (auto)      : 1936-1 / 1941-1
F-value            : 4.48873
F-prob(6,17)       : 0.00668723

Predictive Power:

Date, Prd., Actual  , Predict   , Error      , Std. Error, T-stat

1936, 1   ,  57.7   ,  56.5544  ,  1.14564   ,  1.01181  ,  1.13227
1937, 1   ,  58.7   ,  59.931   , -1.23099   ,  1.0201   , -1.20673
1938, 1   ,  57.5   ,  57.9721  , -0.472122  ,  0.968638 , -0.487409
1939, 1   ,  61.6   ,  61.5207  ,  0.0793139 ,  1.20048  ,  0.0660685
1940, 1   ,  65     ,  65.3957  , -0.395718  ,  1.24227  , -0.318545
1941, 1   ,  69.7   ,  73.7965  , -4.09655   ,  1.6693   , -2.45405


...ESTIMATE OK
```

Simulation

The simulation of an econometric model basically consists in solving the system of the equations describing the model for each time period in the specified time interval. Since the equations may not be linear in the variables, and since the graph derived from the incidence matrix may be cyclic, the usual methods based on linear algebra are not applicable. The simulation must be solved by using an iterative algorithm.

BIMETS simulation capabilities support:

- *Static simulations*: a static multiple equation simulation, in which the historical values for the lagged endogenous variables are used in the solutions of subsequent periods;

- *Dynamic simulations*: a dynamic simulation, in which the simulated values for the lagged endogenous variables are used in the solutions of subsequent periods;

- *Forecast simulations*: similar to dynamic simulation, but during the initialization of the iterative algorithm the starting values of endogenous variables in a period are set equal to the simulated values of the previous period. This allows the simulation of future endogenous observations, i.e. the forecast;

- *Residuals check*: a single period, single equation simulation; simulated time series in output are just the computation of the RHS (right-hand-side) of their equation, by using the historical values of the involved time series and by accounting for error autocorrelation and PDLs, if any;

- *Stochastic Simulation*: see `STOCHSIMULATE`;

- *Partial or total exogenization of endogenous variables*: in the provided time interval (i.e. partial exog.) or in whole simulation time range (i.e. total exog.), the values of the selected endogenous variables can be definitely set to their historical values, by excluding their equations from the iterative algorithm of simulation;

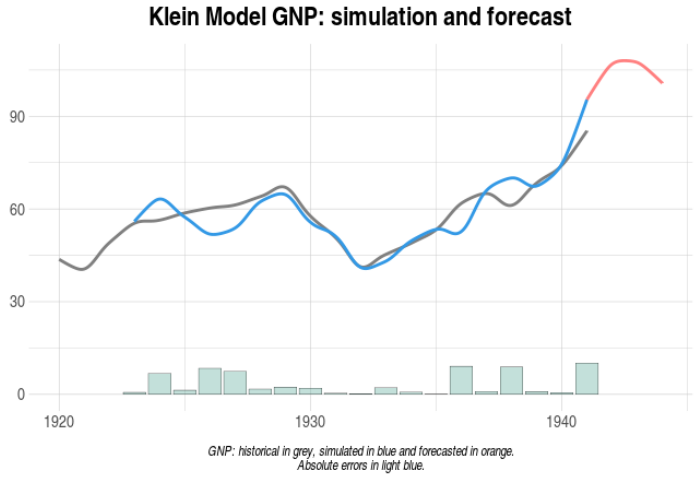
- *Constant adjustment of endogenous variables (add-factors)*: adds up a new exogenous time series (i.e. the "constant adjustment") in the equation of the selected endogenous variables.

- *Gauss-Seidel* and *Newton-Raphson* simulation algorithms: the Gauss-Seidel algorithm is simple, robust, and works well for many backward-looking macro-econometric models. Equations are evaluated as-is in a proper order until the convergence, if any, is verified on feedback variables. It is slower than Newton algorithms for a very low convergence criterion, and fails to find a convergence for a small set of econometric models, even when a convergence exists. The Newton-Raphson algorithm allows users to solve a broader set of macro-econometric models than the Gauss-Seidel algorithm. Moreover, it is usually faster than the Gauss-Seidel algorithm (on modern computers, users must simulate an extensive econometric model with a low convergence criterion to appreciate the speedup). This type of algorithm requires the construction and the inversion of the Jacobian matrix for the feedback variables; thus, in some scenarios, numerical issues can arise, and users are required to manually exclude some feedback variables from the Jacobian matrix by using the `JacobianDrop` argument of the `SIMULATE` procedure.

More details are available in the SIMULATE help page.

Back to Kelin's model example, let's forecast the *GNP* (i.e. the "y" endogenous variable, originally referred as "Net national income, measured in billions of 1934 dollars", pag. 141 in "Economic Fluctuations in the United States" by L. R. Klein, Wiley and Sons Inc., New York, 1950) up to 1943:

```
R> #FORECAST GNP in 1942:1944

R> #we need to extend exogenous variables in 1942 up to 1944
R> kleinModel$modelData <- within(kleinModel$modelData,{
            w2   = TSEXTEND(w2,  UPTO=c(1944,1),EXTMODE='CONSTANT')
            t    = TSEXTEND(t,   UPTO=c(1944,1),EXTMODE='CONSTANT')
            g    = TSEXTEND(g,   UPTO=c(1944,1),EXTMODE='CONSTANT')
            time = TSEXTEND(time,UPTO=c(1944,1),EXTMODE='LINEAR')
            })


R> #simulate model
R> kleinModel <- SIMULATE(kleinModel
            ,simType='FORECAST'
            ,TSRANGE=c(1941,1,1944,1)
            ,simConvergence=0.00001
            ,simIterLimit=100
            )

Simulation: 100.00 %
...SIMULATE OK

R> #get forecasted GNP
R> TABIT(kleinModel$simulation$y)

Date, Prd., kleinModel$simulation$y
1941, 1   ,  95.41613
1942, 1   ,  106.8923
1943, 1   ,  107.4302
1944, 1   ,  100.7512
```



Below is an example of advanced simulation using the NEWTON algorithm:

```
R> #DYNAMIC NEWTON SIMULATION EXAMPLE
R> #WITH EXOGENIZATION AND CONSTANT ADJUSTMENTS

R> #define exogenization list
R> #'cn' exogenized in 1923-1925
R> #'i' exogenized in whole TSRANGE
R> exogenizeList <- list(
                  cn = c(1923,1,1925,1),
                  i  = TRUE
                  )

R> #define add-factor list
R> #cn add-factor will be 1 in 1923 and -1 in 1924
R> #y add-factor will be 0.1 in 1926, -0.1 in 1927 and -0.5 in 1928
R> constantAdjList <- list(
                 cn = TIMESERIES(1,-1,START=c(1923,1),FREQ='A'),
                 y  = TIMESERIES(0.1,-0.1,-0.5,START=c(1926,1),FREQ='A')
                 )

R> #simulate model
R> kleinModel <- SIMULATE(kleinModel
                  ,simAlgo='NEWTON'
                  ,simType='DYNAMIC'
                  ,TSRANGE=c(1923,1,1941,1)
                  ,simConvergence=0.00001
                  ,simIterLimit=100
```

```
,Exogenize=exogenizeList
,ConstantAdjustment=constantAdjList
)
```

Stochastic Simulation

Forecasts produced by structural econometric models are subject to several sources of error, such as random disturbance term of each stochastic equation, errors in estimated coefficients, errors in forecasts of exogenous variables, errors in preliminary data and mis-specification of the model.

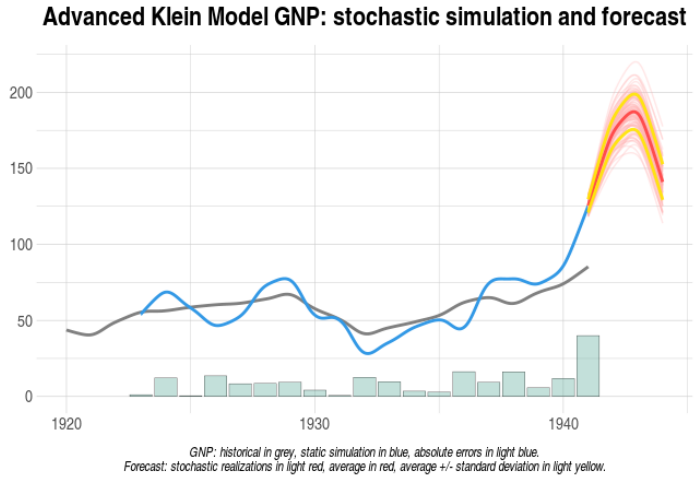
The forecast error depending on the structural disturbances can be analyzed by using the stochastic simulation procedure.

The deterministic simulation is the simultaneous solution of an econometric model obtained by applying, for each stochastic (behavioral) equation, the expected values of the structural disturbances, which are all zero by assumption. In the BIMETS `STOCHSIMULATE` stochastic simulation, the structural disturbances are given values that have specified stochastic properties. The error terms of the estimated behavioral equation of the model are appropriately perturbed. Identity equations and exogenous variables can be as well perturbed by disturbances that have specified stochastic properties. The model is then solved for each data set with different values of the disturbances. Finally, mean and standard deviation are computed for each simulated endogenous variable.

In terms of computational efficiency, the procedure takes advantage of the fact that multiple datasets are bound together in matrices. Therefore, to achieve a global convergence, the iterative simulation algorithm is executed once for all perturbed datasets. This solution can be viewed as a sort of a SIMD (i.e. Single Instruction Multiple Data) parallel simulation: `STOCHSIMULATE` function transforms time series into matrices; consequently, the procedure can easily bind multiple datasets by column. At the same time, a single run ensures a fast code execution. Finally, each column in the output matrices represents a stochastic realization.

By using the `StochStructure` argument of this function, users can define a stochastic structure for the disturbances. For each variable of the model, users can provide a distinct probability distribution for the disturbance, and a specific time range of application. Mean and standard deviation for each simulated endogenous time series will be stored in the `stochastic_simulation` element of the output model object; all the stochastic realizations will be stored in the `simulation_MM` element of the output model object as named matrices.

In the following figure, the advanced Klein model (see code example), has been perturbed during the forecast operation by applying a normal disturbance to the endogenous *Consumption* behavioral `cn` add-factor in year 1942, and a uniform disturbance to the exogenous *Government Expenditure* time series `g` along all the simulation `TSRANGE`. The normal disturbance applied to the `cn` behavioral has a zero mean and a standard deviation equal to the behavioral regression standard error, i.e. `advancedKleinModel$behaviorals$cn$statistics$StandardErrorRegression`, thus roughly replicating the `ESTIMATE` regression error during the current perturbation (not accounting for inter-equations cross-covariance).




```
R> #we want to perform a stochastic forecast of the GNP up to 1944
R> #we will add normal disturbances to endogenous Consumption 'cn'
R> #in 1942 by using its regression standard error
R> #we will add uniform disturbances to exogenous Government Expenditure 'g'
R> #in whole TSRANGE
R> nSD=advancedKleinModel$behaviorals$cn$statistics$StandardErrorRegression
R> myStochStructure <- list(
  cn=list(
        TSRANGE=c(1942,1,1942,1),
        TYPE='NORM',
        PARS=c(0,nSD)
        ),
  g=list(
        TSRANGE=TRUE,
        TYPE='UNIF',
        PARS=c(-1,1)
        )
  )

R> #we need to extend exogenous variables up to 1944
R> advancedKleinModel$modelData <- within(advancedKleinModel$modelData,{
    w2    = TSEXTEND(w2,  UPTO=c(1944,1),EXTMODE='CONSTANT')
    t     = TSEXTEND(t,   UPTO=c(1944,1),EXTMODE='LINEAR')
    g     = TSEXTEND(g,   UPTO=c(1944,1),EXTMODE='CONSTANT')
    k     = TSEXTEND(k,   UPTO=c(1944,1),EXTMODE='LINEAR')
    time  = TSEXTEND(time,UPTO=c(1944,1),EXTMODE='LINEAR')
  })
```

```
R> #stochastic model forecast
R> advancedKleinModel <- STOCHSIMULATE(advancedKleinModel
                      ,simType='FORECAST'
                      ,TSRANGE=c(1941,1,1944,1)
                      ,StochStructure=myStochStructure
                      ,StochSeed=123
                      )

R> #print mean and standard deviation of forecasted GNP
R> with(advancedKleinModel$stochastic_simulation,TABIT(y$mean, y$sd))

      Date, Prd., y$mean         , y$sd

      1941, 1   ,  125.5045      ,  4.250935
      1942, 1   ,  173.2946      ,  9.2632
      1943, 1   ,  185.9602      ,  11.87774
      1944, 1   ,  141.0807      ,  11.6973

R> #print the unperturbed forecasted GNP along with the
R> #first 5 perturbed realizations
R> with(advancedKleinModel$simulation_MM,print(y[,1:6]))
```

Multipliers Analysis

The BIMETS MULTMATRIX function computes the matrix of both impact and interim multipliers, for a selected set of endogenous variables (i.e. TARGET) with respect to a selected set of exogenous variables (i.e. INSTRUMENT), by subtracting the results from different simulations in each period of the provided time range (i.e. TSRANGE). The simulation algorithms are the same as those used for the SIMULATE operation.

The MULTMATRIX procedure is articulated as follows:

1. simultaneous simulations are done;

2. the first simulation establishes the base line solution (without shocks);

3. the other simulations are done with shocks applied to each of the INSTRUMENT one at a time for every period in TSRANGE;

4. each simulation follows the defaults described in the SIMULATE help page, but has to be STATIC for the IMPACT multipliers and DYNAMIC for INTERIM multipliers;

5. given MM_SHOCK shock amount as a very small positive number, derivatives are computed by subtracting the base line solution of the TARGET from the shocked solution, then dividing by the value of the base line INSTRUMENT times the MM_SHOCK;

BIMETS users can also declare an endogenous variable as the INSTRUMENT variable. In this case, the constant adjustment (see SIMULATE) related to the provided endogenous variable will be used as the INSTRUMENT exogenous variable.

Back to our Klein's model example, we can calculate impact multipliers of *Government Expenditure* "g" and *Government Wage Bill* "w2" with respect of *Consumption* "cn" and *Gross National Product* "y" in the year 1941 by using the previously estimated model:

```
R> kleinModel <- MULTMATRIX(kleinModel,
                        symType='STATIC',
                        TSRANGE=c(1941,1,1941,1),
                        INSTRUMENT=c('w2','g'),
                        TARGET=c('cn','y')
                        )
Multiplier Matrix: 100.00 %
...MULTMATRIX OK

R> kleinModel$MultiplierMatrix

           w2_1        g_1
cn_1  0.4540346   1.671956
y_1   0.2532000   3.653260
```

Results show that the impact multiplier of "y" with respect to "g" is +3.65. If we change the *Government Expenditure* "g" value in 1941 from 22.3 (his historical value) to 23.3 (+1), then the simulated *Gross National Product* "y" in 1941 changes from 95.2 to 99, thusly roughly confirming the +3.65 impact multiplier. Note that "g" only appears once in the model definition, and only in the "y" equation, with a coefficient equal to one (Keynes would approve).

An interim-multiplier example follows:

```
R> #multi-period interim multipliers
R> kleinModel <- MULTMATRIX(kleinModel,
                    TSRANGE=c(1940,1,1941,1),
                    INSTRUMENT=c('w2','g'),
                    TARGET=c('cn','y'))
Multiplier Matrix: 100.00 %
...MULTMATRIX OK

R> #output multipliers matrix (note the zeros where the period
R> #of the INSTRUMENT is greater than the period of the TARGET)
R> kleinModel$MultiplierMatrix

            w2_1       g_1       w2_2      g_2
cn_1   0.4478202  1.582292  0.0000000 0.000000
y_1    0.2433382  3.510971  0.0000000 0.000000
cn_2  -0.3911001  1.785042  0.4540346 1.671956
```

```
y_2   -0.6251177  2.843960  0.2532000 3.653260
```

Endogenous Targeting

The endogenous targeting of econometric models consists of solving the model while interchanging the role of one or more endogenous variables with an equal number of exogenous variables.

The BIMETS RENORM procedure determines the values for the INSTRUMENT exogenous variables that allow the objective TARGET endogenous values to be achieved, with respect to the constraints given by the model equations (see MDL).

This is an approach to economic and monetary policy analysis, and is based on two assumptions:

1. there exists a desired level for a set of the n endogenous variables defined as TARGET;
2. there exists a set of the n exogenous variables defined as INSTRUMENT;

Given these premises, the endogenous targeting process consists in determining the values of the exogenous variables chosen as INSTRUMENT allowing us to achieve the desired values for the endogenous variables designated as TARGET. In other words the procedure allows users to exchange the role of exogenous and endogenous among a set of variables pairs.

Given a list of exogenous INSTRUMENT variables and a list of TARGET endogenous time series, the iterative procedure can be split into the following steps:

1. Computation of the multipliers matrix MULTMAT of the TARGET endogenous variables with respect to the INSTRUMENT exogenous variables (this is a square matrix by construction);

2. Solution of the linear system (if any):
$V_{exog}(i+1) = V_{exog}(i) + \texttt{MULTMAT}^{-1} * (V_{endog}(i) - \texttt{TARGET})$, where $V_{exog}(i)$ are the exogenous variables in the INSTRUMENT list and $V_{endog}(i)$ are the endogenous variables that have a related target in the TARGET list, given i the current iteration;

3. Simulation of the model with the new set of exogenous variables computed in step 2, then a convergence check by comparing the subset of endogenous variables arising from this simulation and the related time series in TARGET list. If the convergence condition is satisfied, or the maximum number of iterations is reached, the algorithm will stop, otherwise it will go back to step 1;

Users can also declare an endogenous variable as an INSTRUMENT variable. In this case, the constant adjustment (see SIMULATE) related to the provided endogenous variable will be used as the instrument exogenous variable. This procedure is particularly suited for the automatic computation of the add-factors needed to fine tune the model into a baseline path and to improve the forecasting accuracy.

If the convergence condition is satisfied, the RENORM procedure will return the INSTRUMENT time series allowing us to achieve the desired values for endogenous variables designated as TARGET.

Back to our Klein's model example, we can perform the endogenous targeting of the previously estimated model. First of all, the targets must be defined:

```
R> #we want an arbitrary value on Consumption of 66 in 1940 and 78 in 1941
R> #we want an arbitrary value on GNP of 77 in 1940 and 98 in 1941

R> kleinTargets  <-  list(
              cn = TIMESERIES(66,78,START=c(1940,1),FREQ=1),
              y  = TIMESERIES(77,98,START=c(1940,1),FREQ=1)
              )
```

Then, we can perform the model endogenous targeting by using the "w2" (*Wage Bill of the Government Sector*) and the "g" (*Government Expenditure*) exogenous variables as INSTRUMENT, in the years 1940 and 1941:

```
R> kleinModel <- RENORM(kleinModel
                   ,INSTRUMENT = c('w2','g')
                   ,TARGET = kleinTargets
                   ,TSRANGE = c(1940,1,1941,1)
                   ,simIterLimit = 100
                    )
```

Once RENORM completes, the calculated values of exogenous INSTRUMENT allowing us to achieve the desired endogenous TARGET values are stored in the model:

```
R> with(kleinModel,TABIT(modelData$w2,
                      renorm$INSTRUMENT$w2,
                      modelData$g,
                      renorm$INSTRUMENT$g,
                      TSRANGE=c(1940,1,1941,1)
                      )
          )

Date, Prd., modelData$w2, renorm$INSTRUMENT$w2, modelData$g, renorm$INSTRUMENT$g
1940,  1  , 8           , 7.413331            , 15.4       , 16.1069
1941,  1  , 8.5         , 9.3436              , 22.3       , 22.65985
```

So, if we want to achieve on "cn" (*Consumption*) an arbitrary simulated value of 66 in 1940 and 78 in 1941, and if we want to achieve on "y" (*GNP*) an arbitrary simulated value of 77 in 1940 and 98 in 1941, we need to change exogenous "w2" (*Wage Bill of the Government Sector*) from 8 to 7.41 in 1940 and from 8.5 to 9.34 in 1941, and we need to change exogenous "g" (*Government Expenditure*) from 15.4 to 16.1 in 1940 and from 22.3 to 22.66 in 1941.

Let's verify:

```
R> #create a new model
R> kleinRenorm <- kleinModel

R> #get instruments to be used
```

```
R> newInstruments <- kleinModel$renorm$INSTRUMENT

R> #change exogenous by using new instruments data
R> kleinRenorm$modelData <- within(kleinRenorm$modelData,
                 {
                   w2[[1940,1]]=newInstruments$w2[[1940,1]]
                   w2[[1941,1]]=newInstruments$w2[[1941,1]]
                   g[[1940,1]] =newInstruments$g[[1940,1]]
                   g[[1941,1]] =newInstruments$g[[1941,1]]
                 }
                )
R> #users can also replace last two commands with:
R> #kleinRenorm$modelData <- kleinRenorm$renorm$modelData

R> #simulate the new model
R> kleinRenorm <- SIMULATE(kleinRenorm
                      ,TSRANGE=c(1940,1,1941,1)
                      ,simConvergence=0.00001
                      ,simIterLimit=100
                      )
Simulation: 100.00 %
...SIMULATE OK

R> #verify targets are achieved
R> with(kleinRenorm$simulation,
     TABIT(cn,y)
     )

Date, Prd., cn        , y
1940,  1  , 66.01116  , 77.01772
1941,  1  , 78.02538  , 98.04121
```

Optimal Control

An approach to policy evaluation is via a so-called "social welfare function". This approach relaxes the assumptions of the instruments-targets framework, i.e. the RENORM procedure. Rather than assuming specific desired targets for some endogenous variables, it assumes the existence of a social welfare function determining a scalar measure of performance based on both endogenous and policy (exogenous) variables.

The social welfare function can incorporate information about tradeoffs in objectives that are not allowed by the RENORM instruments-targets approach.

BIMETS supplies the OPTIMIZE procedure in order to perform optimal control exercises on econometric models.

The optimization consists of maximizing a social welfare function, i.e. the objective-function, depending on exogenous and (simulated) endogenous variables, subject to user constraints plus the constraints imposed by the econometric model equations. Users are allowed to define constraints and objective-functions of any degree, and are allowed to provide different constraints and objective-functions in different optimization time periods.

The core of the OPTIMIZE procedure is based on a Monte Carlo method that takes advantage of the STOCHSIMULATE procedure. Policy variables, i.e. INSTRUMENT, are uniformly perturbed in the range defined by the user-provided boundaries, then the INSTRUMENT values that i) verify the user-provided constraints and ii) maximize the objective-functions are selected and stored into the optimize element of the output BIMETS model.

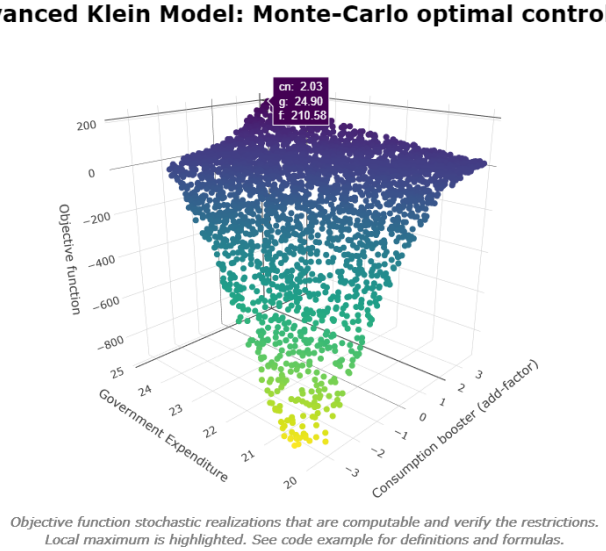
In the following figure, the scatter plot is populated with 2916 objective function stochastic realizations, computed by using the example code at the end of this section; the 210.58 local maximum is highlighted
(i.e. advancedKleinModel$optimize$optFunMax in code example).

In this example:

i) The objective function definition is:
$f(y, cn, g) = (y - 110) + (cn - 90) * |cn - 90| - \sqrt{g - 20}$
given y as the simulated *Gross National Product*, cn as the simulated *Consumption* and g as the exogenous *Government Expenditure*: the basic idea is to maximize *Consumption*, and secondarily the *Gross National Product*, while reducing the *Government Expenditure*;

ii) The INSTRUMENT variables are the cn *Consumption* "booster" (i.e. the add-factor, not to be confused with the simulated *Consumption* in the objective function) and the g *Government Expenditure*, defined over the following domains: $cn \in (-5, 5)$, $g \in (15, 25)$;

iii) The following restrictions are applied to the INSTRUMENT: $g + cn^2/2 < 27 \wedge g + cn > 17$, given cn as the *Consumption* "booster" (i.e. the add-factor) and g as the *Government Expenditure*;



Objective function stochastic realizations that are computable and verify the restrictions. Local maximum is highlighted. See code example for definitions and formulas.

The figure clearly shows that non-linear restrictions have been applied, and that non-computable objective functions have been discarded, e.g. the stochastic realizations having $g < 20$ due to the square root operation in the objective function, given instrument $g \in (15, 25)$.

An example of an optimal control exercise applied to the previoously defined `advancedKleinModel` follows:

```
R> #reset time series of the model object that have been
R> #modified in the stochastic simulation section
R> advancedKleinModel <- LOAD_MODEL_DATA(advancedKleinModel,kleinModelData)

R> #we want to maximize the non-linear objective function:
R> #f()=(y-110)+(cn-90)*ABS(cn-90)-(g-20)^0.5
R> #in 1942 by using INSTRUMENT cn in range (-5,5)
R> #(cn is endogenous so we use the add-factor)
R> #and g in range (15,25)
R> #we will also impose the following non-linear restriction:
R> #g+(cn^2)/2<27 & g+cn>17

R> #we need to extend exogenous variables up to 1942
R> advancedKleinModel$modelData <- within(advancedKleinModel$modelData,{
    w2    = TSEXTEND(w2,  UPTO=c(1942,1),EXTMODE='CONSTANT')
    t     = TSEXTEND(t,   UPTO=c(1942,1),EXTMODE='LINEAR')
    g     = TSEXTEND(g,   UPTO=c(1942,1),EXTMODE='CONSTANT')
    k     = TSEXTEND(k,   UPTO=c(1942,1),EXTMODE='LINEAR')
    time  = TSEXTEND(time,UPTO=c(1942,1),EXTMODE='LINEAR')
})

R> #define INSTRUMENT and boundaries
```

```
R> myOptimizeBounds <- list(
    cn=list(TSRANGE=TRUE,
            BOUNDS=c(-5,5)),
    g=list(TSRANGE=TRUE,
           BOUNDS=c(15,25))
)

R> #define restrictions
R> myOptimizeRestrictions <- list(
    myRes1=list(
        TSRANGE=TRUE,
        INEQUALITY='g+(cn^2)/2<27 & g+cn>17')
)

R> #define objective function
R> myOptimizeFunctions <- list(
    myFun1=list(
        TSRANGE=TRUE,
        FUNCTION='(y-110)+(cn-90)*ABS(cn-90)-(g-20)^0.5')
)

R> #Monte-Carlo optimization by using 10000 stochastic realizations
R> #and 1E-4 convergence criterion
R> advancedKleinModel <- OPTIMIZE(advancedKleinModel
                          ,simType = 'FORECAST'
                          ,TSRANGE=c(1942,1,1942,1)
                          ,simConvergence= 1E-4
                          ,simIterLimit  = 1000
                          ,StochReplica  = 10000
                          ,StochSeed = 123
                          ,OptimizeBounds = myOptimizeBounds
                          ,OptimizeRestrictions = myOptimizeRestrictions
                          ,OptimizeFunctions = myOptimizeFunctions)
OPTIMIZE(): optimization boundaries for the add-factor of endogenous
            variable "cn" are (-5,5) from year-period 1942-1 to 1942-1.
OPTIMIZE(): optimization boundaries for the exogenous
            variable "g" are (15,25) from year-period 1942-1 to 1942-1.
OPTIMIZE(): optimization restriction "myRes1" is active
            from year-period 1942-1 to 1942-1.
OPTIMIZE(): optimization objective function "myFun1" is active
            from year-period 1942-1 to 1942-1.

Optimize:     100.00 %
OPTIMIZE(): 2916 out of 10000 objective function realizations (29%)
            are finite and verify the provided restrictions.
...OPTIMIZE OK

R> #print local maximum
```

```
R> advancedKleinModel$optimize$optFunMax
[1] 210.5755

R> #print INSTRUMENT that allow local maximum to be achieved
R> advancedKleinModel$optimize$INSTRUMENT
$cn
Time Series:
Start = 1942
End = 1942
Frequency = 1
[1] 2.032203

$g
Time Series:
Start = 1942
End = 1942
Frequency = 1
[1] 24.89773

R> #LET'S VERIFY RESULTS
R> #copy into modelData the computed INSTRUMENT
R> #that allow to maximize the objective function
R> advancedKleinModel$modelData <- advancedKleinModel$optimize$modelData

R> #simulate the model by using the new INSTRUMENT
R> #note: we used cn add-factor as OPTIMIZE instrument, so we need
R> #to pass the computed cn add-factor to the SIMULATE call
R> newConstantAdjustment <- advancedKleinModel$optimize$ConstantAdjustment
R> advancedKleinModel <- SIMULATE(advancedKleinModel
                  ,simType = 'FORECAST'
                  ,TSRANGE = c(1942,1,1942,1)
                  ,simConvergence = 1E-5
                  ,simIterLimit = 1000
                  ,ConstantAdjustment = newConstantAdjustment)

R> #calculate objective function by using the SIMULATE output time series
R> #(y-110)+(cn-90)*ABS(cn-90)-(g-20)^0.5
R> y  <- advancedKleinModel$simulation$y
R> cn <- advancedKleinModel$simulation$cn
R> g  <- advancedKleinModel$modelData$g
R> optFunTest <- (y-110)+(cn-90)*abs(cn-90)-(g-20)^0.5

R> #verify computed max is equal to optimization max
R> #(in the following command TSPROJECT could be omitted because
R> #myFun1$TSRANGE = TRUE)
R> abs(sum(TSPROJECT(optFunTest
              ,TSRANGE=c(1942,1,1942,1)
              ,ARRAY = TRUE)
```

```
        ) - advancedKleinModel$optimize$optFunMax)  < 1E-4
[1] TRUE
```

Details

Package:	bimets - Time Series And Econometric Modeling In R
Type:	Package
License:	GPL-3

BIMETS estimation and simulation results have been compared to the output results of leading commercial econometric software by using several large and complex models.

The models used in the comparison have more than:

+100 behavioral equations;
+700 technical identities;
+500 coefficients;
+1000 time series of endogenous and exogenous variables;

In these models, there are equations with restricted coefficients, polynomial distributed lags, error autocorrelation, and conditional evaluation of technical identities; all models have been simulated in *static*, *dynamic*, and *forecast* mode, with exogenization and constant adjustments of endogenous variables through the use of BIMETS capabilities.

In the +800 endogenous simulated time series over the +20 simulated periods (i.e. more than 16.000 simulated observations), the average *percentage* difference between BIMETS and leading commercial software results has a magnitude of $10^{-7}\%$. The difference between results calculated by using different commercial software has the same average magnitude.

BIMETS stands for Bank of Italy Model Easy Time Series; it does not depend on compilers or third-party software so it can be freely downloaded and installed on Linux, MS Windows(R) and Mac OSX(R), without any further requirements.

More details in:

- MDL
- LOAD_MODEL
- ESTIMATE
- SIMULATE
- STOCHSIMULATE
- MULTMATRIX
- RENORM
- OPTIMIZE

Disclaimer: *The views and opinions expressed in these pages are those of the authors and do not necessarily reflect the official policy or position of the Bank of Italy. Examples of analysis per-*

formed within these pages are only examples. They should not be utilized in real-world analytic products as they are based only on very limited and dated open source information. Assumptions made within the analysis are not reflective of the position of the Bank of Italy.

Author(s)

Andrea Luciani <andrea.luciani@bancaditalia.it>
Roberto Stok <roberto.stok@bancaditalia.it>

A1D *A1D*

Description

This function returns the array built with input argument values. Input can be time series, numerical arrays, or strings.

Usage

```
A1D(..., length = NULL, avoidCompliance = FALSE)
```

Arguments

`...`	Input argument list. This function accepts strings, time series, objects of class `numeric` or `logical`. Input time series must satisfy the compliance control check defined in `is.bimets`
`length`	Length of output array, that must be greater than the sum of each input argument size: if the length of the output array is provided, than the output array will be eventually filled with zeros.
`avoidCompliance`	If `TRUE`, compliance control check of input time series will be skipped. See `is.bimets`

Value

This function returns an array of the same class of the input.

See Also

`NOELS`
`is.bimets`
`BIMETS indexing`
`TIMESERIES`
`TSDATES`

LOCS
NAMELIST
INTS
TSINFO
TSLOOK
TABIT
ELIMELS
INDEXNUM

Examples

```
n<-10;
#create ts
ts1<-TSERIES(rnorm(n),START=c(2000,1),FREQ=1)

#create A1D() array with scalars, ts, and NA
out_a1d<-A1D(length=25, ts1, 1, -8.9, NA)
print(out_a1d)

#same example no length specified
out_a1d<-A1D(ts1, 1, -8.9, NA)
print(out_a1d)

#strings example
out_a1d<-A1D(length=5,'aa','bb','ccc')
print(out_a1d)
```

ANNUAL *Annual Time Series (Dis)Aggregation*

Description

This function returns a yearly aggregated time series, by using as input a semiannual, quarterly, monthly or daily time series.

Usage

```
ANNUAL(x = NULL, fun = NULL, avoidCompliance = FALSE, ...)
YEARLY(x = NULL, fun = NULL, avoidCompliance = FALSE, ...)
```

Arguments

x	Input time series, that must satisfy the compliance control check defined in is.bimets.

`fun` **STOCK**: the output value of a year is equal to the value of the input time series in the last period of the same year
NSTOCK: the output value of a year is equal to the value of the input time series in the last non-missing NA period of the same year
SUM: the output value of a year is equal to the sum of all the observations of the input time series in the same year
NSUM: the output value of a year is equal to the sum of all the non-missing NA observations of the input time series in the same year
AVE: the output value of a year is equal to the average of all the observations of the input time series in the same year
NAVE: the output value of a year is equal to the average of all the non-missing NA observations of the input time series in the same year

`avoidCompliance` If TRUE, compliance control check of input time series will be skipped. See `is.bimets`

`...` Backward compatibility.

Value

This function returns a yearly BIMETS time series.

See Also

`SEMIANNUAL`
`QUARTERLY`
`MONTHLY`
`DAILY`

Examples

```
#TS DAILY TO ANNUAL
n<-366
ts1<-TIMESERIES(0:n,START=c(2000,1),FREQ='D')
ts1[10]<-NA
TABIT(ANNUAL(ts1,fun='NAVE'))

#TS DAILY TO ANNUAL
n<-36
ts1<-TIMESERIES(0:n,START=c(2000,1),FREQ='M')
ts1[10]<-NA
TABIT(YEARLY(ts1,fun='SUM'))
```

as.bimets *Convert a Time Series to BIMETS*

Description

This function tries to convert a time series of class ts() or xts() into a BIMETS time series that satisfy the compliance control check defined in is.bimets

All the information in the input time series will be preserved.

Input time series must be of class ts() or xts(), and will be converted in the BIMETS class-type specified in the global option BIMETS_CONF_CCT (see BIMETS configuration).

If the input time series has a temporal discontinuity (i.e. a missing pair Date-Value in the case of xts() time series) then the missing pair Date-Value is inserted in the output time series with a missing value NA, or with the value provided in the FILLVALUE argument.

If BIMETS_CONF_CCT='XTS', in the case of a monthly input time series the .indexCLASS is converted to the class yearmon(); in the case of a quarterly input time series the .indexCLASS is converted to yearqtr(); in the case of other input frequency the .indexCLASS is converted to Date(). If BIMETS_CONF_CCT='XTS' the dates of all the output observations are set accordingly to the BIMETS global option BIMETS_CONF_DIP, i.e. the first or the last dates in the period (see BIMETS configuration).

If the input time series has multiple observations in the same date, e.g. an xts() with a two or more observations in the same date, the duplication is removed and the output time series will contain only the later observation (see example).

If the input time series is multivariate, the output time series will contain only the first column of the input matrix of data (where the matrix of data is the matrix built by binding input time series values as columns).

If the input time series is a daily time series of class xts() and the global option BIMETS_CONF_CCT='TS' then the 366th period of the output time series in each non-bissextile year will have the value of the 365th period in the same year (duplicated value).

Usage

```
as.bimets(x=NULL, FILLVALUE=NA, VERBOSE=FALSE, ...)
```

Arguments

x	Input time series of class ts() or xts().
FILLVALUE	Value inserted in the output time series in the case of temporal discontinuity. Defaults to missing NA.
VERBOSE	If TRUE, a verbose description of inserted and/or removed observations, if any, will be shown.

... Backward compatibility.

Value

This function returns a BIMETS time series (see also BIMETS configuration) .

See Also

is.bimets
TIMESERIES
BIMETS indexing
BIMETS configuration
fromBIMETStoXTS
fromBIMETStoTS

Examples

```
#xts series with dates equal to the first date in the period,
#and some missing observations
#first...set option and work with xts
setBIMETSconf('BIMETS_CONF_CCT','XTS')
#create xts
xt<-xts(1:10,order.by=seq(as.Date('2000-01-01'),len=10,by='year'))
#remove some data
xt<-xt[-5]
xt<-xt[-3]
#convert to bimets
xtBimets<-as.bimets(xt)
#print before and after...
print(xt)
print(xtBimets)

#ts bivariate series into xts
setBIMETSconf('BIMETS_CONF_CCT','XTS')
ts<-ts(matrix(c(1,2,3,4,5,6),nrow=3,ncol=2),start=c(2000,1),frequency=1)
print(ts)
xtsBimets<-as.bimets(ts)
print(xtsBimets)

#reset defaults
setBIMETSconf('BIMETS_CONF_DIP','LAST')
setBIMETSconf('BIMETS_CONF_CCT','TS')

#xts quarterly with irregular dates and missings data
xt<-xts(1:10,order.by=seq(as.Date('2000-01-03'),len=10,by='3 months'))
#remove some data
xt<-xt[-5]
xt<-xt[-3]
#convert
tsBimets<-as.bimets(xt)
```

```
#print before and after
print(xt)
print(tsBimets)

#xts daily with duplicated observations and missing data
xt<-xts(1:11,order.by=c(as.Date('2000-01-01'),
                    seq(as.Date('2000-01-01'),
                    len=10,by='day')))
xt<-xt[-5]
xt<-xt[-3]
#convert
tsBimets<-as.bimets(xt)
#print before and after
print(xt)
print(tsBimets)

#verbose and fillvalue
xt<-xts(1:11,order.by=c(as.Date('2000-01-01'),
                      seq(as.Date('2000-01-01'),
                      len=10,by='day')))
xt<-xt[-5]
xt<-xt[-3]
as.bimets(xt,FILLVALUE=99.99,VERBOSE=TRUE)
```

`bimetsConf` *BIMETS Global Options Configuration*

Description

BIMETS package depends on some options in order to transform and to present time series data. These options can be read or changed by using the functions:
`setBIMETSconf(opt,value)` and `getBIMETSconf(opt)`

Usage

```
setBIMETSconf(opt=NULL, value=NULL, suppressOutput=FALSE)
getBIMETSconf(opt=NULL)
```

Arguments

`opt` Name of the BIMETS option. Available option names are:

BIMETS_CONF_DIP: Date In Period. Users can associate to each observation in a time series the first or the last date in the period, i.e. 1 January or 31 December in the case of a yearly time series, 1 January/1 July or 30 June/31 December in the case of a semiannual time series, 1 January/31 January in the case of a monthly time series in January, etc. The assignments by date to time series (e.g. `ts['2000-01-01']=value`) need to be coherent to the value of this

global option. Accepted values are:
LAST: (default) each observation has associated the last date of the period, e.g. 31 Mar for a quarterly time series, 31 January for a monthly time series in the first period, etc.
FIRST: each observation has associated the first date of the period, e.g. 1 Jan for a quarterly time series, 1 Feb for a monthly time series in the second period, etc.

BIMETS_CONF_CCT: Constructor Class Type. The package supports `ts()` and `xts()` time series as input arguments. Users can select the base class of a BIMETS time series, i.e. the class used when a time series is created with `TIMESERIES()` or converted to a BIMETS time series using `as.bimets()`. This is a global option; users can locally override the selection of the output class using the `class='TS'` or `class='XTS'` argument in the `TIMESERIES()` function. The option `BIMETS_CONF_CCT` can be assigned to the following values:
TS: (default) the time series constructor `TIMESERIES()` and the conversion function `as.bimets()` return an object of class `ts()`
XTS: the time series constructor `TIMESERIES()` and the conversion function `as.bimets()` return an object of class `xts()`.

Please note that BIMETS package is faster using BIMETS_CONF_CCT='TS'

BIMETS_CONF_NOC: NO Compliance test. If this option is set `TRUE` then the compliance control check on input time series, i.e. `is.bimets()`, will be globally disabled. The default is set to `FALSE`. The compliance check on input time series should generally be active, otherwise a malformed input time series can produce unwanted results in operations.

`value` The value to be assigned to the BIMETS option.

`suppressOutput` If `TRUE`, the output messages will be disabled.

Value

This function set or read global BIMETS options, and return a NULL value.

See Also

`TIMESERIES`
`is.bimets`
`as.bimets`
`fromBIMETStoXTS`
`fromBIMETStoTS`
`BIMETS indexing`

Examples

```
#default BIMETS_CONF_DIP is LAST
```

```
#create ts
ts1<-TSERIES((1:10),START=c(2000,1),FREQ=1)

#transform to xts and print
xt1<-fromBIMETStoXTS(ts1)
print(xt1)  #....dates as of 31 Dec

#set configuration BIMETS_CONF_DIP to FIRST
setBIMETSconf('BIMETS_CONF_DIP','FIRST')

#create ts
ts1<-TSERIES((1:10),START=c(2000,1),FREQ=1)

#transform to xts and print
xt1<-fromBIMETStoXTS(ts1)
print(xt1)  #....dates as of 1 Jan

#set configuration BIMETS_CONF_DIP to LAST
setBIMETSconf('BIMETS_CONF_DIP','LAST')

#default to XTS
setBIMETSconf('BIMETS_CONF_CCT','XTS')

#check compliance of xt1 and fail...
is.bimets(xt1) #... FALSE

#set configuration BIMETS_CONF_DIP to FIRST
setBIMETSconf('BIMETS_CONF_DIP','FIRST')

#check compliance of xt1 and ok...
is.bimets(xt1) #... TRUE

print(getBIMETSconf('BIMETS_CONF_DIP')) # ... returns FIRST
print(getBIMETSconf('BIMETS_CONF_CCT')) # ... returns XTS

print(is.xts(TSERIES(1:10,START=c(2000,1),FREQ=1))) #...print TRUE
print(is.ts(TSERIES(1:10,START=c(2000,1),FREQ=1,class='TS'))) #...print TRUE



#NOC
setBIMETSconf('BIMETS_CONF_CCT','XTS')
is.bimets(xts()) #FALSE
setBIMETSconf('BIMETS_CONF_NOC',TRUE)
is.bimets(xts()) #TRUE


#...back to defaults
setBIMETSconf('BIMETS_CONF_DIP','LAST')
setBIMETSconf('BIMETS_CONF_CCT','TS')
setBIMETSconf('BIMETS_CONF_NOC',FALSE)
```

bimetsDataset *BIMETS Internal Datasets*

Description

BIMETS package contains hidden datasets that allow a faster performance in time series analysis.

See Also

TIMESERIES
is.bimets
as.bimets
fromBIMETStoXTS
fromBIMETStoTS
BIMETS indexing

CUMPROD *Cumulative Product*

Description

This function returns the cumulative product of the elements of the input array or time series. The result is an object of the same class of the input, and its elements are the cumulative product of the current and the previous elements of the input.
If the input is a time series, users can provide the argument TSRANGE in order to project the input time series before the cumulative product.

Usage

```
CUMPROD(x=NULL, TSRANGE=NULL, avoidCompliance=FALSE, ...)
```

Arguments

x	Input numerical array or time series that must satisfy the compliance control check defined in is.bimets.
TSRANGE	Optional date range of operations that process the input time series. TSRANGE must be specified as an array composed by starting year, starting period, ending year and ending period, i.e. TSRANGE=c(START_Y,START_P,END_Y,END_P). The projection into the time interval specified in TSRANGE takes place before the cumulative product.
avoidCompliance	If TRUE, compliance control check of input time series will be skipped. See is.bimets
...	Backward compatibility.

Value

This function returns an object of the same class of the input, i.e. an array or a BIMETS time series.

See Also

TSPROJECT
MOVAVG
TSDELTA
TSLAG
TSPROJECT
TSEXTEND
TSLEAD
INDEXNUM
VERIFY_MAGNITUDE

Examples

```
#create ts
ts1<-TSERIES(INTS(1,10),START=c(2000,1),FREQ='M')
out_CUMPROD<-CUMPROD(ts1)
TABIT(out_CUMPROD)

out_CUMPROD<-CUMPROD(ts1,TSRANGE=c(2000,4,2001,1))
TABIT(ts1,out_CUMPROD)

#define an array
arr1<-c(INTS(1,5),INTS(-1,-5))
out_CUMPROD<-CUMPROD(arr1)
print(out_CUMPROD)
```

CUMSUM *Cumulative Sum*

Description

This function returns the cumulative sum of the elements of the input array or time series. The result is an object of the same class of the input, and its elements are the cumulative sum of the current and the previous elements of the input.
If the input is a time series, users can provide the argument TSRANGE in order to project the input time series before the cumulative sum.
CUMULO is an alias form CUMSUM with the argument MODE='YEARLY'.

Usage

```
CUMSUM(x=NULL, TSRANGE=NULL, MODE=NULL, avoidCompliance=FALSE,...)
CUMULO(x=NULL, TSRANGE=NULL, avoidCompliance=FALSE,...)
```

Arguments

x	Input numerical array or time series that must satisfy the compliance control check defined in is.bimets.
TSRANGE	Optional date range of operations that process time series. TSRANGE must be specified as an array composed by starting year, starting period, ending year and ending period, i.e. TSRANGE=c(START_YEAR,START_PERIOD,END_YEAR,END_PERIOD). The projection into the time interval specified in TSRANGE takes place before the cumulative sum.
MODE	When selecting MODE='YEARLY' or MODE='MONTHLY' the sum is reset to zero when a new year or a new month starts.
avoidCompliance	If TRUE, compliance control check of input time series will be skipped. See is.bimets
...	Backward compatibility.

Value

This function returns an object of the same class of the input, i.e. an array or a BIMETS time series.

See Also

TSPROJECT
MOVAVG
TSDELTA
TSLAG
TSPROJECT
TSEXTEND
TSLEAD
INDEXNUM
CUMPROD
VERIFY_MAGNITUDE
GETRANGE

Examples

```
#create ts
ts1<-TSERIES(INTS(1,30),START=c(2000,1),FREQ='M')
out_CUMSUM<-CUMSUM(ts1)
TABIT(out_CUMSUM)
```

```
out_CUMSUM<-CUMSUM(ts1,TSRANGE=c(2000,4,2001,7))
out_CUMSUM_Y<-CUMSUM(ts1,TSRANGE=c(2000,4,2001,7),MODE='YEARLY')
TABIT(ts1,out_CUMSUM,out_CUMSUM_Y)



#define an array
arr1<-c(INTS(1,10),INTS(-1,-10))
out_CUMSUM<-CUMSUM(arr1)
print(out_CUMSUM)

#print...1  3  6 10 15 21 ... 27 19 10  0
```

DAILY *Daily Time Series (Dis)Aggregation*

Description

This function returns a daily disaggregated time series, by using as input an annual, semiannual, quarterly or monthly time series.

Usage

```
DAILY(x = NULL, fun = NULL, avoidCompliance = FALSE, ...)
```

Arguments

x	Input time series, that must satisfy the compliance control check defined in is.bimets.
fun	**NULL**: (default) the output value of each daily observation is set equal to the value of the input observation the date belongs to (i.e. duplicated values over the period) **INTERP_END**: the value of the input time series in a period is copied into the last day of the output time series that lies in the same period. Other values are calculated by linear interpolation. **INTERP_CENTER**: the value of the input time series in a period is copied into the median day of the output time series that lies in the same period. Other values are calculated by linear interpolation. **INTERP_BEGIN**: the value of the input time series in a period is copied into the first day of the output time series that lies in the same period. Other values are calculated by linear interpolation.
avoidCompliance	If TRUE, compliance control check of input time series will be skipped. See is.bimets
...	Backward compatibility.

Value

This function returns a daily BIMETS time series.

See Also

YEARLY
SEMIANNUAL
QUARTERLY
MONTHLY

Examples

```
#TS quarterly
ts1<-TSERIES((1:2),START=c(2000,1),FREQ='Q')
TABIT(DAILY(ts1,fun='INTERP_CENTER'))

#TS monthly
ts1<-TSERIES((1:4),START=c(2000,1),FREQ=12)
TABIT(DAILY(ts1))
```

---

date2yp *Date to Year-Period Conversion*

Description

This function converts an object of class `Date()` to an array `c(YEAR,PERIOD)`, where `YEAR` and `PERIOD` are the year-period the input `Date()` belongs to, given an input frequency.

Usage

```
date2yp(x = NULL, f = 1)
```

Arguments

`x`	Input of class `Date()`.
`f`	Positive integer. Valid values are: 1, 2, 3, 4, 12, 24, 36, 53, 366

Value

This function returns a two-dimensional array `c(YEAR,PERIOD)`.

See Also

yq2yp
ym2yp
GETDATE
INTS
TABIT

Examples

```
print(date2yp(as.Date('2001/06/30'),2)) #2001,1
print(date2yp(as.Date('2002/03/23'),1)) #2002,1
print(date2yp(as.Date('2003/07/01'),366)) #2003,182
print(date2yp(as.Date('2004/09/13'),2)) #2004,2
print(date2yp(as.Date('2004/01/13'),12)) #2004,1
print(date2yp(as.Date('2004/07/24'),4)) #2004,3
print(date2yp(c(as.Date('1900-01-01'),as.Date('1944-12-01'),
as.Date('1964-06-12'),as.Date('1923-03-01'),
as.Date('1943-12-05')),f=366)) #...
```

ELIMELS *Eliminate Elements from Arrays or Time Series*

Description

This function eliminates the selected elements from the input array or the input time series.

Usage

```
ELIMELS(x=NULL, idx=NULL, avoidCompliance=FALSE, ...)
```

Arguments

x	Input numerical array or time series that must satisfy the compliance control check defined in is.bimets.
idx	Numerical array built with the indices of selected elements to be removed from the input. If the input is a time series the index must be provided as a sequence of numbers IDX=YEAR+PERIOD/FREQ with YEAR and PERIOD the year and the period to be removed (see example).
avoidCompliance	If TRUE, compliance control check of input time series will be skipped. See is.bimets
...	Backward compatibility.

Value

This function returns an array with the kept elements from the input array or the input time series.

See Also

TIMESERIES
is.bimets
BIMETS indexing
GETYEARPERIOD
LOCS
NAMELIST
INTS
TSINFO
TSLOOK
TABIT
NOELS

Examples

```
print(ELIMELS(INTS(1,10),c(1,4,5)))
#print 2 3 6 7 8 9 10

print(ELIMELS(TSERIES(1:10,START=c(2000,1),FREQ=4),c(2000.25,2000.75)))
#print 2  4  5  6  7  8  9 10
```

ESTIMATE *Estimate a BIMETS model*

Description

This function estimates equations that are linear in the coefficients, as specified in the behavioral equations of the input model object. Coefficients can be estimated for single equations or blocks of simultaneous equations. Coefficients restriction procedure derives from Lagrange Multipliers' theory, while the Cochrane-Orcutt method allows accounting for residuals autocorrelation.

The estimation function supports:

- *Ordinary Least Squares*;
- *Instrumental Variables*;
- *Deterministic linear restrictions on the coefficients*;
- *Almon Polynomial Distributed Lags*;
- *Autocorrelation of the errors*;
- *Structural stability analysis (Chow tests)*;

Further details on estimation calculus can be found in MDL help page; further details on Chow test can be found below in this section;

Usage

```
ESTIMATE(model=NULL,
         eqList=NULL,
         TSRANGE=NULL,
         forceTSRANGE=FALSE,
         estTech='OLS',
         IV=NULL,
         forceIV=FALSE,
         quietly=FALSE,
         tol=1e-28,
         digits=getOption('digits'),
         centerCOV=TRUE,
         CHOWTEST=FALSE,
         CHOWPAR=NULL,
         avoidCompliance=FALSE,
         ...)
```

Arguments

model	The BIMETS model object to be estimated (see also LOAD_MODEL).
eqList	The character array of behavioral names to be estimated. If it is NULL then all the behaviorals of the model will be estimated.
TSRANGE	The time range of the estimation, as a four dimensional numerical array, i.e. TSRANGE=c(start_year, start_period, end_year, end_period). The TSRANGE provided in the behavioral MDL definition takes precedence over this function argument.
forceTSRANGE	If TRUE, the TSRANGE defined in the previous argument takes precedence over the TSRANGE provided in the behavioral MDL definition.
estTech	The estimation technique used in the regression. Ordinary Least Squares OLS and Instrumental Variables estimation IV are supported.
IV	The character array built with the Instrumental Variable expressions, in the case of Instrumental Variables estimation (see example).
forceIV	If TRUE, the IV defined in the previous argument takes precedence over the IV provided in the behavioral MDL definition.
quietly	If TRUE, information messages will be suppressed, e.g. results and regression statistics.
tol	The tolerance for detecting linear dependencies in a matrix's columns when its inverse is requested.
digits	Controls the number of digits to print when printing coefficients and statistics of the estimation. Valid values are 1 to 22 with a default of 7.
centerCOV	If TRUE, the function subtracts the mean from the residuals before calculating the residual covariance matrix.
CHOWTEST	If TRUE, the structural stability analysis will be performed.

CHOWPAR — Indicates the last year-period where the stability test is performed. If NULL it will be automatically calculated by using all available time series data. It must be provided as an integer array, e.g. c(YEAR,PERIOD).

avoidCompliance — If TRUE, compliance control check of model time series will be skipped. See is.bimets

... — Backward compatibility.

Value

If outputText=TRUE, for each behavioral in the eqList this function will print out:

- the name of the estimated behavioral;
- the estimation technique used;
- the autocorrelation order of the error, if any, and the iterations count required to achieve the convergence;
- the estimated equation with calculated coefficients and regressor expression; for each coefficient the T-statistic and the significance will be printed out;
- the restriction equations imposed on the coefficients, if any;
- the F-test for the restrictions, including the PDL restrictions, if any;
- the final autocorrelation parameters for the error, along with their standard error, the T-statistic and the significance;
- the *R-Squared* and the *Adjusted R-Squared*;
- the *Durbin-Watson Statistic*;
- the *Sum of squares of residuals*;
- the *Standard Error of Regression*;
- the *Log of the Likelihood Function*;
- the *F-statistic* and the *F-probability*;
- the *AIC* and the *BIC*;
- the *Mean of the Dependent Variable*;
- the *Number of Observations*;
- the *Number of Degrees of Freedom*;
- the *Current Sample*, i.e. the TSRANGE of estimation;

All probabilities lie between [0, 1].

For each behavioral in the eqList this function will add 4 new named elements into the related behavioral of the output model object:

1) coefficients: a numerical array built with the estimated coefficients;
2) errorCoefficients: a numerical array built with the estimated coefficient for the error autoregression, if the ERROR> structure has been provided in the model MDL definition;
3) residuals: the time series of the regression residuals. If an ERROR> structure has been provided in the behavioral definition, the related residuals will be calculated as described in the Cochrane-Orcutt procedure (see MDL).
3) residuals_no_error_correction: if an ERROR> structure has been provided in the behavioral definition, the residuals calcucated by using the original dependent and independent variables are stored into this list element.

4) `statistics`: a list built with the parameters and the statistics of the estimation, e.g.:
- `TSRANGE`: TSRANGE requested in the latest estimation of the behavioral;
- `estimationTechinque`: estimation technique requested in the latest estimation of the behavioral;
- `CoeffCovariance`: coefficients covariance;
- `StandardErrorRegression` and `StandardErrorRegressionNotCentered`: standard error of the regression (centered and not-centered);
- `CoeffTstatistic`: T-statistic of the coefficients;
- `RSquared`: R-Squared;
- `AdjustedRSquared`: adjusted R-Squared;
- `DegreesOfFreedom`: degrees of freedom of the regression;
- `CoeffPvalues`: coefficients p-values;
- `LogLikelihood`: Log of the Likelihood Function;
- `Fstatistics`: F-statistics;
- `RhosTstatistics`: rhos T-statistic (if any);
- `FtestRestrValue`: F-test value for the restrictions;
- `FtestRestrProbability`: F-test probability for the restrictions;
- `AIC`: Akaike's Information Criterion;
- `BIC`: Schwarz's Information Criterion;
- `matrixX`: the regressors matrix;
- `vectorY`: the dependent variable;
- `matrixX_error_corrected`: the regressors matrix arising from the Cochrane-Orcutt procedure;
- etc.

Structural Stability - Chow test

One of the main purposes of econometric modeling is its use for forecast and policy evaluation and, to this end, the stability of any behavioral equation parameters over time should be verified. In order to check for structural stability two different procedures, which can be derived from the so-called Chow-tests, are applied.

Given a sample of $T_0 = t_k, ..., t_n$ observations (i.e. the base `TSRANGE`) and selecting an arbitrary forward extension in $T_1 = t_k, ..., t_n, ..., t_m$ observations (i.e. the extended `TSRANGE`), with $k < n < m$, in the general case we have the following two regressions:

1) $Y_0 = \beta_0 * X_0 + \epsilon_0, \quad \epsilon_0 \sim \mathcal{N}(0, \sigma_0^2)$, having time series projected on the base `TSRANGE`
2) $Y_1 = \beta_1 * X_1 + \epsilon_1, \quad \epsilon_1 \sim \mathcal{N}(0, \sigma_1^2)$, having time series projected on the extended `TSRANGE`

In general, a stability analysis is carried on in the following ways:
- comparing the parameter estimates arising from the two regressions: this is known as the covariance analysis;
- checking the accuracy of the forecast for the dependent variable in the extended `TSRANGE`, using the estimates produced in the base `TSRANGE`: this is known as the predictive power test.

The first Chow test is calculated as: $\tau = \frac{SSR_1 - SSR_0}{SSR_0} \frac{DoF_1}{DoF_1 - DoF_0}$,

with SSR_i as the sum of squared residuals and DoF_i as the number of degrees of freedom in the regression $i = 0, 1$.

The test is completed by calculating the following time series on the extended TSRANGE:
- the forecast error;
- the standard error of forecast;
- the t-statistic for the error;

The standard error of the forecast for the t_j observation in the extended TSRANGE is computed according to:

$$SE_j = \sigma_0 \sqrt{1 + x_j^\top * (X_0^\top * X_0)^{-1} * x_j}$$

having x_j as the independent values (i.e. regressors) on the t_j observation in the T_1 extended TSRANGE, with $n < j \leq m$.

The null hypothesis for τ is:

$H^* : \beta_1 = \beta_0$, given $\sigma_1^2 = \sigma_0^2$

The test statistic τ follows the F distribution with $(DoF_1 - DoF_0)$ and DoF_1 degrees of freedom, and can be performed during the ESTIMATE() function execution by using the CHOWTEST argument set to TRUE.

If CHOWTEST is TRUE, for each behavioral in the eqList the output model will have the following named element:
- ChowTest: it contains the statistics and the time series computed during the last structural analysis performed on the related behavioral.

See Also

MDL
LOAD_MODEL
SIMULATE
STOCHSIMULATE
MULTMATRIX
RENORM
OPTIMIZE
TIMESERIES
BIMETS indexing
BIMETS configuration
summary

Examples

```
#define model
myModelDefinition<-
"MODEL
```

```
COMMENT> Modified Klein Model 1 of the U.S. Economy with PDL,
COMMENT> autocorrelation on errors, restrictions
COMMENT> and conditional evaluations

COMMENT> Consumption
BEHAVIORAL> cn
TSRANGE 1925 1 1941 1
EQ> cn = a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)
COEFF> a1 a2 a3 a4
ERROR> AUTO(2)

COMMENT> Investment
BEHAVIORAL> i
TSRANGE 1923 1 1941 1
EQ> i = b1 + b2*p + b3*TSLAG(p,1) + b4*TSLAG(k,1)
COEFF> b1 b2 b3 b4
RESTRICT> b2 + b3 = 1

COMMENT> Demand for Labor
BEHAVIORAL> w1
TSRANGE 1925 1 1941 1
EQ> w1 = c1 + c2*(y+t-w2) + c3*TSLAG(y+t-w2,1) + c4*time
COEFF> c1 c2 c3 c4
PDL> c3 1 3

COMMENT> Gross National Product
IDENTITY> y
EQ> y = cn + i + g - t

COMMENT> Profits
IDENTITY> p
EQ> p = y - (w1+w2)

COMMENT> Capital Stock with switches
IDENTITY> k
EQ> k = TSLAG(k,1) + i
IF> i > 0
IDENTITY> k
EQ> k = TSLAG(k,1)
IF> i <= 0
END"

#define model data
myModelData<-list(
  cn
  =TIMESERIES(39.8,41.9,45,49.2,50.6,52.6,55.1,56.2,57.3,57.8,55,50.9,
              45.6,46.5,48.7,51.3,57.7,58.7,57.5,61.6,65,69.7,
              START=c(1920,1),FREQ=1),
  g
  =TIMESERIES(4.6,6.6,6.1,5.7,6.6,6.5,6.6,7.6,7.9,8.1,9.4,10.7,10.2,9.3,10,
              10.5,10.3,11,13,14.4,15.4,22.3,
              START=c(1920,1),FREQ=1),
```

```
  i
  =TIMESERIES(2.7,-.2,1.9,5.2,3,5.1,5.6,4.2,3,5.1,1,-3.4,-6.2,-5.1,-3,-1.3,
              2.1,2,-1.9,1.3,3.3,4.9,
              START=c(1920,1),FREQ=1),
  k
  =TIMESERIES(182.8,182.6,184.5,189.7,192.7,197.8,203.4,207.6,210.6,215.7,
              216.7,213.3,207.1,202,199,197.7,199.8,201.8,199.9,
              201.2,204.5,209.4,
              START=c(1920,1),FREQ=1),
  p
  =TIMESERIES(12.7,12.4,16.9,18.4,19.4,20.1,19.6,19.8,21.1,21.7,15.6,11.4,
              7,11.2,12.3,14,17.6,17.3,15.3,19,21.1,23.5,
              START=c(1920,1),FREQ=1),
  w1
  =TIMESERIES(28.8,25.5,29.3,34.1,33.9,35.4,37.4,37.9,39.2,41.3,37.9,34.5,
              29,28.5,30.6,33.2,36.8,41,38.2,41.6,45,53.3,
              START=c(1920,1),FREQ=1),
  y
  =TIMESERIES(43.7,40.6,49.1,55.4,56.4,58.7,60.3,61.3,64,67,57.7,50.7,41.3,
              45.3,48.9,53.3,61.8,65,61.2,68.4,74.1,85.3,
              START=c(1920,1),FREQ=1),
  t
  =TIMESERIES(3.4,7.7,3.9,4.7,3.8,5.5,7,6.7,4.2,4,7.7,7.5,8.3,5.4,6.8,7.2,
              8.3,6.7,7.4,8.9,9.6,11.6,
              START=c(1920,1),FREQ=1),
  time
  =TIMESERIES(NA,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,
              START=c(1920,1),FREQ=1),
  w2
  =TIMESERIES(2.2,2.7,2.9,2.9,3.1,3.2,3.3,3.6,3.7,4,4.2,4.8,5.3,5.6,6,6.1,
              7.4,6.7,7.7,7.8,8,8.5,
              START=c(1920,1),FREQ=1)
)


#load model
myModel<-LOAD_MODEL(modelText=myModelDefinition)

#load data into the model
myModel<-LOAD_MODEL_DATA(myModel,myModelData,showWarnings = TRUE)

################################################
#OLS case

#estimate the model
myModel<-ESTIMATE(myModel)

#HERE BELOW THE OUTPUT OF THE ESTIMATION (COMMENTED OUT):
#.CHECK_MODEL_DATA(): warning, there are undefined values in time series "time".
#
#Estimate the Model myModelDefinition:
#the number of behavioral equations to be estimated is 3.
#The total number of coefficients is 14.
```

```
#
#_____________________________________________
#
#BEHAVIORAL EQUATION: cn
#Estimation Technique: OLS
#Autoregression of Order  2  (Cochrane-Orcutt procedure)
#
#Convergence was reached in  9  /  20  iterations.
#
#
#cn                  =   19.01352
#                        T-stat. 12.13083    ***
#
#                    +   0.3442816   p
#                        T-stat. 3.533253    **
#
#                    +   0.03443117  TSLAG(p,1)
#                        T-stat. 0.3937881
#
#                    +   0.6993905   (w1+w2)
#                        T-stat. 14.0808     ***
#
#ERROR STRUCTURE:  AUTO(2)
#
#AUTOREGRESSIVE PARAMETERS:
#Rho              Std. Error      T-stat.
# 0.05743131      0.3324101       0.1727725
# 0.007785936     0.2647013       0.02941404
#
#
#STATs:
#R-Squared                      : 0.985263
#Adjusted R-Squared             : 0.9785644
#Durbin-Watson Statistic        : 1.966609
#Sum of squares of residuals    : 9.273455
#Standard Error of Regression   : 0.9181728
#Log of the Likelihood Function : -18.97047
#F-statistic                    : 147.0844
#F-probability                  : 1.090551e-09
#Akaike's IC                    : 51.94093
#Schwarz's IC                   : 57.77343
#Mean of Dependent Variable     : 55.71765
#Number of Observations         : 17
#Number of Degrees of Freedom   : 11
#Current Sample (year-period)   : 1925-1 / 1941-1
#
#
#Signif. codes:   *** 0.001  ** 0.01  * 0.05
#
#
#
#_____________________________________________
#
```

```
#BEHAVIORAL EQUATION: i
#Estimation Technique: OLS
#
#i                   =   2.868104
#                        T-stat. 0.3265098
#
#                    +   0.5787626   p
#                        T-stat. 4.456542    ***
#
#                    +   0.4212374   TSLAG(p,1)
#                        T-stat. 3.243579    **
#
#                    -   0.09160307  TSLAG(k,1)
#                        T-stat. -2.11748
#
#RESTRICTIONS:
#b2+b3=1
#
#RESTRICTIONS F-TEST:
#F-value            : 8.194478
#F-prob(1,15)       : 0.0118602
#
#
#STATs:
#R-Squared                      : 0.8928283
#Adjusted R-Squared             : 0.8794319
#Durbin-Watson Statistic        : 1.173106
#Sum of squares of residuals    : 26.76483
#Standard Error of Regression   : 1.293368
#Log of the Likelihood Function : -30.215
#F-statistic                    : 66.64659
#F-probability                  : 1.740364e-08
#Akaike's IC                    : 68.43001
#Schwarz's IC                   : 72.20776
#Mean of Dependent Variable     : 1.310526
#Number of Observations         : 19
#Number of Degrees of Freedom   : 16
#Current Sample (year-period)   : 1923-1 / 1941-1
#
#
#Signif. codes:   *** 0.001  ** 0.01  * 0.05
#
#
#
#_________________________________________
#
#BEHAVIORAL EQUATION: w1
#Estimation Technique: OLS
#
#w1                  =   1.12869
#                        T-stat. 0.6479266
#
#                    +   0.4398767   (y+t-w2)
```

```
#                         T-stat. 12.01268    ***
#
#                     +   c3          TSLAG(y+t-w2,1)
#                         PDL
#
#                     +   0.1368206   time
#                         T-stat. 3.373905    **
#
#PDL:
#c3 1 3
#
#Distributed Lag Coefficient: c3
#Lag     Coeff.         Std. Error     T-stat.
#0        0.1076812      0.04283967     2.513586      *
#1        0.05074557     0.01291231     3.930015      **
#2       -0.00619005     0.03110492    -0.1990055
#SUM      0.1522367      0.03873693
#
#RESTRICTIONS F-TEST:
#F-value            : 0.06920179
#F-prob(1,11)       : 0.7973647
#
#
#STATs:
#R-Squared                      : 0.9890855
#Adjusted R-Squared             : 0.9854474
#Durbin-Watson Statistic        : 2.174168
#Sum of squares of residuals    : 6.392707
#Standard Error of Regression   : 0.7298805
#Log of the Likelihood Function : -15.80848
#F-statistic                    : 271.8645
#F-probability                  : 1.172284e-11
#Akaike's IC                    : 43.61697
#Schwarz's IC                   : 48.61625
#Mean of Dependent Variable     : 37.69412
#Number of Observations         : 17
#Number of Degrees of Freedom   : 12
#Current Sample (year-period)   : 1925-1 / 1941-1
#
#
#Signif. codes:   *** 0.001  ** 0.01  * 0.05
#
#
#...ESTIMATE OK

#get residuals of 'cn'
myModel$behaviorals$cn$residuals
#Time Series:
#Start = 1925
#End = 1941
#Frequency = 1
# [1] -0.88562504  0.25109884  0.66750111  ...
#[17] -1.41795908
```

```
#get residuals of 'i'
myModel$behaviorals$i$residuals
#Time Series:
#Start = 1923
#End = 1941
#Frequency = 1
# [1]  1.464518775 -1.469763968  0.078674017  ...
#[16] -2.425079127 -0.698071507 -1.352967430 -1.724306054

#get estimation coefficients of 'cn' and 'w1'
myModel$behaviorals$cn$coefficients
#          [,1]
#a1 19.01352476
#a2  0.34428157
#a3  0.03443117
#a4  0.69939052

myModel$behaviorals$cn$errorCoefficients
#             [,1]
#RHO_1 0.057431312
#RHO_2 0.007785936

myModel$behaviorals$w1$coefficients
#                [,1]
#c1         1.12869024
#c2         0.43987666
#c3         0.10768118
#c3_PDL_1   0.05074557
#c3_PDL_2 -0.00619005
#c4         0.13682057


################################################
#IV case

#estimation of Consumption "cn" with arbitrary IVs
#and error autocorrelation
myModel<-ESTIMATE(myModel,
                  eqList = 'cn',
                  estTech = 'IV',
                  IV=c('1',
                       'TSLAG(y)',
                       'TSLAG(w1)*pi+0.5',
                       'exp(w2)'))
#Estimate the Model myModelDefinition:
#the number of behavioral equations to be estimated is 1.
#The total number of coefficients is 4.
#
#_________________________________________
#
#BEHAVIORAL EQUATION: cn
#Estimation Technique: IV
```

```
#Autoregression of Order  2  (Cochrane-Orcutt procedure)
#
#Convergence was reached in  7  /  20  iterations.
#
#
#cn                  =   18.07073
#                        T-stat. 11.72958    ***
#
#                    +   0.2530483   p
#                        T-stat. 1.583881
#
#                    +   0.08631646  TSLAG(p,1)
#                        T-stat. 0.7556204
#
#                    +   0.7363227   (w1+w2)
#                        T-stat. 13.11572    ***
#
#ERROR STRUCTURE:  AUTO(2)
#
#AUTOREGRESSIVE PARAMETERS:
#Rho             Std. Error      T-stat.
#0.01559806      0.343195        0.04544955
#-0.1196327       0.283432       -0.422086
#
#
#STATs:
#R-Squared                       : 0.9843186
#Adjusted R-Squared              : 0.9771907
#Durbin-Watson Statistic         : 1.917329
#Sum of squares of residuals     : 9.867739
#Standard Error of Regression    : 0.9471363
#Log of the Likelihood Function : -19.49844
#F-statistic                     : 138.0938
#F-probability                   : 1.532807e-09
#Akaike's IC                     : 52.99689
#Schwarz's IC                    : 58.82938
#Mean of Dependent Variable      : 55.71765
#Number of Observations          : 17
#Number of Degrees of Freedom    : 11
#Current Sample (year-period)    : 1925-1 / 1941-1
#
#
#Signif. codes:   *** 0.001  ** 0.01  * 0.05
#
#
#...ESTIMATE OK

#define model
myShortModelDefinition<-
  "MODEL
  COMMENT> Consumption with IV
  BEHAVIORAL> cn
  TSRANGE 1925 1 1941 1
```

```
  EQ> cn = a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)
  COEFF> a1 a2 a3 a4
  IV> 1
  IV> TSLAG(y)
  IV> TSLAG(w1)*pi+0.5
  IV> exp(w2)
  END
  "


#load model
myShortModel<-LOAD_MODEL(modelText=myShortModelDefinition)

#load data into the model
myShortModel<-LOAD_MODEL_DATA(myShortModel,myModelData,showWarnings = TRUE)

#estimation of Consumption "cn" with arbitrary IVs
#and error autocorrelation
myShortModel<-ESTIMATE(myShortModel,
                 eqList = 'cn',
                 estTech = 'IV')


#estimation of Investment "i" with arbitrary IVs
#and coefficient restrictions
myModel<-ESTIMATE(myModel,
                 eqList = 'i',
                 estTech = 'IV',
                 IV=c('1',
                      'TSLAG(w2)',
                      'TSLAG(w1)*pi+0.5',
                      'exp(w2)'))

#.CHECK_MODEL_DATA(): warning, there are undefined values in time series "time".
#
#Estimate the Model myModelDefinition:
#the number of behavioral equations to be estimated is 1.
#The total number of coefficients is 4.
#
#_________________________________________
#
#BEHAVIORAL EQUATION: i
#Estimation Technique: IV
#
#i                   =   34.517544
#                        T-stat. 1.264388
#
#                    +   0.3216326   p
#                        T-stat. 0.8648297
#
#                    +   0.6783672   TSLAG(p,1)
#                        T-stat. 1.824043
#
```

```
#                     -   0.2475568   TSLAG(k,1)
#                         T-stat. -1.842520
#
#RESTRICTIONS:
#b2+b3=1
#
#RESTRICTIONS F-TEST:
#F-value            : 2.465920
#F-prob(1,15)       : 0.137190
#
#
#STATs:
#R-Squared                      : 0.805773
#Adjusted R-Squared             : 0.781494
#Durbin-Watson Statistic        : 0.940534
#Sum of squares of residuals    : 48.50580
#Standard Error of Regression   : 1.741152
#Log of the Likelihood Function : -35.86365
#F-statistic                    : 33.18894
#F-probability                  : 2.025229e-06
#Akaike's IC                    : 79.72731
#Schwarz's IC                   : 83.50506
#Mean of Dependent Variable     : 1.310526
#Number of Observations         : 19
#Number of Degrees of Freedom   : 16
#Current Sample (year-period)   : 1923-1 / 1941-1
#
#
#Signif. codes:   *** 0.001  ** 0.01  * 0.05
#
#
#...ESTIMATE OK

################################################################
#CHOW TEST on w1

#base TSRANGE set to 1925 / 1935
myModel<-ESTIMATE(myModel,
                  eqList='w1',
                  TSRANGE=c(1925,1,1935,1),
                  forceTSRANGE=TRUE,
                  CHOWTEST=TRUE)

#Estimate the Model myModelDefinition:
#the number of behavioral equations to be estimated is 1.
#The total number of coefficients is 6.
#
#_________________________________________
#
#BEHAVIORAL EQUATION: w1
#Estimation Technique: OLS
#
#w1                = -   4.48873
```

```
#                        T-stat. -2.47402   *
#
#                    +   0.545102   (y+t-w2)
#                        T-stat. 15.3462    ***
#
#                    +   c3         TSLAG(y+t-w2,1)
#                        PDL
#
#                    +   0.292018   time
#                        T-stat. 5.58588    **
#
#PDL:
#c3 1 3
#
#Distributed Lag Coefficient: c3
#Lag     Coeff.         Std. Error    T-stat.
#0        0.0413985     0.0336676     1.22963
#1        0.0493551     0.00742323    6.64873      ***
#2        0.0573116     0.0265487     2.15873
#SUM      0.148065      0.0222697
#
#RESTRICTIONS F-TEST:
#F-value            : 3.35954
#F-prob(1,5)        : 0.126295
#
#
#STATs:
#R-Squared                      : 0.995931
#Adjusted R-Squared             : 0.993219
#Durbin-Watson Statistic        : 2.43313
#Sum of squares of residuals    : 0.737093
#Standard Error of Regression   : 0.350498
#Log of the Likelihood Function : -0.742173
#F-statistic                    : 367.183
#F-probability                  : 2.68564e-07
#Akaike's IC                    : 13.4843
#Schwarz's IC                   : 15.8717
#Mean of Dependent Variable     : 34.9909
#Number of Observations         : 11
#Number of Degrees of Freedom   : 6
#Current Sample (year-period)   : 1925-1 / 1935-1
#
#
#Signif. codes:   *** 0.001  ** 0.01  * 0.05
#
#
#
#STABILITY ANALYSIS:
#Behavioral equation: w1
#
#Chow test:
#Sample (auto)      : 1936-1 / 1941-1
#F-value            : 15.3457
```

```
#F-prob(6,12)       : 5.34447e-05
#
#Predictive Power:
#
#Date, Prd., Actual        , Predict       , Error         , Std. Error    , T-stat
#
#1936, 1   ,  36.8         ,  38.439       , -1.63901      ,  0.547471     , -2.99378
#1937, 1   ,  41           ,  40.824       ,  0.176033     ,  0.630905     ,  0.279017
#1938, 1   ,  38.2         ,  39.6553      , -1.4553       ,  0.672192     , -2.165
#1939, 1   ,  41.6         ,  45.0547      , -3.45466      ,  0.834433     , -4.14012
#1940, 1   ,  45           ,  49.0118      , -4.01179      ,  0.966472     , -4.15096
#1941, 1   ,  53.3         ,  56.6727      , -3.37275      ,  1.23486      , -2.73127
#
#
#...ESTIMATE OK
```

`frequency` *Frequency of a Time Series*

Description

This function returns the frequency of a time series. In the case of a sparse `xts()` time series, and in other cases, the R functions `xts::periodicity()` and `frequency()` do not return BIMETS compliant values. Therefore, these functions have been extended.

Usage

```
## S3 method for class 'xts'
frequency(x,...)
```

Arguments

`x`	Input time series.
`...`	Backward compatibility.

Value

This function returns the integer value stored in the attribute `.bimetsFreq` of the input time series, if any. Otherwise, the frequency will be calculated by using the shortest time difference between two observations, while accounting for day-saving and bissextile years.

See Also

`normalizeYP`
`NUMPERIOD`
`BIMETS indexing`

Examples

```
#build a sparse xts()
xArr<-rnorm(13)
dateArr<-seq(as.Date('2000/01/01'),by='6 months',length=10)
dateArr2<-seq(as.Date('2010/01/01'),by='3 months',length=3)

#strange array of dates here below...
dateArr3<-c(dateArr,dateArr2)
dataF<-data.frame(dateArr3,xArr)
xt<-xts(dataF[,2],order.by=dataF[,1])

#get bimets calculated frequency
cat(frequency(xt)) #print 4... without bimets R returns 1

#...legacy periodicity()
periodicity(xt)
```

---

fromBIMETStoTS *Convert BIMETS to TS*

Description

This function transforms a BIMETS compliant time series (as defined in `is.bimets`) into a time series of class `ts()`. The core R function as.ts() does not satisfy all the compliance control check requirements, so it has been extended. Attributes and description of the input time series will be copied to the output time series (see `TIMESERIES`).

Usage

```
fromBIMETStoTS(x = NULL, ...)
```

Arguments

`x`	Input time series that must satisfy the compliance control check defined in `is.bimets`.
`...`	Backward compatibility.

Value

This function returns a time series of class `ts()` that has the same observations of the input BIMETS time series.

See Also

fromBIMETStoXTS
as.bimets
is.bimets
BIMETS indexing
BIMETS configuration

Examples

```
#work with XTS
setBIMETSconf('BIMETS_CONF_CCT','XTS')

#create yearly time series
xts<-TSERIES(1:10,START=c(2000,1),FREQ='A')
print(is.ts(xts))#FALSE
#convert to ts
ts<-fromBIMETStoTS(xts)
print(is.ts(ts))#TRUE
print(ts)

#create monthly time series
xts<-TSERIES(1:10,START=c(2000,1),FREQ='M')
print(is.ts(xts))#FALSE
#convert to ts
ts<-fromBIMETStoTS(xts)
print(is.ts(ts))#TRUE
print(ts)

#create daily time series
xts<-TSERIES(1:10,START=c(2000,1),FREQ='D')
print(is.ts(xts))#FALSE
#convert to ts
ts<-fromBIMETStoTS(xts)
print(is.ts(ts))#TRUE
print(ts)



#reset default
setBIMETSconf('BIMETS_CONF_CCT','TS')
```

---

fromBIMETStoXTS *Convert BIMETS to XTS*

Description

This function transforms a BIMETS compliant time series (as defined in is.bimets) into a time series of class xts().
The core XTS function as.xts() does not satisfy all the compliance control check requirements, so it has been extended. If the output time series will have an .indexClass of type Date(), i.e. neither monthly nor quarterly, the output dates will be chosen accordingly to the BIMETS option BIMETS_CONF_DIP: if this option is set to LAST (default), the output xts() time series will have the date of the period set equal to the last day in the same period, e.g. 31 December for yearly time series, 30 June for semiannual, etc.; if the BIMETS option BIMETS_CONF_DIP is set to FIRST, the output xts() time series will have the date of the period set equal to the first day in the same period, e.g. 1 January for yearly time series, 1 July for semiannual time series on the second period, etc.
In the case of quarterly time series the .indexClass=yearqtr;
in the case of monthly time series the .indexClass=yearmon.
Attributes and description of the input time series will be copied to the output time series (see TIMESERIES)

Usage

```
fromBIMETStoXTS(x = NULL, ...)
```

Arguments

x	Input time series that must satisfy the compliance control check defined in is.bimets.
...	Backward compatibility.

Value

This function returns a time series of class xts() that has the same observations of the input BIMETS time series.

See Also

fromBIMETStoTS
as.bimets
is.bimets
BIMETS indexing
BIMETS configuration

Examples

```
#create yearly time series
ts<-TSERIES(1:10,START=c(2000,1),FREQ='A')
print(is.xts(ts))#FALSE
#convert to xts
xts<-fromBIMETStoXTS(ts)
print(is.xts(xts))#TRUE
print(xts)
```

```
#create monthly time series
ts<-TSERIES(1:10,START=c(2000,1),FREQ='M')
print(is.xts(ts))#FALSE
#convert to xts
xts<-fromBIMETStoXTS(ts)
print(is.xts(xts))#TRUE
print(xts)

#create daily time series
ts<-TSERIES(1:10,START=c(2000,1),FREQ='D')
print(is.xts(ts))#FALSE
#convert to xts
xts<-fromBIMETStoXTS(ts)
print(is.xts(xts))#TRUE
print(xts)

#create yearly time series with first date on period
setBIMETSconf('BIMETS_CONF_DIP','FIRST')
ts<-TSERIES(1:10,START=c(2000,1),FREQ='A')
print(is.xts(ts))#FALSE
#convert to xts
xts=fromBIMETStoXTS(ts)
print(is.xts(xts))#TRUE
print(xts)#dates on Jan 1

#reset default
setBIMETSconf('BIMETS_CONF_DIP','LAST')
```

`fromTStoXTS` *Convert TS to XTS*

Description

This function transforms a BIMETS compliant `ts` time series (as defined in `is.bimets`) into a time series of class `xts()`.
The core XTS function as.xts() does not satisfy all the compliance control check requirements, so it has been extended. If the output time series has an `.indexClass` of type `Date()`, i.e. neither monthly nor quarterly, the output dates are chosen accordingly to the BIMETS option `BIMETS_CONF_DIP`: if this option is set to `LAST` (default), the output `xts()` time series will have the date of the period set equal to the last day in the same period, e.g. 31 December for yearly time series, 30 June for semiannual, etc.; if BIMETS option `BIMETS_CONF_DIP` is set to `FIRST`, the output `xts()` time series will have the date of the period set equal to the first day in the same period, e.g. 1 January for yearly time series, 1 July for semiannual time series on the second period, etc.
In the case of quarterly time series the `.indexClass=yearqtr`;
in the case of monthly time series the `.indexClass=yearmon`.
Attributes and description of the input time series will be copied to the output time series (see `TIMESERIES`)

Usage

```
fromTStoXTS(x = NULL, avoidCompliance = FALSE, ...)
```

Arguments

x	Input ts time series that must satisfy the compliance control check defined in is.bimets.
avoidCompliance	If TRUE, compliance control check of input time series will be skipped. See is.bimets.
...	Backward compatibility.

Value

This function returns a time series of class xts() that has the same observations of the input ts time series.

See Also

fromXTStoTS
as.bimets
is.bimets
BIMETS indexing
BIMETS configuration

Examples

```
#day and month names can change depending on locale
Sys.setlocale('LC_ALL','C')
Sys.setlocale('LC_TIME','C')

#BIMETS_CONF_DIP default on LAST
print('yearly')
t<-ts(1:20,start=c(2005,2),frequency=1)
ts<-fromTStoXTS(t)
print(t);print(ts) #...dates on 31 Dec

print('semiannual')
t<-ts(1:20,start=c(2005,2),frequency=2)
ts<-fromTStoXTS(t)
print(t);print(ts) #...dates on 31 Dec/30 Jun

#set configuration BIMETS_CONF_DIP on FIRST
setBIMETSconf('BIMETS_CONF_DIP','FIRST')

print('yearly')
t<-ts(1:20,start=c(2005,2),frequency=1)
ts<-fromTStoXTS(t)
```

```
print(t);print(ts) #...dates on 1 Jan

print('semiannual')
t<-ts(1:20,start=c(2005,2),frequency=2)
ts<-fromTStoXTS(t)
print(t);print(ts) #...dates on 1 Jan/1 Jul

print('quarterly')
t<-ts(1:20,start=c(2004,3),frequency=4)
ts<-fromTStoXTS(t)
print(t);print(ts)

print('monthly')
t<-ts(1:20,start=c(2003,5),frequency=12)
ts<-fromTStoXTS(t)
print(t);print(ts)


print('daily')
t<-ts(1:20,start=c(2003,125),frequency=366)
ts<-fromTStoXTS(t)
print(t);print(ts)
```

`fromXTStoTS` *Convert XTS to TS*

Description

This function transforms a BIMETS compliant `xts()` time series (as defined in `is.bimets`) into a time series of class `ts()`. The core R function as.ts() does not satisfy all the compliance control check requirements, so it has been extended. Attributes and description of the input time series will be copied to the output time series (see `TIMESERIES`).

Usage

```
fromXTStoTS(x = NULL, avoidCompliance = FALSE, ...)
```

Arguments

`x`	Input `xts()` time series that must satisfy the compliance control check defined in `is.bimets`.
`avoidCompliance`	If `TRUE`, compliance control check of input time series will be skipped. See `is.bimets`.
`...`	Backward compatibility.

Value

This function returns a time series of class `ts()` that has the same observations of the input `xts()` time series.

See Also

fromTStoXTS
as.bimets
is.bimets
BIMETS indexing
BIMETS configuration

Examples

```
#day and month names can change depending on locale
Sys.setlocale('LC_ALL','C')
Sys.setlocale('LC_TIME','C')

#set configuration BIMETS_CONF_DIP on FIRST
setBIMETSconf('BIMETS_CONF_DIP','FIRST')

#set configuration BIMETS_CONF_CCT on XTS
setBIMETSconf('BIMETS_CONF_CCT','XTS')

#semiannual with Date()
n<-10
xArr<-rnorm(n)
dateArr<-seq(as.Date('2000/07/01'),by='6 months',length=n)
dataF<-data.frame(dateArr,xArr)
xt<-xts(dataF[,2],order.by=dataF[,1])
print(fromXTStoTS(xt))

#set configuration BIMETS_CONF_DIP on LAST
setBIMETSconf('BIMETS_CONF_DIP','LAST')

#yearly with Date()
n<-10
xArr<-rnorm(n)
dateArr<-seq(as.Date('2000/12/31'),by='year',length=n)
dataF<-data.frame(dateArr,xArr)
xt<-xts(dataF[,2],order.by=dataF[,1])
print(fromXTStoTS(xt))

#quarterly with yearqtr()
n<-10
xArr<-rnorm(n+1)
dateArr<-as.yearqtr('2000 Q2') + 0:n/4
dataF<-data.frame(dateArr,xArr)
xt<-xts(dataF[,2],order.by=dataF[,1])
```

```
print(fromXTStoTS(xt))

#monthly with yearmon()
n<-10
xArr<-rnorm(n+1)
dateArr<-as.yearmon('Jul 2000') + 0:n/12
dataF<-data.frame(dateArr,xArr)
xt<-xts(dataF[,2],order.by=dataF[,1])
print(fromXTStoTS(xt))

#daily with Date()
n<-10
xArr<-rnorm(n)
dateArr<-seq(as.Date('2000/07/14'),by='day',length=n)
dataF<-data.frame(dateArr,xArr)
xt<-xts(dataF[,2],order.by=dataF[,1])
print(fromXTStoTS(xt))

#restore defaults
setBIMETSconf('BIMETS_CONF_DIP','LAST')
setBIMETSconf('BIMETS_CONF_CCT','TS')
```

GETDATE *Retrieve Dates of Time Series*

Description

This function returns the date array of selected observations, in the requested print format. Dates will be provided accordingly to the BIMETS configuration option BIMETS_CONF_DIP
(see BIMETS configuration)

Usage

```
GETDATE(x=NULL, index=NULL, format='%Y-%m-%d', avoidCompliance=FALSE, ...)
```

Arguments

x	Input time series that must satisfy the compliance control check defined in is.bimets.
index	Index of observations to be selected. The output dates will be the dates of the selected observations. If index=NULL this function will retrieve all available dates in the input time series.
format	Output print format, provided as a paste of the following codes: **%Y**: 4 digits year **%y**: 2 digits year **%j**: period in the year for daily time series **%q**: quarter index, available only if also %y or %Y have been requested

%m: 2 digits month
%b: 3 digits month
%B: full month name
%d: 2 digits day
%a: 3 letters weekday
%A: full weekday name

`avoidCompliance`
: If `TRUE`, compliance control check of input time series will be skipped. See `is.bimets`

`...`
: Backward compatibility.

Value

This function returns the date array of selected observations, in the requested print format.

See Also

`BIMETS configuration`
`BIMETS indexing`
`yq2yp`
`ym2yp`
`date2yp`
`LOCS`
`NAMELIST`
`TSLOOK`
`TABIT`
`ELIMELS`

Examples

```
#day and month names can change depending on locale
Sys.setlocale('LC_ALL','C')
Sys.setlocale('LC_TIME','C')

#work on xts
setBIMETSconf('BIMETS_CONF_CCT','XTS')

#XTS yearly
n<-10
xArr<-(n:1)
dateArr<-seq(as.Date('2000-12-31'),by='year',length=n)
dataF<-data.frame(dateArr,xArr)
ts1<-xts(dataF[,2],order.by=dataF[,1])
ts1[5]<-NA
print(GETDATE(ts1,5)) #...print 2004-12-31
print(GETDATE(ts1,5,'%A %d %b %Y')) #print... Friday 31 Dec 2004
print(GETDATE(ts1)) #print... "2000-12-31" "2001-12-31" ... "2009-12-31"
```

```
#XTS quarterly
n<-15
xArr<-(n:0)
dateArr<-as.yearqtr('2000 Q1')+0:n/4
dataF<-data.frame(dateArr,xArr)
ts1<-xts(dataF[,2],order.by=dataF[,1])
print(GETDATE(ts1,9,'%b %Y')) #print...Mar 2002

#XTS monthly
#set configuration BIMETS_CONF_DIP to FIRST
setBIMETSconf('BIMETS_CONF_DIP','FIRST')
n<-15
xArr<-(n:0)
dateArr<-as.yearmon('Jan 2000')+0:n/12
dataF<-data.frame(dateArr,xArr)
ts1<-xts(dataF[,2],order.by=dataF[,1])
print(GETDATE(ts1,9,'%b %Y')) #print...Sep 2000

#set configuration BIMETS_CONF_DIP to LAST
setBIMETSconf('BIMETS_CONF_DIP','LAST')

#2000 is bissextile...
print(GETDATE(ts1,2)) #print... 2000-02-29

#quarter...
print(GETDATE(ts1,5,'%Y Q%q')) #print... 2000 Q2

#restore default
setBIMETSconf('BIMETS_CONF_CCT','TS')
```

GETRANGE *Time Series Common Range*

Description

Given a time series list, this function returns intersection or union of time series' ranges.

Usage

```
GETRANGE( x=list(),
          type='INNER',
          avoidCompliance=FALSE,
          ...)
```

Arguments

x	Input list, having elements as time series of class `ts` or `xts`. This argument can be also a single time series.

type — If type='INNER' then this funtion will return the intersection (if not NULL) of input time series' ranges. If type='OUTER' then this funtion will return the union of input time series' ranges.

avoidCompliance — If TRUE, compliance control check of input time series will be skipped. See is.bimets

... — Backward compatibility.

Value

This function returns the common range of the input time series as a 4-integer array built by c(START_Y,START_P,END_Y,END_P). If type='INNER' and there is no intersection of time series' ranges, then this function will return a NULL.

See Also

TSJOIN
TSEXTEND
TSMERGE
MOVAVG
GETYEARPERIOD
CUMSUM

Examples

```
#create ts
ts1=TSERIES((1:40), START=c(2000,1), FREQ=4)
ts2=TSERIES((1:40), START=c(2001,1), FREQ=4)
ts3=TSERIES((1:40), START=c(2002,1), FREQ=4)

myList=list(ts1,ts2,ts3)

print(GETRANGE(myList))
print(GETRANGE(myList,type='OUTER'))
```

GETYEARPERIOD *Get Time Series Year-Period*

Description

This function returns a two-element list (or a two-columns matrix in the case of JOIN=TRUE) built with of the years and the periods of the input time series observations. Users can provide the output list names.

Usage

```
GETYEARPERIOD(x=NULL, YEARS='YEAR', PERIODS='PRD', JOIN=FALSE, avoidCompliance=FALSE, ...)
TSDATES(x=NULL, YEARS='YEAR', PERIODS='PRD', JOIN=FALSE, avoidCompliance=FALSE, ...)
```

Arguments

x	Input time series, that must satisfy the compliance control check defined in is.bimets
YEARS	Argument of type string that will be the output list name for the array of observation years.
PERIODS	Argument of type string that will be the output list name for the array of observation periods.
JOIN	If TRUE, the output will be a matrix having each row built with the year and the period of the related observation.
avoidCompliance	If TRUE, compliance control check of input time series will be skipped. See is.bimets
...	Backward compatibility.

Value

This function returns an object of class `list()`. If JOIN=TRUE, this function returns a matrix.

See Also

NOELS
TSERIES
is.bimets
BIMETS indexing
TSLEAD
TSINFO
TSLOOK
TABIT
ELIMELS

Examples

```
#create quarterly ts
n<-20
ts1<-TSERIES((n:1),START=c(2000,1),FREQ=4)
myYP<-GETYEARPERIOD(ts1)
print(myYP$YEAR) #print 2000 2000 ...
print(myYP$PRD) #print 1 2 3 4 1 2 ...

#create monthly ts
ts1<-TSERIES((n:1),START=c(2000,1),FREQ='M')
```

```
myYP<-GETYEARPERIOD(ts1)
print(myYP$YEAR) #print 2000 2000 ...
print(myYP$PRD) #print 1 2 3 4 5 6 7 ...

#create yearly ts
ts1<-TSERIES((1:n),START=c(2000,1),FREQ=1)
myYP<-GETYEARPERIOD(ts1,YEARS='YEARSS', PERIODS='PRDSS')
print(myYP$YEARSS) #print 2000 2001 2002 ...
print(myYP$PRDSS) #print 1 1 1 1.....

#JOIN=TRUE
ts1<-TSERIES((n:1),START=c(2000,1),FREQ='M')
myYP<-GETYEARPERIOD(ts1,JOIN=TRUE)
print(myYP) #print 2000 2000 ...
#      [,1] [,2]
#[1,] 2000    1
#[2,] 2000    2
#[3,] 2000    3
#...
```

`idxOver` *BIMETS Time Series Indexing*

Description

Bimets package extends the way users can access and modify time series data.

SELECTING BY DATE: users can select a single observation by date by using the syntax `ts['Date']` and multiple observations by using `ts['StartDate/EndDate']` or
`ts['StartDate'+(0:n)/f]`, with `f=frequency`, `n=#observations`.
Data modification follows the same syntax:
`ts['Date']=value`, `ts['Date/Date']=c(value1,value2,...,valueN)`, etc. Users can also provide the string representing only the year of selection, or the year and the month of selection. For quarterly and monthly time series it is possible to select dates by using instances of class `yearmon()` and `yearqtr()` (See example).

SELECTING BY YEAR-PERIOD: Users can select observations by providing the year and the period requested. Selection and modification of data require the double square bracket syntax,
e.g. `ts[[YEAR,PERIOD]]=value`. Users can also assign an array of values to the input time series, starting from the `[[YEAR,PERIOD]]` provided,
i.e. `ts[[YEAR,PERIOD]]=c(value1,value2,...,valueN)`: in this case the input time series will be eventually extended in order to sequentially insert all values of the provided array (See example).

SELECTING BY INDICES: (core R) Users can select observations by simply providing the array of requested indices,
e.g. `ts[c(idx1,idx2,...,idxN)]` while reading and
`ts[c(idx1,idx2,...,idxN)]=c(value1,value2,...,valueN)` while modifying time series data.

See Also

GETDATE
BIMETS configuration
date2yp
yq2yp
ym2yp
as.bimets
is.bimets
LOCS
NAMELIST
TABIT
ELIMELS

Examples

```
  #day and month names can change depending on locale
  Sys.setlocale('LC_ALL','C')
  Sys.setlocale('LC_TIME','C')

#monthly
  #------------------------------
  print('MONTHLY GET by DATE')

  n<-25

  #create ts
  ts1<-TIMESERIES((0:n),START=c(2000,1),FREQ=12)

  print(ts1['2001-01']) #get Jan 2001
  print(ts1[as.yearmon('Jan 2001')]) #get Jan 2001
  print(ts1['2000-09/2001-01']) #get data from Sep 2000 to Jan 2001
  print(ts1['2000-09/']) #get data from Sep 2000
  print(ts1['/2001-01']) #get data till Jan 2001
  print(ts1['2001'])  #gat all data in year 2001

  #get 3 consecutive months starting from Jan 2001
  print(ts1[as.yearmon('Jan 2001')+ 0:2/12])


  print(ts1[c(2,4,5)]) #get observation number 2,4 and 5

  print('MONTHLY GET by YEAR-PERIOD')

  print(ts1[[2000,5]]) #get year 2000 period 5

  #get year 2010 period 1 (out of range)
  tryCatch({print(ts1[[2010,1]])},error=function(e){cat(e$message)})

  print(ts1[[2002,2]]) #get year 2002 period 2
```

```
print('MONTHLY SET by DATE')

ts1['2000-08']<-9.9 #assign to Aug 2000
ts1[as.yearmon('Feb 2001')]<-8.8 #assign to Feb 2001

#assign 8.8 on Feb 2001 and give warning
ts1[as.yearmon('Feb 2001')]=c(8.8,7.7)

#assign same value to all observation in range Sep 2000 - Jan 2001
ts1['2000-09/2001-01']<-11.11

#assign repeatedly the two values to each observation starting from Sep 2001
ts1['2001-09/']<-c(1.1,2.2)
print(ts1)

print('MONTHLY SET by YEAR-PERIOD')

ts1[[2000,5]]<-NA #set year 2000 period 5

#assign an array starting from year 2002 period 2 (extend time series)
ts1[[2002,2]]<-c(-1,-2,-3,-4,-5)
TABIT(ts1)


#quarterly
#------------------------------
print('QUARTERLY GET by DATE')

#create ts
ts1<-TSERIES((0:n),START=c(2000,1),FREQ=4)

print(ts1[as.yearqtr('2001 Q1')]) #get 2001 Q1
print(ts1['2001']) #get all data in year 2001

#get 4 consecutive quarters starting from 2002 Q2
print(ts1[as.yearqtr('2002 Q2')+ 0:3/4])

print(ts1['2003/']) #gat all data from 2003 Q1

print('QUARTERLY GET by YEAR-PERIOD')

print(ts1[[2002,4]]) #get year 2002 period 4

print('QUARTERLY SET by DATE')

ts1[as.yearqtr('2001 Q1')]<-7.7 #assign to 2001 Q1
ts1['2002']<-NA #assign to all observations of 2002

#assign to 3 quaters starting from 2003 Q2
ts1[as.yearqtr('2003 Q2')+ 0:2/4]<-0

ts1['2004/']<- -1 #assign to all observations starting from 2004
```

```
TABIT(ts1)

print('QUARTERLY SET by YEAR-PERIOD')

ts1[[2005,4]]<-c(1,2,3) #assign array starting from year 2005 period 4
TABIT(ts1)


#yearly
#------------------------------
print('YEARLY GET by DATE')

#create ts
ts1<-TSERIES((1:n),START=c(2000,1),FREQ=1)

print(ts1['2002-12-31']) #get 2002 data
print(ts1['2002'])  #get 2002 data
print(ts1['2000/2004'])  #get data from 2000 to 2004
print(ts1['2005/']) #get data starting from 2005

print('YEARLY GET by YEAR-PERIOD')

print(ts1[[2005,1]]) #get year 2005
#get year 2032 (out of range)
tryCatch({print(ts1[[2032,1]])},error=function(e){cat(e$message)})


print('YEARLY SET by DATE')

ts1['2004']<-NA #assign to 2004
ts1['2007/']<-0.0 #assign starting from 2007
ts1['2000/2002']<- -1 #assign in range 2000/2002
TABIT(ts1)

print('YEARLY SET by YEAR-PERIOD')

ts1[[2005,1]]<-NA #assign to 2005
ts1[[2014,1]]<- c(-1,-2,-3) #assign array starting from 2014 (extend series)
TABIT(ts1)

#daily
#------------------------------
print('DAILY GET by DATE')

#create ts
ts1<-TSERIES((1:n),START=c(2000,1),FREQ='D')

print(ts1['2000-01-12']) #get Jan 12, 2000 data

print('DAILY GET by YEAR-PERIOD')

print(ts1[[2000,14]]) #get year 2000 period 14
#get year 2032 (out of range)
```

```
tryCatch({print(ts1[[2032,1]]) },error=function(e){cat(e$message)})

print('DAILY SET by DATE')

ts1['2000-01-15']<-NA #assign to Jan 15, 2000
TABIT(ts1)

print('DAILY SET by YEAR-PERIOD')

ts1[[2000,3]]<-NA #assign to Jan 3, 2000

#assign array starting from 2000 period 35 (extend series)
ts1[[2000,35]]<- c(-1,-2,-3)
TABIT(ts1)
```

INDEXNUM *Rebase a Time Series*

Description

This function rebases an input time series to the value of 100 in the year selected by the BASEYEAR argument. If the input time series frequency is greater than one, the initial reference is set to the average value of the input time series observations that lie in the BASEYEAR.

Usage

```
INDEXNUM(x=NULL, BASEYEAR=NULL, avoidCompliance=FALSE, ...)
```

Arguments

x	Input time series that must satisfy the compliance control check defined in is.bimets.
BASEYEAR	Rebasing year.
avoidCompliance	If TRUE, compliance control check of input time series will be skipped. See is.bimets
...	Backward compatibility.

Value

This function returns a BIMETS time series.

See Also

TSJOIN
TSEXTEND
TSMERGE
MOVAVG
GETYEARPERIOD
CUMSUM

Examples

```
#create yearly ts
n<-20
ts1<-TSERIES(1:n,START=c(2000,1),FREQ=1)
TABIT(ts1, INDEXNUM(ts1,2005))

#quarterly
ts1<-TSERIES(1:n,START=c(2000,1),FREQ=4)
ts1[5]<-NA
TABIT(ts1, INDEXNUM(ts1,2000))
```

INTS *Create Range of Indices*

Description

A command such `INTS(i,j)` returns a one-dimensional array built of the integers `i`, `i+1`, `i+2`, `...`, `j` when `i`, `j` are both scalars, and `j` is greater than `i`. When `j` is less than `i`, the command shown above defines a one-dimensional array built of the integers `i`, `i-1`, `i-2`, `...`, `j`.

Users can specify the `k` increment using a syntax like `INTS(i, j, k)` which defines a one-dimensional array built of the values `i`, `i+k`, `i+2*k`, `...`, `i+N*k`.

The value of the last element of the array is the maximum value of `i+N*k` that is less than or equal to `j`, for positive `k`. For negative `k`, the value of the last element of the array is the minimum value of `i+N*k` that is greater than or equal to `j`.

The command can be used with one parameter by using a syntax like `INTS(i)` where `i` is a positive scalar. The result is a one-dimensional array built with the integers `1`, `2`, `3`, `...`, `i`. When `i` is less than `1`, the array is built with the integers `-1`, `-2`, `...`, `-i`.

Usage

```
INTS(FROM=NULL, TO=NULL, BY=NULL, ...)
```

Arguments

FROM	The first integer of the sequence. If arguments TO and BY are NULL and FROM>0 the sequence will start from 1 and will end in FROM; If arguments TO and BY are NULL and FROM<0 the sequence will start from -1 and will end in FROM (see example).
TO	The last integer of the sequence.
BY	The increment between two elements of the sequence.
...	Backward compatibility.

Value

This function returns an object of class c().

See Also

TSJOIN
TSEXTEND
TSMERGE
MOVAVG
GETYEARPERIOD
TSLAG
TSINFO
TABIT
ELIMELS

Examples

```
print(INTS(10,1,-2)) #... 10  8  6  4  2

#...Error in INTS(10, 1, -0.5) : INTS(): inputs must be integers.
tryCatch({print(INTS(10,1,-0.5));},error=function(e){cat(e$message)})

print(INTS(10))  #...  1  2  3  4  5  6  7  8  9 10
print(INTS(-10)) #  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10

# Error in INTS(0) : INTS(): magnitude must be >=1
tryCatch({print(INTS(0));},error=function(e){cat(e$message)})

print(INTS(-10,-45)) # -10 -11 -12 ... -41 -42 -43 -44 -45

#...Error in seq.default(FROM, TO, BY) : wrong sign in 'by' argument
tryCatch({print(INTS(-10,-45,3));},error=function(e){cat(e$message)})

print(INTS(-10,-45,-3)) # -10 -13 -16 -19 -22 -25 -28 -31 -34 -37 -40 -43
```

is.bimets *Check the Compliance of a Time Series*

Description

This function checks the compliance of the input time series that must verify the following BIMETS requirements:

- the input time series must be of the class defined in BIMETS_CONF_CCT (see BIMETS configuration)

- If BIMETS_CONF_CCT='TS' the input time series must be of class ts, univariate, with at least one observation and with a frequency f=1, 2, 3, 4, 12, 24, 36, 53 or 366 per year.

- if BIMETS_CONF_CCT='XTS' the input time series must be of class xts, univariate, with at least one observation and with a frequency f=1, 2, 3, 4, 12, 24, 36, 53 or 366 per year; the input time series must also be stricty regular, i.e. without any temporal discontinuity, and must have an .indexClass of type yearmon() for monthly time series, of type yearqtr() for quarterly time series and of type Date() for any other frequency. If configuration option BIMETS_CONF_DIP='LAST', i.e. the default value, the provided observation dates of the input xts() time series must be the last dates in the period, e.g. Dec. 31 for yearly time series, Jun. 30 for the first period in a semiannual time series, etc.; If configuration option BIMETS_CONF_DIP='FIRST' the provided observation dates of the input xts() time series must be the first dates in the period, e.g. Jan. 1 for an yearly time series, Jul. 1 for the second period in a semiannual time series, etc.;

BIMETS package functions return time series that are compliant to the above requirements.

The compliance check can be locally disabled by using the function argument avoidCompliance=TRUE, that is available in almost all package functions. The compliance check of a BIMETS generated time series can be avoided; moreover, disabling the control check can speed up the execution time, and is suggested when users concatenate several call to the package functions, e.g. the compliance check of the ts2 time series in the following example can be avoided:
ts2=TSLAG(ts1);ts3=TSDELTA(ts2,avoidCompliance=TRUE);.

Time series must lie in the year range 1800-2199: in this range the conversion between a date and the related year-period (and vice versa) has been hardcoded in order to speed up the code execution.

If the compliance check is disabled, i.e. avoidCompliance=TRUE and the input time series does not verify all the above requirements, the package functions can have an erroneous behavior. Should any doubt arise, we suggest to call the package functions using the default arguments; we also suggest to create time series object by using the command TIMESERIES.

Usage

```
is.bimets(x = NULL, suppressErrors=TRUE, ...)
```

Arguments

x	Input time series.
suppressErrors	If suppressErrors=TRUE the function returns a logical value TRUE/FALSE whenever the input time series is BIMETS compliant. If suppressErrors=FALSE the function will throw an error if the input time series is not BIMETS compliant.
...	Backward compatibility.

Value

This function returns a logical value TRUE/FALSE whenever the input time series is compliant to the above BIMETS requirements. If the test fails and suppressErrors=FALSE this function will throw an error.

See Also

as.bimets
TIMESERIES
BIMETS indexing
BIMETS configuration
fromBIMETStoTS
fromBIMETStoXTS

Examples

```
#day and month names can change depending on locale
Sys.setlocale('LC_ALL','C')
Sys.setlocale('LC_TIME','C')

#set day in period to last
setBIMETSconf('BIMETS_CONF_DIP','LAST')

#set constructor class type
setBIMETSconf('BIMETS_CONF_CCT','XTS')

#create an xts
xt<-TIMESERIES(1:10,START=c(2000,1),FREQ='A')

print(xt); #...dates are at Dec 31

print(is.bimets(xt)) #...TRUE

#change setting
setBIMETSconf('BIMETS_CONF_DIP','FIRST')

print(is.bimets(xt)) #...FALSE

#set constructor class type
setBIMETSconf('BIMETS_CONF_CCT','TS')
```

```
#bivariate ts
tsBiv<-ts(matrix(c(1,2,3,4,5,6),nrow=3,ncol=2),start=c(2000,1),frequency=1)

print(is.bimets(tsBiv)) #...FALSE

#...error
tryCatch({is.bimets(tsBiv,suppressError=FALSE)},
 error=function(e){cat(e$message)});try({is.bimets(tsBiv,suppressError=FALSE)})

#ts year
n<-10
xArr<-rnorm(n)
t<-ts(data=xArr,start=c(2000,1),frequency=1)
cat('is compliant?',is.bimets(t),'\n')

#ts semestral
n<-10
xArr<-rnorm(n)
t<-ts(data=xArr,start=c(2000,1),frequency=2)
cat('is compliant?',is.bimets(t),'\n')

#set configuration BIMETS_CONF_DIP on FIRST
setBIMETSconf('BIMETS_CONF_DIP','FIRST')

#work with XTS
setBIMETSconf('BIMETS_CONF_CCT','XTS')

#xts yearly with dates
n<-10
xArr<-rnorm(n)
dateArr<-seq(as.Date('2000/01/01'),by='year',length=n)
dataF<-data.frame(dateArr,xArr)
xt<-xts(dataF[,2],order.by=dataF[,1])
cat('is compliant?',is.bimets(xt),'\n')


#xts daily
n<-10
xArr<-rnorm(n)
dateArr<-seq(as.Date('2000/01/01'),by='day',length=n)
dataF<-data.frame(dateArr,xArr)
xt<-xts(dataF[,2],order.by=dataF[,1])
cat('is compliant?',is.bimets(xt),'\n')

#xts monthly with dates
n<-10
xArr<-rnorm(n)
dateArr<-seq(as.Date('2000/01/01'),by='month',length=n)
dataF<-data.frame(dateArr,xArr)
xt<-xts(dataF[,2],order.by=dataF[,1])
cat('monthly with dates is compliant? ',is.bimets(xt),'\n') #...false
```

```
#xts monthly with yearmon
n<-10
xArr<-rnorm(n+1)
dateArr<-as.yearmon('Jan 2001')+0:n/12
dataF<-data.frame(dateArr,xArr)
xt<-xts(dataF[,2],order.by=dataF[,1])
cat('monthly with yearmon is compliant? ',is.bimets(xt),'\n') #...true


#restore defaults
setBIMETSconf('BIMETS_CONF_CCT','TS')
setBIMETSconf('BIMETS_CONF_DIP','LAST')
```

LOAD_MODEL *Load a BIMETS model description file*

Description

This function parses a MDL model definition and creates an equivalent R data structure that can be estimated and simulated. The input model definition can be either an external plain text file or a character variable.

Usage

```
LOAD_MODEL( modelFile=NULL,
            modelText=NULL,
            quietly=FALSE,
            oldStyleModel=FALSE,
            ...)
```

Arguments

modelFile	The path to the text file containing the MDL model definition.
modelText	The character variable containing the MDL model definition. modelText takes precedence over modelFile if both are defined.
quietly	If TRUE, information messages will be suppressed.
oldStyleModel	Backward compatibility.
...	Backward compatibility.

Value

This function returns a BIMETS model object containing all the information gathered the input model definition's parsing.

A BIMETS model created with the LOAD_MODEL function can be viewed as a complex R list()

containing the following elements (see example):

- **rawData** and **cleanModel**: string arrays containing the original model definition. `cleanModel` is a clean version of the model definition, i.e. without comments, blank lines, etc.;

- **behaviorals** and **identities**: sub-lists containing all the information gathered from the behavioral and the identity definitions. This sub lists are described later in this page;

- **vendog** and **vexog**: string array containing the names of the endogenous and exogenous variables of the model; the former is also subsetted into **vendogBehaviorals** and **vendogIdentities**

- **totNumEqs**, **totNumIds** and **eqCoeffNum**: integer variables containing the behaviorals count, the identities count and the coefficients count of the model;

- **incidence_matrix**: the incidence matrix built from the model equations; it is a square matrix in which each row and each column represent an endogenous variable. If the `(i,j)` element is equal to `1` than in the model definition the current value of the endogenous variable referred by the `i`-row directly depends on the current value of the endogenous variable referred by the `j`-column. (see example)

- **vpre**, **vsim**, **vfeed** and **vpost**: the simulation process takes advantage of an appropriate ordering of the equations to increase the performances by iteratively solving only one subset of equations, while the other equations are solved straightforwardly. (Ref: *Don Gallo - Solving large sparse systems of equations in econometric models - Journal of Forecasting 1987* and *Numerical methods for simulation and optimal control of large-scale macroeconomic models - Nepomiastchy, Rachidi, Ravelli - 1980*). The optimal reordering of the model equations is achieved by using an iterative algorithm applied to the incidence matrix, that produces 4 ordered arrays of endogenous variables:
1. `vpre` is the ordered list containing the names of the endogenous pre-recursive variables to be sequentially computed (once per simulation period, by using their `EQ>` definition in the `MDL`) before the simulation iterative algorithm takes place;
2. `vsim` (the simultaneous block) is the ordered list containing the names of the endogenous variables to be sequentially computed during each iteration of the simulation iterative algorithm;
3. `vfeed` is the list containing the names of the endogenous feedback variables; generally `vfeed` are the last variables of the ordered `vsim` list;
4. `vpost` is the ordered list containing the names of the endogenous post-recursive variables to be sequentially computed (once per simulation period, by using their `EQ>` definition in the `MDL`) after the simulation iterative algorithm has found a solution in the simultaenous block (more details in `SIMULATE` help pages);

- **max_lag**: the max lag of the model, i.e. the highest number of periods a time series of the model is lagged by in the `MDL` definition. It also accounts for recursive lagging
(e.g. `TSLAG(...TSLAG(...)...)`), PDLs and for the order of the error autocorrelation, if any;

- **modelName**: the name of the model, copied from the input file name or from the input `character` variable name containing the model definition;

BEHAVIORALS and IDENTITIES

The elements **'behaviorals'** and **'identities'** of the BIMETS model are named lists that contain information on behaviorals and identities of the model. In both of this two lists, the name of each element is the name of the behavioral or the identity the data refer to, as specified in the model definition file: e.g. given a BIMETS model named `myModel`, the information on a behavioral named `cn` (i.e there exists a "BEHAVIORAL> cn" in the `MDL` definition of the model) is stored into `myModel$behaviorals$cn`.

Behavioral elements have the following components:

- **eq**: the equation of the behavioral, as a `character` variable;

- **eqCoefficientsNames**: the names of the coefficients (the original ones and eventually the ones created by the `PDL>` expansion);

- **eqCoefficientsNamesOriginal**: the names of the original coefficients;

- **eqComponentsNames**: the names of endogenous and exogenous variables that appear in the behavioral equation;

- **eqComponentsNamesBehaviorals**: the names of behavioral endogenous variables that appear in the behavioral equation;

- **eqComponentsNamesIdentities**: the names of identity endogenous variables that appear in the behavioral equation;

- **eqComponentsNamesExogenous**: the names of exogenous variables that appear in the behavioral equation;

- **tsrange**: the estimation time range as a 4 integer array;

- **eqRegressorsNames**: a `character` array containing the regressor expressions (the original ones and eventually the ones created by the `PDL>` expansion);

- **eqRegressorsNamesOriginal**: a `character` array containing the expressions of the original regressors;

- **errorRaw**: the original definition of the error autocorrelation, if any (see `MDL`);

- **errorType**: the type of the error structure;

- **errorDim**: the dimension of the error autocorrelation;

- **eqSimExp**: the `R` optimized `expression` of the behavioral equation that is used in the simulation algorithm;

- **matrixR**: the `R` Lagrange matrix that is used in restriction analysis (see `MDL`);

- **vectorR**: the `r` Lagrange vector that is used in restriction analysis (see `MDL`);

- **restrictRaw**: the original definition of the coefficient restrictions, if any.

- **pdlRaw**: the original definition of the PDL restrictions, if any (see example and MDL).

- **pdlRestrictionMatrix**: the R Lagrange matrix that is used in PDL restriction analysis (see example and MDL);

- **IVComponentsNames**: the names of endogenous and exogenous variables that appear in the instrumental variables equations, if any;

- **iv**: the original definitions of instrumental variables, if any.

For example, given a BIMETS model named myModel, the information on a technical identity named y (i.e there exists an "IDENTITY> y" in the MDL definition of the model) is stored in myModel$identities$y.

Identity elements have the following components:

- **eqRaw**: the original equations of the identity (more than one if the identity has multiple equations and has IF> conditions), as a character variable (see example and MDL);

- **ifRaw**: the original IF> conditions, if any, of the identity, as a character variable;

- **eqFull**: the full expression of the identity, composed with IF> conditions and related equations (see example), as a character variable;

- **eqComponentsNames**: the names of endogenous and exogenous variables that appear in the identity equation;

- **eqComponentsNamesBehaviorals**: the names of behavioral endogenous variables that appear in the identity equation;

- **eqComponentsNamesIdentities**: the names of identity endogenous variables that appear in the identity equation;

- **eqSimExp**: the R optimized expression of the identity equation that is used in the simulation algorithm;

- **hasIF**: boolean, TRUE if the identity has an IF> condition;

See Also

MDL
LOAD_MODEL_DATA
ESTIMATE
SIMULATE

STOCHSIMULATE
MULTMATRIX
RENORM
TIMESERIES
BIMETS indexing
BIMETS configuration

Examples

```
#define model
myModelDefinition<-
"MODEL

COMMENT> Modified Klein Model 1 of the U.S. Economy with PDL,
COMMENT> autocorrelation on errors, restrictions and conditional evaluations

COMMENT> Consumption
BEHAVIORAL> cn
TSRANGE 1925 1 1941 1
EQ> cn = a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)
COEFF> a1 a2 a3 a4
ERROR> AUTO(2)

COMMENT> Investment
BEHAVIORAL> i
TSRANGE 1923 1 1941 1
EQ> i = b1 + b2*p + b3*TSLAG(p,1) + b4*TSLAG(k,1)
COEFF> b1 b2 b3 b4
RESTRICT> b2 + b3 = 1

COMMENT> Demand for Labor
BEHAVIORAL> w1
TSRANGE 1925 1 1941 1
EQ> w1 = c1 + c2*(y+t-w2) + c3*TSLAG(y+t-w2,1) + c4*time
COEFF> c1 c2 c3 c4
PDL> c3 1 3

COMMENT> Gross National Product
IDENTITY> y
EQ> y = cn + i + g - t

COMMENT> Profits
IDENTITY> p
EQ> p = y - (w1+w2)

COMMENT> Capital Stock with switches
IDENTITY> k
EQ> k = TSLAG(k,1) + i
IF> i > 0
IDENTITY> k
EQ> k = TSLAG(k,1)
```

```
IF> i <= 0
END"

#load model
myModel<-LOAD_MODEL(modelText=myModelDefinition)

#retrieve model structure...
#get definition
myModel$cleanModel
# [1] "BEHAVIORAL> cn"
# [2] "TSRANGE 1925 1 1941 1"
# [3] "EQ> cn = a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)"
# [4] "COEFF> a1 a2 a3 a4"
# [5] "ERROR> AUTO(2)"
# [6] "BEHAVIORAL> i"
# [7] "TSRANGE 1923 1 1941 1"
# [8] "EQ> i = b1 + b2*p + b3*TSLAG(p,1) + b4*TSLAG(k,1)"
# [9] "COEFF> b1 b2 b3 b4"
#[10] "RESTRICT> b2 + b3 = 1"
#[11] "BEHAVIORAL> w1"
#[12] "TSRANGE 1925 1 1941 1"
#[13] "EQ> w1 = c1 + c2*(y+t-w2) + c3*TSLAG(y+t-w2,1)+c4*time"
#[14] "COEFF> c1 c2 c3 c4"
#[15] "PDL> c3 1 3"
#[16] "IDENTITY> y"
#[17] "EQ> y = cn + i + g - t"
#[18] "IDENTITY> p"
#[19] "EQ> p = y - (w1+w2)"
#[20] "IDENTITY> k"
#[21] "EQ> k = TSLAG(k,1) + i"
#[22] "IF> i > 0"
#[23] "IDENTITY> k"
#[24] "EQ> k = TSLAG(k,1)"
#[25] "IF> i <= 0"

#get endogenous and exogenous
myModel$vendog
#[1] "cn" "i"  "w1" "y"  "p"  "k"
myModel$vexog
#[1] "w2"   "t"    "time" "g"

#get behaviorals, identities and coefficients count
myModel$totNumEqs
#[1] 3
myModel$totNumIds
#[1] 3
myModel$eqCoeffNum
#[1] 12

#get the incidence matrix
myModel$incidence_matrix
#   cn i w1 y p k
#cn  0 0  1 0 1 0
```

```
#i   0 0  0 0 1 0
#w1  0 0  0 1 0 0
#y   1 1  0 0 0 0
#p   0 0  1 1 0 0
#k   0 1  0 0 0 0

#get the optimal reordering arrays
myModel$vpre
#NULL
myModel$vsim
#[1] "w1" "p"  "cn" "i"  "y"
myModel$vfeed
#[1] "y"
myModel$vpost
#[1] "k"

#get the model max lag and the model name
myModel$max_lag
#[1] 3
myModel$modelName
#myModelDefinition



#get infos on behavioral w1

myModel$behaviorals$w1$eq
#[1] "w1=c1+c2*(y+t-w2)+c3*TSLAG(y+t-w2,1)+c4*time"

myModel$behaviorals$w1$eqCoefficientsNames
#[1] "c1"       "c2"       "c3"       "c3_PDL_1" "c3_PDL_2" "c4"

myModel$behaviorals$w1$eqCoefficientsNamesOriginal
#[1] "c1" "c2" "c3" "c4"

myModel$behaviorals$w1$eqComponentsNames
#[1] "t"    "time" "w1"   "w2"   "y"

myModel$behaviorals$w1$tsrange
#[1] 1925    1 1941    1

myModel$behaviorals$w1$eqRegressorsNames
#[1] "1"                       "(y+t-w2)"
#[3] "TSLAG(y+t-w2,1)"  "TSLAG(TSLAG(y+t-w2,1),1)" "TSLAG(TSLAG(y+t-w2,1),2)" "time"

myModel$behaviorals$w1$eqRegressorsNamesOriginal
#[1] "1"                "(y+t-w2)"
#[3] "TSLAG(y+t-w2,1)" "time"

myModel$behaviorals$w1$pdlRaw
#[1] "c3 1 3;"

myModel$behaviorals$w1$pdlRestrictionMatrix
```

```
#     [,1] [,2] [,3] [,4] [,5] [,6]
#[1,]    0    0    1   -2    1    0



#get infos on behavioral cn

myModel$behaviorals$cn$errorRaw
#[1] "AUTO(2)"

myModel$behaviorals$cn$errorType
#[1] "AUTO"

myModel$behaviorals$cn$errorDim
#[1] 2

myModel$behaviorals$cn$eqSimExp
#expression(cn[4,]=cn__ADDFACTOR[4,]+cn__a1+cn__a2*p[4,]+cn__a3*(p[3,])+
#cn__a4*(w1[4,]+w2[4,])+cn__RHO_1*(cn[3,]-(cn__ADDFACTOR[3,]+
#cn__a1+cn__a2*p[3,]+cn__a3*(p[2,])+cn__a4*(w1[3,]+w2[3,])))+
#cn__RHO_2*(cn[2,]-(cn__ADDFACTOR[2,]+cn__a1+cn__a2*p[2,]+
#cn__a3*(p[1,])+cn__a4*(w1[2,]+w2[2,]))))



#get infos on behavioral i

myModel$behaviorals$i$matrixR
#    [,1] [,2] [,3] [,4]
#[1,]   0    1    1    0

myModel$behaviorals$i$vectorR
#[1] 1

myModel$behaviorals$i$restrictRaw
#[1] "b2+b3=1;"



#get infos on identitiy k

myModel$identities$k$eqRaw
#[1] "k=TSLAG(k,1)+i;k=TSLAG(k,1);"

myModel$identities$k$ifRaw
#[1] "i > 0;i <= 0;"

myModel$identities$k$eqFull
#[1] "__IF__ (i > 0) __THEN__ k=TSLAG(k,1)+i;__IF__ (i <= 0) __THEN__ k=TSLAG(k,1);"

myModel$identities$k$eqComponentsNames
#[1] "i" "k"
```

```
myModel$identities$k$eqSimExp
#expression(k[4,]=.MODEL_VIF(k[4,],i[4,] > 0,k_ADDFACTOR[4,]+
#(k[3,])+i[4,]),k[4,]=.MODEL_VIF(k[4,],i[4,] <= 0,
#k_ADDFACTOR[4,]+(k[3,])))

myModel$identities$k$hasIF
#[1] TRUE
```

LOAD_MODEL_DATA *Load time series data into a BIMETS model*

Description

This function verifies the input time series list and copies the data into a BIMETS model object. Provided time series must be BIMETS compliant, as defined in `is.bimets`

Usage

```
LOAD_MODEL_DATA(model=NULL, modelData=NULL, quietly=FALSE, ...)
```

Arguments

`model`	The BIMETS model object (see `LOAD_MODEL`).
`modelData`	The input time series list containing endogenous and exogenous data (see example).
`quietly`	If `TRUE`, information messages will be suppressed.
`...`	Backward compatibility.

Value

This function add a new named element, i.e. `modelData`, into the output model object.

The new `modelData` element is a named list that contains all the input time series. Each element name of this list is set equal to the name of the endogenous or exogenous variable the time series data refer to.

See Also

`MDL`
`LOAD_MODEL`
`ESTIMATE`
`SIMULATE`
`STOCHSIMULATE`

MULTMATRIX
RENORM
TIMESERIES
BIMETS indexing
BIMETS configuration

Examples

```
#define model data
myModelData<-list(
  cn
  =TIMESERIES(39.8,41.9,45,49.2,50.6,52.6,55.1,56.2,57.3,57.8,55,50.9,
              45.6,46.5,48.7,51.3,57.7,58.7,57.5,61.6,65,69.7,
              START=c(1920,1),FREQ=1),
  g
  =TIMESERIES(4.6,6.6,6.1,5.7,6.6,6.5,6.6,7.6,7.9,8.1,9.4,10.7,10.2,9.3,10,
              10.5,10.3,11,13,14.4,15.4,22.3,
              START=c(1920,1),FREQ=1),
  i
  =TIMESERIES(2.7,-.2,1.9,5.2,3,5.1,5.6,4.2,3,5.1,1,-3.4,-6.2,-5.1,-3,-1.3,
              2.1,2,-1.9,1.3,3.3,4.9,
              START=c(1920,1),FREQ=1),
  k
  =TIMESERIES(182.8,182.6,184.5,189.7,192.7,197.8,203.4,207.6,210.6,215.7,
              216.7,213.3,207.1,202,199,197.7,199.8,201.8,199.9,
              201.2,204.5,209.4,
              START=c(1920,1),FREQ=1),
  p
  =TIMESERIES(12.7,12.4,16.9,18.4,19.4,20.1,19.6,19.8,21.1,21.7,15.6,11.4,
              7,11.2,12.3,14,17.6,17.3,15.3,19,21.1,23.5,
              START=c(1920,1),FREQ=1),
  w1
  =TIMESERIES(28.8,25.5,29.3,34.1,33.9,35.4,37.4,37.9,39.2,41.3,37.9,34.5,
              29,28.5,30.6,33.2,36.8,41,38.2,41.6,45,53.3,
              START=c(1920,1),FREQ=1),
  y
  =TIMESERIES(43.7,40.6,49.1,55.4,56.4,58.7,60.3,61.3,64,67,57.7,50.7,41.3,
              45.3,48.9,53.3,61.8,65,61.2,68.4,74.1,85.3,
              START=c(1920,1),FREQ=1),
  t
  =TIMESERIES(3.4,7.7,3.9,4.7,3.8,5.5,7,6.7,4.2,4,7.7,7.5,8.3,5.4,6.8,7.2,
              8.3,6.7,7.4,8.9,9.6,11.6,
              START=c(1920,1),FREQ=1),
  time
  =TIMESERIES(NA,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,
              START=c(1920,1),FREQ=1),
  w2
  =TIMESERIES(2.2,2.7,2.9,2.9,3.1,3.2,3.3,3.6,3.7,4,4.2,4.8,5.3,5.6,6,6.1,
              7.4,6.7,7.7,7.8,8,8.5,
              START=c(1920,1),FREQ=1)
)
```

```
#define model
myModelDefinition<-
"MODEL

COMMENT> Modified Klein Model 1 of the U.S. Economy with PDL,
COMMENT> autocorrelation on errors, restrictions and conditional evaluations

COMMENT> Consumption
BEHAVIORAL> cn
TSRANGE 1925 1 1941 1
EQ> cn = a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)
COEFF> a1 a2 a3 a4
ERROR> AUTO(2)

COMMENT> Investment
BEHAVIORAL> i
TSRANGE 1923 1 1941 1
EQ> i = b1 + b2*p + b3*TSLAG(p,1) + b4*TSLAG(k,1)
COEFF> b1 b2 b3 b4
RESTRICT> b2 + b3 = 1

COMMENT> Demand for Labor
BEHAVIORAL> w1
TSRANGE 1925 1 1941 1
EQ> w1 = c1 + c2*(y+t-w2) + c3*TSLAG(y+t-w2,1)+c4*time
COEFF> c1 c2 c3 c4
PDL> c3 1 3

COMMENT> Gross National Product
IDENTITY> y
EQ> y = cn + i + g - t
COMMENT> Profits
IDENTITY> p
EQ> p = y - (w1+w2)

COMMENT> Capital Stock with switches
IDENTITY> k
EQ> k = TSLAG(k,1) + i
IF> i > 0
IDENTITY> k
EQ> k = TSLAG(k,1)
IF> i <= 0

END"

#load model
myModel<-LOAD_MODEL(modelText=myModelDefinition)

#load data into the model
myModel<-LOAD_MODEL_DATA(myModel,myModelData,showWarnings = TRUE)
#Load model data "myModelData" into model "myModelDefinition"...
#CHECK_MODEL_DATA(): warning, there are missing values in series "time".
```

```
#...LOAD MODEL DATA OK


#retrieve data from model object

myModel$modelData$cn
#Time Series:
#Start = 1920
#End = 1941
#Frequency = 1
# [1] 39.8 41.9 45.0 49.2 50.6 52.6 55.1 56.2 57.3
#57.8 55.0 50.9 45.6 46.5 48.7 51.3 57.7 58.7 57.5 61.6
#[21] 65.0 69.7

myModel$modelData$w1
#Time Series:
#Start = 1920
#End = 1941
#Frequency = 1
# [1] 28.8 25.5 29.3 34.1 33.9 35.4 37.4 37.9 39.2
#41.3 37.9 34.5 29.0 28.5 30.6 33.2 36.8 41.0 38.2 41.6
#[21] 45.0 53.3

myModel$modelData$i
#Time Series:
#Start = 1920
#End = 1941
#Frequency = 1
# [1]  2.7 -0.2  1.9  5.2  3.0  5.1  5.6  4.2  3.0  5.1
#1.0 -3.4 -6.2 -5.1 -3.0 -1.3  2.1  2.0 -1.9  1.3
#[21]  3.3  4.9

myModel$modelData$time
#Time Series:
#Start = 1920
#End = 1941
#Frequency = 1
# [1]  NA -10  -9  -8  -7  -6  -5  -4  -3  -2  -1
#0   1   2   3   4   5   6   7   8   9  10
```

LOCS *Select Time Series Indices*

Description

This function returns the indices of the input TRUE elements. The input can be either an array or a time series. The result is usually used as a structured index to produce a new array.

Usage

```
LOCS(x=NULL, options='ALL', ...)
```

Arguments

x	This function accepts as input a boolean array or a boolean time series, often as the result of a logic comparison between an expression and a numerical array or a numerical time series: e.g. `LOCS(c(a,b,c,...)>=k)`; `LOCS(ts==j)`; `LOCS(ts<expr)`; `LOCS(is.na(ts))`; `etc...`
options	A selection option can refine the result: **ALL**: (default) all the `TRUE` indices will be returned in the output. **UNIQUE**: return the index of the unique `TRUE` result; if there are multiple `TRUE` results then an error will be thrown. **FIRST**: return the first `TRUE` result. **LAST**: return the last `TRUE` result.
...	Backward compatibility.

Value

This function returns a numerical array built with the indices of the values that are `TRUE` in the input boolean array or in the input boolean time series.

See Also

NOELS
is.bimets
BIMETS indexing
TSERIES
GETYEARPERIOD
NOELS
NAMELIST
INTS
TSINFO
TABIT
ELIMELS

Examples

```
#create ts
n<-10
ts1<-TSERIES((1:n),START=c(2000,1),FREQ=1)
print(LOCS(ts1>7,options='FIRST')) #print 8

#generate error: print LOCS(): input has more than one TRUE element.
tryCatch({print(LOCS(ts1>=3,options='UNIQUE'));},error=function(e){print(e$message);})

print(LOCS(is.na(c(1,2,NA,4,5,6,7,NA,NA)))) #print c(3,8,9)
```

MDL *BIMETS Model Description Language*

Description

BIMETS provides a language to unambiguously specify an econometric model. This page describes how to create a model and its general structure. The specification of an econometric model is translated and identified by keyword statements which are grouped in a model file, i.e. a plain text file or a `character` variable with a specific syntax. Collectively, these keyword statements constitute the BIMETS Model Description Language (from now on `MDL`). The model specifications consist of groups of statements. Each statement begins with a keyword. The keyword classifies the component of the model which is being specified.

Below is an example of a Klein's model with an `MDL` compliant syntax which can either be stored in a `character` variable or in a plain text file.

The content of the *klein1.txt* variable is:

```
R> klein1.txt="
MODEL

COMMENT> Consumption
BEHAVIORAL> cn
TSRANGE 1921 1 1941 1
EQ> cn =  a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)
COEFF> a1 a2 a3 a4

COMMENT> Investment
BEHAVIORAL> i
TSRANGE 1921 1 1941 1
EQ> i = b1 + b2*p + b3*TSLAG(p,1) + b4*TSLAG(k,1)
COEFF> b1 b2 b3 b4

COMMENT> Demand for Labor
BEHAVIORAL> w1
TSRANGE 1921 1 1941 1
EQ> w1 = c1 + c2*(y+t-w2) + c3*TSLAG(y+t-w2,1)+c4*time
COEFF> c1 c2 c3 c4

COMMENT> Gross National Product
IDENTITY> y
EQ> y = cn + i + g - t

COMMENT> Profits
IDENTITY> p
EQ> p = y - (w1+w2)
```

```
COMMENT> Capital Stock
IDENTITY> k
EQ> k = TSLAG(k,1) + i

END
"
```

Please note that there are circular dependencies between equations of the model, e.g. `p <- w1 <- y <- p` as shown in the "BIMETS package" section figure in the pdf version of the manual. Circular dependencies imply that the model simulation must be solved with an iterative algorithm.

As shown, the model definition is quite intuitive. The first keyword is `MODEL`, while at the end of the model definition we can find the `END` keyword. Available tags in the definition of a generic BIMETS model are:

- **EQUATION>** or **BEHAVIORAL>** indicate the beginning of a series of keyword statements describing a behavioral equation. The behavioral statement general form is:
`BEHAVIORAL> name [TSRANGE startYear, startPeriod, endYear, endPeriod]`
where `name` is the name of the behavioral equation and the optional `TSRANGE` specifies that the provided time interval must be used to estimate the coefficients. The optional `TSRANGE` is defined as a 4-dimensional numerical array built with starting year, starting period, ending year, and ending period.

Given $Y = \beta * X + \epsilon$, where Y are the historical values of the dependent variable and X are the historical values of the regressors, if the requested estimation method is `OLS` (Ordinary Least Squares), in the general case (i.e. no restrictions nor error auto-correlation, as described later) the coefficients will be calculated as: $\beta_{OLS} = (X' * X)^{-1} * X' * Y$.

If the requested estimation method is `IV` (Instrumental Variables), given Z the matrix built with instrumental variables as columns Z_i, that should not be correlated to the disturbance terms, i.e. $E[\epsilon' * Z_i] = 0$, the coefficients will be either calculated as
$\beta_{IV} = (Z' * X)^{-1} * Z' * Y$, or more generally as: $\beta_{IV} = (\hat{X}' * \Omega^{-1} * \hat{X})^{-1} * \hat{X}' * \Omega^{-1} * Y$ where $\hat{X} = Z * (Z' * Z)^{-1} * Z' * X$ and $\Omega = \sigma^2 * I$, $\sigma^2 = E[\epsilon' * \epsilon]$

- **IDENTITY>** indicates the beginning of a series of keyword statements describing an identity or technical equation. The identity statement general form is:
`IDENTITY> name`
where `name` is the identity name.

- **EQ>** specifies the mathematical expression for a behavioral or an identity equation.

The equation statement general form for a behavioral equation is:
`EQ> LHS = coeff1*f1 + coeff2*f2 + coeff3*f3 + ...`
where `LHS` is a function of the behavioral variable,
`coeff1, coeff2, coeff3, ...` are the coefficient names of the equation and
`f1, f2, f3, ...` are functions of variables.

The equation statement general form for an identity equation is:
`EQ> LHS = f1 + f2 + f3 + ...`

where LHS is a function of the identity variable and
f1, f2, f3, ... are functions of variables.

The following MDL functions can be used in the LHS left-hand side of the equation, with name as the name of the behavioral or the identity variable:

- name - i.e. the identity function;

- TSDELTA(name,i) - i-periods difference of the name time series;

- TSDELTAP(name,i) - i-periods percentage difference of the name time series;

- TSDELTALOG(name,i) - i-periods logarithmic difference of the name time series;

- LOG(name) - log of the name time series;

- EXP(name) - exponential of the name time series.

On the other side, the mathematical expression available for use in the RHS right-hand side of the EQ> equation and in the IV> expression described later in this page (i.e. f1, f2, f3, ...) can include the standard arithmetic operators, parentheses and the following MDL functions:

- TSLAG(ts,i) - lag the ts time series by i-periods;

- TSDELTA(ts,i) - i-periods difference of the ts time series;

- TSDELTAP(ts,i) - i-periods percentage difference of the ts time series;

- TSDELTALOG(ts,i) - i-periods logarithmic difference of the ts time series;

- MOVAVG(ts,i) - i-periods moving average of the ts time series;

- MOVSUM(ts,i) - i-periods moving sum of the ts time series;

- LOG(ts) - log of the ts time series;

- EXP(ts) - exponential of the ts time series;

- ABS(ts) - absolute values of the ts time series.

MDL function names are reserved names. They cannot be used as variable or coefficient names. The coefficient names are specified in a subsequent COEFF> keyword statement within a behavioral equation. By definition, identities do not have any coefficient that must be assessed. Any name not specified as a coefficient name or mentioned on the list of the available MDL functions is assumed to be a variable.

- **COEFF>** specifies the coefficient names used in the EQ> keyword statement of a behavioral equation. The coefficients statement general form is:

COEFF> coeff0 coeff1 coeff2 ... coeffn.
The coefficients order in this statement must be the same as it appears in the behavioral equation.

- **ERROR>** specifies an autoregressive process of a given order for the regression error. The error statement general form is:
ERROR> AUTO(n)
where n is the order of the autoregressive process for the error.

During an estimation, users must ensure that the required data are available for the specified error structure: n periods of data before the time interval specified by TSRANGE must be defined in any time series involved in the regression.

The solution requires an iterative algorithm. Given $Y_1 = \beta_1 * X_1 + \epsilon_1$, where Y_1 are the historical values of the dependent variable and X_1 are the historical values of the regressors, the iterative algorithm is based on the Cochrane-Orcutt procedure:

1) Make an initial estimation by using the original TSRANGE extended backward n periods (given n as the autocorrelation order).

2) Estimate the error autocorrelation coefficients $\rho_i = \rho_{i,1}, ..., \rho_{i,n}$ with $i = 1$ by regressing the residuals ϵ_i on their lagged values by using the auxiliary model:
$\epsilon_i = \rho_{i,1} * TSLAG(\epsilon_i, 1) + ... + \rho_{i,n} * TSLAG(\epsilon_i, n)$

3) Transform the data for the dependent and the independent variables by using the estimated ρ_i. The new dependent variable will be: $Y_{i+1} = P_i * Y_i$, and the new independent variables will be $X_{i+1} = P_i * X_i$ with the matrix P_i defined as:

$$P_i = \begin{pmatrix} 1 & 0 & 0 & 0 & ... & 0 & 0 \\ -\rho_{i,1} & 1 & 0 & 0 & ... & 0 & 0 \\ -\rho_{i,2} & -\rho_{i,1} & 1 & 0 & ... & 0 & 0 \\ & & & ... & & & \\ 0 & 0 & ... & -\rho_{i,n} & ... & -\rho_{i,1} & 1 \end{pmatrix}$$

4) Run another estimation on the original model $Y_{i+1} = \beta_{i+1} * X_{i+1} + \epsilon_{i+1}$ by using the suitable TSRANGE and the transformed data coming out of step 3 and compute the new time series for the residuals.

5) Estimate the new autocorrelation coefficients $\rho_{i+1} = \rho_{i+1,1}, ..., \rho_{i+1,n}$, by regressing the new residuals arising from step 4 (similar to step 2)

6) Carry out the convergence check through a comparison among the previous ρ_i and the new ones arising from steps 5.
If $all(abs(\rho_{i+1} - \rho_i) < \delta)$, where ρ_i is the ρ vector at the iteration i and δ is a small convergence factor, then exit otherwise repeat from step 3 with i <- i+1.

- **RESTRICT>** is a keyword that can be used to specify linear coefficient restrictions. A deterministic restriction can be applied to any equation coefficient. Any number of RESTRICT> keywords is allowed for each behavioral equation.

A deterministic (exact) coefficient restriction sets a linear expression containing one or more coefficients equal to a constant. The restriction only affects the coefficients of the behavioral equation in which it is specified. The restriction statement general form is:

```
RESTRICT> linear_combination_of_coefficients_1 = value_1
...
linear_combination_of_coefficients_n = value_n
```

where `linear_combination_of_coefficients_i, i=1..n` is a linear combination of the coefficient(s) to be restricted and `value_i` is the in-place scalar value to which the linear combination of the coefficients is set equal. Each linear combination can be set equal to a different value.

`MDL` example:

```
RESTRICT> coeff1 = 0
coeff2 = 10.5
coeff3-3*coeff4+1.2*coeff5 = 0
```

In many econometric packages, linear restrictions have to be coded by hand in the equations. BIMETS allows users to write down the restriction in a natural way, thus applying a constrained minimization. This procedure, although it leads to approximate numerical estimates, allows an easy implementation.

The theory behind this procedure is that of the Lagrange multipliers. Presented here is an example of its implementation.

Suppose that we have an equation defined as:

```
EQUATION> Y TSRANGE 2010 1 2015 4
EQ> Y = C1*X1 + C2*X2 + C3*X3
COEFF> C1 C2 C3
RESTRICT> 1.1*C1 + 1.3*C3 = 2.1
1.2*C2 = 0.8
```

Coefficients `C1, C2, C3` are to be estimated. They are subject to the linear constraints specified by the `RESTRICT>` keyword statement. In the case of `OLS` estimation, this is carried out in the following steps:

1) Compute the cross-product matrices $X'X$ and $X'Y$ where X is a matrix with dimension `[NOBS x NREG]` containing the values of the independent variables (regressors) historical observations (and a vector of ones for the constant term, if any), and where Y is a `NOBS` elements vector of the dependent variable (regressand) historical observations; `NOBS` is the number of observations available on the `TSRANGE` specified in the behavioral equation, and `NREG` is the number of regressors, or coefficients;

2) Build the restriction matrices. In the example:

$$R = \begin{pmatrix} 1.1 & 0 & 1.3 \\ 0 & 1.2 & 0 \end{pmatrix}$$

and

$$r = \begin{pmatrix} 2.1 \\ 0.8 \end{pmatrix}$$

R is a matrix of [NRES x NREG] size, and r is a vector of [NRES] length, where NRES is the number of restrictions;

3) Compute the scaling factors for the augmentation to be performed in the next step:

$$Rscale[i] = \frac{mean(X'X)}{max(abs(R[i,]))}$$

where $R[i,]$ is the i-th row of the R matrix.

Assuming $mean(X'X) = 5000$, in the example above we will have:
$Rscale[1] = 5000/1.3$
$Rscale[2] = 5000/1.2$

The augmented matrices will then be defined as:

$$R_{aug} = \begin{pmatrix} 1.1 * Rscale[1] & 0 & 1.3 * Rscale[1] \\ 0 & 1.2 * Rscale[2] & 0 \end{pmatrix}$$

and

$$r_{aug} = \begin{pmatrix} 2.1 * Rscale[1] \\ 0.8 * Rscale[2] \end{pmatrix}$$

4) Compute the so-called "augmented" cross-product matrix $(X'X)_{aug}$ by adding to the cross-product matrix $(X'X)$ a total of NRES rows and NRES columns:

$$(X'X)_{aug} = \begin{pmatrix} X'X & R'_{aug} \\ R_{aug} & 0 \end{pmatrix}$$

5) Similarly, compute the so-called "augmented" cross-product matrix $(X'Y)_{aug}$ by adding a total of NRES elements to the cross-product matrix $(X'Y)$:

$$(X'Y)_{aug} = \begin{pmatrix} X'Y \\ r_{aug} \end{pmatrix}$$

6) Calculate the $\hat{\beta}_{aug}$ augmented coefficients by regressing the $(X'Y)_{aug}$ on the $(X'X)_{aug}$.

The first NREG values of the augmented coefficients $\hat{\beta}_{aug}$ array are the estimated coefficients with requested restrictions. The last NRES values are the errors we have on the deterministic restrictions.

In the case of IV estimation, the procedure is the same as in the OLS case, but the matrix X has to be replaced with the matrix $\hat{X}$, as previously defined in the BEHAVIORAL> keyword.

- **PDL>** is a keyword that defines an Almon polynomial distributed lag to be used in estimation. Almon Polynomial distributed lags are specific kind of deterministic restrictions imposed on the

coefficients of the distributed lags of a specific regressor. Multiple PDLs on a single behavioral equation can be defined.

The PDL> statement general form is:
`PDL> coeffname degree laglength [N] [F]`
where `coeffname` is the name of a coefficient, `degree` is an integer scalar specifying the degree of the polynomial, `laglength` is an integer scalar specifying the length of the polynomial (in number of time periods), the optional `N` (i.e. "nearest") means that the nearest lagged term of the expansion, i.e., the first term, is restricted to zero, and the optional `F` (i.e. "farthest") means that the farthest lagged term of the expansion, i.e., the last term, is restricted to zero; the `PDL>` keyword statement thusly defined applies an Almon polynomial distributed lag to the regressor associated with the `coeffname` coefficient, of `laglength` length and `degree` degree, by providing the appropriate expansion and the deterministic restrictions for the degree and length specified. These expansions are not explicitly shown to the user, i.e., the original model is not changed.

`laglength` must be greater than `degree` (see example below).

A PDL term can be further referenced in a `RESTRICT>` keyword statement by using the following syntax: `LAG(coefname, pdllag)`.

Example: `RESTRICT> LAG(coeff2,3) = 0` means that, during the estimation, the regressor related to the coefficient `coeff2` and lagged by 3 periods in the PDL expansion must have a coefficient equal to zero. This example also implies that a `PDL> coeff2 x y` with `y > 3` has been declared in the same behavioral.

The implementing rules are the following:

1) Read off the `laglength` of the PDL keyword and expand the column of the regressor related to `coeffname` in the matrix `X` (i.e. the original regressors matrix) with the lagged values of the regressor, from left to right, starting form the lag 1 to the lag `laglength-1`. The matrix `X` will now have a `[NOBS x (NREG+laglength-1)]` size, with `NOBS` as the number of observations in the specified `TSRANGE` and `NREG` as the number of regressors, or coefficients.

2) Build the restriction matrix `R` with the following `[ Nrow x Ncol ]` dimensions:
`Nrow = laglength - ( degree + 1 )`
`Ncol = NREG + laglength - 1`

This matrix's elements will be zero except for the (`laglength`)-columns related to the section of the expanded columns in the `X` matrix. For every row we will have to insert `degree+2` numbers different from zero.

The `degree+2` numbers are taken form the Tartaglia's-like triangle:

```
1 -2  1
1 -3  3 -1
1 -4  6 -4   1
1 -5 10 -10  5  1
... ... ... ...
```

where in the i-th row we find the numbers for a PDL of degree=i.

The r vector giving the knows terms for the restrictions is a vector of NRES = laglength - (degree + 1) elements equal to zero.

An example will clarify:

```
EQUATION> Y TSRANGE 2010 1 2015 4
EQ> Y = C1*X1 + C2*X2 + C3*X3
COEFF> C1 C2 C3
PDL> C2 2 5
```

then

$$R = \begin{pmatrix} 0 & 1 & -3 & 3 & 1 & 0 & 0 \\ 0 & 0 & 1 & -3 & 3 & 1 & 0 \end{pmatrix}$$

and

$$r = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

The expanded regressors are:
X1, X2, TSLAG(X2,1), TSLAG(X2,2), TSLAG(X2,3), TSLAG(X2,4), X3.

The scaling factor is given, as in the standard restriction case, by: $mean(X'X)/max(abs(R[i,]))$

- **IF>** keyword is used to conditionally evaluate an identity during a simulation, depending on a logical expression's value. Thus, it is possible to have a model alternating between two or more identity specifications for each simulation period, depending upon results from other equations.

The IF> statement general form is:
IF> logical_expression

The IF> keyword must be specified within an identity group; this keyword causes the equation specified in the identity group to be evaluated during the current simulation period only when the logical_expression is TRUE.

Only one IF> keyword is allowed in an identity group. Further occurrences produce an error message, and processing stops.

The logical_expression can be composed of constants, endogenous variables, exogenous variables, an expression among variables, combinations of the logical operators; mathematical operators and the MDL functions listed in the EQ> section are allowed.

In the following MDL example, the value of the endogenous myIdentity variable is specified with two complementary conditional identities, depending on the TSDELTA() result:

```
IDENTITY> myIdentity
```

```
IF> TSDELTA(myEndog*(1-myExog)) > 0
EQ> myIdentity = TSLAG(myIdentity)+1

IDENTITY> myIdentity
IF> TSDELTA(myEndog*(1-myExog)) <= 0
EQ> myIdentity = TSLAG(myIdentity)
```

- **IV>** specifies the mathematical expression for an instrumental variable used in a behavioral equation.

The general form for an instrumental variable expression is:
`IV> f1 + f2 + f3 + ...`
`f1, f2, f3, ...` are functions of variables.

The mathematical expression available for use in the `IV>` definition are those already described in the `EQ>` section.

- **COMMENT>** can be used to insert comments into a model. The general form of this keyword is:
`COMMENT> text`

The `text` following the `COMMENT>` keyword is ignored during all processing and must lie in the same line. Comments cannot be inserted within another keyword statement. A dollar sign in the first position of a line is equivalent to using the COMMENT> keyword, as in this exmaple:
`$This is a comment`

No other keywords are currently allowed in the `MDL` syntax.

See Also

LOAD_MODEL
ESTIMATE
SIMULATE
STOCHSIMULATE
MULTMATRIX
RENORM
OPTIMIZE
TIMESERIES
BIMETS indexing
BIMETS configuration
summary

Examples

```
############################################################
#KLEIN MODEL WITH AUTOCORRELATION, RESTRICTIONS AND
```

```
#CONDITIONAL EVALUATIONS

#define model
myModel<-
"MODEL

COMMENT> Modified Klein Model 1 of the U.S. Economy with PDL,
COMMENT> autocorrelation on errors, restrictions and conditional evaluations

COMMENT> Consumption with autocorrelation on errors
BEHAVIORAL> cn
TSRANGE 1925 1 1941 1
EQ> cn =  a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)
COEFF> a1 a2 a3 a4
ERROR> AUTO(2)

COMMENT> Investment with restrictions
BEHAVIORAL> i
TSRANGE 1923 1 1941 1
EQ> i = b1 + b2*p + b3*TSLAG(p,1) + b4*TSLAG(k,1)
COEFF> b1 b2 b3 b4
RESTRICT> b2 + b3 = 1

COMMENT> Demand for Labor with PDL
BEHAVIORAL> w1
TSRANGE 1925 1 1941 1
EQ> w1 = c1 + c2*(y+t-w2) + c3*TSLAG(y+t-w2,1)+c4*time
COEFF> c1 c2 c3 c4
PDL> c3 1 2

COMMENT> Gross National Product
IDENTITY> y
EQ> y = cn + i + g - t

COMMENT> Profits
IDENTITY> p
EQ> p = y - (w1+w2)

COMMENT> Capital Stock with switches
IDENTITY> k
EQ> k = TSLAG(k,1) + i
IF> i > 0
IDENTITY> k
EQ> k = TSLAG(k,1)
IF> i <= 0

END"

#define model data
modelData<-list(
  cn    =TIMESERIES(39.8,41.9,45,49.2,50.6,52.6,55.1,56.2,57.3,57.8,55,50.9,
           45.6,46.5,48.7,51.3,57.7,58.7,57.5,61.6,65,69.7,
           START=c(1920,1),FREQ=1),
```

```
  g     =TIMESERIES(4.6,6.6,6.1,5.7,6.6,6.5,6.6,7.6,7.9,8.1,9.4,10.7,10.2,9.3,10,
            10.5,10.3,11,13,14.4,15.4,22.3,
            START=c(1920,1),FREQ=1),
  i     =TIMESERIES(2.7,-.2,1.9,5.2,3,5.1,5.6,4.2,3,5.1,1,-3.4,-6.2,-5.1,-3,-1.3,
            2.1,2,-1.9,1.3,3.3,4.9,
            START=c(1920,1),FREQ=1),
  k     =TIMESERIES(182.8,182.6,184.5,189.7,192.7,197.8,203.4,207.6,210.6,215.7,
            216.7,213.3,207.1,202,199,197.7,199.8,201.8,199.9,
            201.2,204.5,209.4,
            START=c(1920,1),FREQ=1),
  p     =TIMESERIES(12.7,12.4,16.9,18.4,19.4,20.1,19.6,19.8,21.1,21.7,15.6,11.4,
            7,11.2,12.3,14,17.6,17.3,15.3,19,21.1,23.5,
            START=c(1920,1),FREQ=1),
  w1    =TIMESERIES(28.8,25.5,29.3,34.1,33.9,35.4,37.4,37.9,39.2,41.3,37.9,34.5,
            29,28.5,30.6,33.2,36.8,41,38.2,41.6,45,53.3,
            START=c(1920,1),FREQ=1),
  y     =TIMESERIES(43.7,40.6,49.1,55.4,56.4,58.7,60.3,61.3,64,67,57.7,50.7,41.3,
            45.3,48.9,53.3,61.8,65,61.2,68.4,74.1,85.3,
            START=c(1920,1),FREQ=1),
  t     =TIMESERIES(3.4,7.7,3.9,4.7,3.8,5.5,7,6.7,4.2,4,7.7,7.5,8.3,5.4,6.8,7.2,
            8.3,6.7,7.4,8.9,9.6,11.6,
            START=c(1920,1),FREQ=1),
  time  =TIMESERIES(NA,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,
            START=c(1920,1),FREQ=1),
  w2    =TIMESERIES(2.2,2.7,2.9,2.9,3.1,3.2,3.3,3.6,3.7,4,4.2,4.8,5.3,5.6,6,6.1,
            7.4,6.7,7.7,7.8,8,8.5,
            START=c(1920,1),FREQ=1)
)

#load model and model data
model<-LOAD_MODEL(modelText=myModel)
model<-LOAD_MODEL_DATA(model,modelData)

#estimate model
model<-ESTIMATE(model)

#simulate model
model<-SIMULATE(model
               ,TSRANGE=c(1923,1,1941,1)
               ,simConvergence=0.00001
               ,simIterLimit=100
)


############################################################
#KLEIN MODEL WITH LHS FUNCTIONS


#define the model with LHS funs
myModel<-'MODEL

COMMENT> Modified Klein Model 1 of the U.S. Economy with PDL,
COMMENT> autocorrelation on errors, restrictions and conditional evaluations
```

```
COMMENT> LHS functions on EQ

COMMENT> Exp Consumption
BEHAVIORAL> cn
TSRANGE 1925 1 1941 1
EQ> EXP(cn) = a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)
COEFF> a1 a2 a3 a4
ERROR> AUTO(2)

COMMENT> Log Investment
BEHAVIORAL> i
TSRANGE 1925 1 1941 1
EQ> LOG(i) = b1 + b2*p + b3*TSLAG(p,1) + b4*TSLAG(k,1)
COEFF> b1 b2 b3 b4
RESTRICT> b2 + b3 = 1

COMMENT> Demand for Labor
BEHAVIORAL> w1
TSRANGE 1925 1 1941 1
EQ> w1 = c1 + c2*(TSDELTA(y)+t-w2) + c3*TSLAG(TSDELTA(y)+t-w2,1)+c4*time
COEFF> c1 c2 c3 c4
PDL> c3 1 3

COMMENT> Delta Gross National Product
IDENTITY> y
EQ> TSDELTA(y) = EXP(cn) + LOG(i) + g - t

COMMENT> Profits
IDENTITY> p
EQ> p = TSDELTA(y) - (w1+w2)

COMMENT> Capital Stock with switches
IDENTITY> k
EQ> k = TSLAG(k,1) + LOG(i)
IF> LOG(i) > 0
IDENTITY> k
EQ> k = TSLAG(k,1)
IF> LOG(i) <= 0

END'


#define model data
modelData<-list(
  cn=TSERIES(39.8,41.9,45,49.2,50.6,52.6,55.1,56.2,57.3,
             57.8,55,50.9,45.6,46.5,48.7,51.3,57.7,58.7,57.5,61.6,65,69.7,
             START=c(1920,1),FREQ=1),
  g=TSERIES(4.6,6.6,6.1,5.7,6.6,6.5,6.6,7.6,7.9,8.1,9.4,
             10.7,10.2,9.3,10,10.5,10.3,11,13,14.4,15.4,22.3,
             START=c(1920,1),FREQ=1),
  i=TSERIES(2.7,-.2,1.9,5.2,3,5.1,5.6,4.2,3,5.1,1,-3.4,
             -6.2,-5.1,-3,-1.3,2.1,2,-1.9,1.3,3.3,4.9,
             START=c(1920,1),FREQ=1),
```

```
  k=TSERIES(182.8,182.6,184.5,189.7,192.7,197.8,203.4,
            207.6,210.6,215.7,216.7,213.3,207.1,202,
            199,197.7,199.8,201.8,199.9,201.2,204.5,209.4,
            START=c(1920,1),FREQ=1),
  p=TSERIES(12.7,12.4,16.9,18.4,19.4,20.1,19.6,19.8,21.1,
            21.7,15.6,11.4,7,11.2,12.3,14,17.6,17.3,15.3,19,21.1,23.5,
            START=c(1920,1),FREQ=1),
  w1=TSERIES(28.8,25.5,29.3,34.1,33.9,35.4,37.4,37.9,39.2,
            41.3,37.9,34.5,29,28.5,30.6,33.2,36.8,41,38.2,41.6,45,53.3,
            START=c(1920,1),FREQ=1),
  y=TSERIES(43.7,40.6,49.1,55.4,56.4,58.7,60.3,61.3,64,67,
            57.7,50.7,41.3,45.3,48.9,53.3,61.8,65,61.2,68.4,74.1,85.3,
            START=c(1920,1),FREQ=1),
  t=TSERIES(3.4,7.7,3.9,4.7,3.8,5.5,7,6.7,4.2,4,7.7,7.5,
            8.3,5.4,6.8,7.2,8.3,6.7,7.4,8.9,9.6,11.6,
            START=c(1920,1),FREQ=1),
  time=TSERIES(NA,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,
            3,4,5,6,7,8,9,10,
            START=c(1920,1),FREQ=1),
  w2=TSERIES(2.2,2.7,2.9,2.9,3.1,3.2,3.3,3.6,3.7,4,4.2,
            4.8,5.3,5.6,6,6.1,7.4,6.7,7.7,7.8,8,8.5,
            START=c(1920,1),FREQ=1)
)


#example data transformation
modelData<-within(modelData,{
  i=exp(i);     #we have LOG(i)     in the model MDL definition
  cn=log(cn);   #we have EXP(cn)    in the model MDL definition
  y=CUMSUM(y)   #we have TSDELTA(y) in the model MDL definition
})

#load model and model data
model<-LOAD_MODEL(modelText=myModel)
model<-LOAD_MODEL_DATA(model,modelData)

#estimate model
model<-ESTIMATE(model)

#simulate model
model<-SIMULATE(model
               ,TSRANGE=c(1925,1,1930,1)
               ,simConvergence=0.00001
               ,simIterLimit=100
)

############################################################
#SIMPLE MODEL WITH IV


#define the model with IVs
myShortModelDefinition<-"
MODEL
```

```
COMMENT> Consumption with IV
BEHAVIORAL> cn
TSRANGE 1925 1 1941 1
EQ> cn = a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)
COEFF> a1 a2 a3 a4
IV> 1
IV> TSLAG(y)
IV> TSLAG(w1)*pi+0.5
IV> exp(w2)
END
"

#load model
myShortModel<-LOAD_MODEL(modelText=myShortModelDefinition)
```

MONTHLY *Monthly Time Series (Dis)Aggregation*

Description

This function returns a monthly (dis)aggregated time series, by using as input an annual, semiannual, quarterly or daily time series.

Usage

```
MONTHLY(x = NULL, fun = NULL, avoidCompliance = FALSE, ...)
```

Arguments

x	Input time series that must satisfy the compliance control check defined in is.bimets.
fun	Only for daily input time series: **STOCK**: the value of the input time series in the last observation of a month is assigned to the same month of the output time series. **NSTOCK**: the value of the input time series in the last non-missing observation of a month is assigned to the same month of the output time series. **SUM**: the sum of input observations in a month is assigned to the same month of the output time series. **NSUM**: the sum of input non-missing observations in a month is assigned to the same month of the output time series. **AVE**: the average of input observations in a month is assigned to the same month of the output time series. **NAVE**: the average of input non-missing observations in a month is assigned to the same month of the output time series. Only for quarterly, semiannual or annual input time series: **NULL**: (default) the output value of each monthly observation is set equal to the value of the input observation the month belongs to (i.e. duplicated values over

the period)
INTERP_END: the value of the input time series in a period is copied into the last month of the output time series that lies in the same period. Other values are calculated by linear interpolation.
INTERP_CENTER: the value of the input time series in a period is copied into the median month of the output time series that lies in the same period. Other values are calculated by linear interpolation.
INTERP_BEGIN: the value of the input time series in a period is copied into the first month of the output time series that lies in the same period. Other values are calculated by linear interpolation.

`avoidCompliance`
: If `TRUE`, compliance control check of input time series will be skipped. See `is.bimets`

`...`
: Backward compatibility.

Value

This function returns a monthly BIMETS time series.

See Also

`YEARLY`
`SEMIANNUAL`
`QUARTERLY`
`DAILY`

Examples

```
#TS FREQ 2 SEMIANNUAL TO MONTHLY
ts1<-TSERIES((1:10),START=c(2000,1),FREQ=2)
TABIT(MONTHLY(ts1,fun='INTERP_CENTER'))

#TS DAILY TO MONTHLY
ts1<-TSERIES((1:366),START=c(2000,1),FREQ='D')
TABIT(MONTHLY(ts1,fun='STOCK'))
```

`MOVAVG` *Moving Average*

Description

This function returns the moving average of the elements of the input array or the input time series. The result is an object of the same class of the input, and its elements are the moving average of length L of the input values. If the input is a time series, the DIRECTION of the moving average, i.e backward, forward or centered, can be provided. MAVE is an alias for MOVAVG

Usage

```
MOVAVG(x = NULL, L = NULL, DIRECTION = NULL, avoidCompliance = FALSE, ...)
MAVE(x = NULL, L = NULL, DIRECTION = NULL, avoidCompliance = FALSE, ...)
```

Arguments

x	Input numerical array or time series that must satisfy the compliance control check defined in is.bimets.
L	Length of the mean. Must be a positive integer.
DIRECTION	if x is a time series, given y as output and x as input: **AHEAD**: the output observation value in index n will be y[n] = mean(x[n],x[n+1],...,x[n+L-1]). **CENTER**: the output observation value in index n will be y[n] = mean(x[n-trunc(L/2)],...,x[n],x[n+1],...,x[n+trunc(L/2)]). **NULL** o **BACK**: (default) the output observation value in index n will be y[n] = mean(x[n+1-L],...,x[n-1],x[n]).
avoidCompliance	If TRUE, compliance control check of input time series will be skipped. See is.bimets
...	Backward compatibility.

Value

This function returns an object of the same class of the input, i.e. an array or a BIMETS time series.

See Also

TSDELTA
TSLAG
TSPROJECT
TSEXTEND
TSLEAD
CUMSUM
INDEXNUM
VERIFY_MAGNITUDE
GETRANGE

Examples

```
#input data
inputArray<-c(1,2,3,4,NA,1,2,3,4,5)

#array lag 3
out_movavg<-MOVAVG(inputArray,3)
print(out_movavg)

#ts lag 4 centered with missings
ts1<-TSERIES(inputArray,START=c(2000,1),FREQ='A')
out_movavg<-MAVE(ts1,4,'CENTER')
TABIT(out_movavg)

#ts daily
ts1<-TSERIES(inputArray,START=c(2000,1),FREQ='D')
out_movavg<-MAVE(ts1,3)
TABIT(ts1,out_movavg)
```

MOVTOT *Moving Sum*

Description

This function returns the moving sum of the elements of the input array or the input time series. The result is an object of the same class of the input, and its elements are the moving sum of length L of the input values. If the input is a time series, the DIRECTION of the moving sum, i.e backward, forward or centered, can be provided. MTOT and MSUM are alias for MOVTOT and MOVSUM

Usage

```
MOVSUM(x = NULL, L = NULL, DIRECTION = NULL, avoidCompliance = FALSE, ...)
MOVTOT(x = NULL, L = NULL, DIRECTION = NULL, avoidCompliance = FALSE, ...)
```

Arguments

x	Input numerical array or time series that must satisfy the compliance control check defined in is.bimets.
L	Length of the sum. It must be a positive integer.
DIRECTION	if x is a time series, given y as output and x as input: **AHEAD**: the output observation value in index n will be y[n] = sum(x[n],x[n+1],...,x[n+L-1]). **CENTER**: the output observation value in index n will be y[n] = sum(x[n-trunc(L/2)],...x[n],x[n+1],...,x[n+trunc(L/2)]). **NULL** o **BACK**: (default) the output observation value in index n will be y[n] = sum(x[n+1-L],...,x[n-1],x[n]).

avoidCompliance
: If TRUE, compliance control check of input time series will be skipped. See is.bimets

...
: Backward compatibility.

Value

This function returns an object of the same class of the input, i.e. an array or a BIMETS time series.

See Also

TSDELTA
TSLAG
TSPROJECT
TSEXTEND
TSLEAD
CUMSUM
INDEXNUM
VERIFY_MAGNITUDE
GETRANGE

Examples

```
#input data
inputArray<-c(1,2,3,4,NA,1,2,3,4,5)

#array lag 3
out_movtot<-MOVSUM(inputArray,3)
print(out_movtot)

#ts lag 4 centered with missings
ts1<-TSERIES(inputArray,START=c(2000,1),FREQ='A')
out_movtot<-MOVSUM(ts1,4,'CENTER')
TABIT(out_movtot)

#ts daily
ts1<-TSERIES(inputArray,START=c(2000,1),FREQ='D')
out_movtot<-MSUM(ts1,3)
TABIT(ts1,out_movtot)
```

MULTMATRIX *Compute the multiplier matrix of a BIMETS model*

Description

This function computes the matrix of both impact and interim multipliers, for a selected set of endogenous variables (i.e. TARGET) with respect to a selected set of exogenous variables (i.e. INSTRUMENT), by subtracting the results from different simulations in each period of the provided time range (i.e. TSRANGE). The simulation algorithms are the same as those used for the SIMULATE operation.

The MULTMATRIX procedure is articulated as follows:

1- simultaneous simulations are done;

2- the first simulation establishes the base line solution (without shocks);

3- the other simulations are done with shocks applied to each of the INSTRUMENT one at a time for every period in TSRANGE;

4- each simulation follows the defaults described in the SIMULATE help page, but has to be STATIC for the IMPACT multipliers and DYNAMIC for INTERIM multipliers;

5- given MM_SHOCK shock amount as a very small positive number, derivatives are computed by subtracting the base line solution of the TARGET from the shocked solution, then dividing by the value of the base line INSTRUMENT time the MM_SHOCK.

The IMPACT multipliers measure the effects of impulse exogenous changes on the endogenous variables in the same time period. They can be defined as partial derivatives of each current endogenous variable with respect to each current exogenous variable, all other exogenous variables being kept constant.

Given $Y(t)$ an endogenous variable at time t and $X(t)$ an exogenous variable at time t the impact multiplier $m(Y, X, t)$ is defined as $m(Y, X, t) = \partial Y(t)/\partial X(t)$ and can be approximated by $m(Y, X, t) \approx (Y_{shocked}(t) - Y(t))/(X_{shocked}(t) - X(t))$, with $Y_{shocked}(t)$ the values fo the simulated endogenous variable Y at time t when $X(t)$ is shocked to $X_{shocked}(t) = X(t)(1 + MM\_SHOCK)$

The INTERIM or delay-r multipliers measure the delay-r effects of impulse exogenous changes on the endogenous variables in the same time period. The delay-r multipliers of the endogenous variable Y with respect to the exogenous variable X related to a dynamic simulation from time t to time t+r can be defined as the partial derivative of the current endogenous variable Y at time t+r with respect to the exogenous variable X at time t, all other exogenous variables being kept constant.

Given $Y(t + r)$ an endogenous variable at time $t + r$ and $X(t)$ an exogenous variable at time t the impact interim or delay-r multiplier $m(Y, X, t, r)$ is defined as $m(Y, X, t, r) = \partial Y(t + r)/\partial X(t)$ and can be approximated by $m(Y, X, t, r) \approx (Y_{shocked}(t + r) - Y(t + r))/(X_{shocked}(t) - X(t))$, with $Y_{shocked}(t + r)$ the values fo the simulated endogenous variable Y at time $t + r$ when $X(t)$ is shocked to $X_{shocked}(t) = X(t)(1 + MM\_SHOCK)$

Users can also declare an endogenous variable as the INSTRUMENT variable. In this case, the constant adjustment (see SIMULATE) related to the provided endogenous variable will be used as the INSTRUMENT exogenous variable (see example);

Usage

```
MULTMATRIX(model=NULL,
           simAlgo='GAUSS-SEIDEL',
           TSRANGE=NULL,
           simType='DYNAMIC',
           simConvergence=0.01,
           simIterLimit=100,
           ZeroErrorAC=FALSE,
           Exogenize=NULL,
           ConstantAdjustment=NULL,
           verbose=FALSE,
           verboseSincePeriod=0,
           verboseVars=NULL,
           TARGET=NULL,
           INSTRUMENT=NULL,
           MM_SHOCK=0.00001,
           quietly=FALSE,
           JACOBIAN_SHOCK=1e-4,
           JacobianDrop=NULL,
           avoidCompliance=FALSE,
           ...)
```

Arguments

model	see SIMULATE
simAlgo	see SIMULATE
TSRANGE	see SIMULATE
simType	see SIMULATE
simConvergence	see SIMULATE
simIterLimit	see SIMULATE
ZeroErrorAC	see SIMULATE
Exogenize	see SIMULATE
ConstantAdjustment	see SIMULATE
verbose	see SIMULATE
verboseSincePeriod	see SIMULATE
verboseVars	see SIMULATE
TARGET	A character array built with the names of the endogenous variables for which the multipliers are requested

INSTRUMENT	A character array built with the names of the exogenous variables with respect to which the multipliers are evaluated. Users can also declare an endogenous variable as INSTRUMENT variable: in this case the constant adjustment (see SIMULATE) related to the provided endogenous variable will be used as the instrument exogenous variable
MM_SHOCK	The value of the shock added to INSTRUMENT variables in the derivative calculation of the multipliers. The default value is 0.00001 times the value of the exogenous variable
quietly	see SIMULATE
JACOBIAN_SHOCK	see SIMULATE
JacobianDrop	see SIMULATE
avoidCompliance	see SIMULATE
...	see SIMULATE

Value

This function will add a new element named MultiplierMatrix into the output BIMETS model object.

The new MultiplierMatrix element is a
(NumPeriods * Nendogenous) X (NumPeriods * Nexogenous) matrix,
with NumPeriods as the number of periods specified in the TSRANGE, Nendogeous the count of the endogenous variables in the TARGET array and Nexogenous the count of the exogenous variables in the INSTRUMENT array.

The arguments passed to the function call during the latest MULTMATRIX run will be inserted into the '__SIM_PARAMETERS__' element of the model simulation list (see SIMULATE); this data can be helpful in order to replicate the multiplier matrix results.

Row and column names in the output multiplier matrix identify the variables and the periods involved in the derivative solution, with the syntax VARIABLE_PERIOD (see example).

See Also

MDL
LOAD_MODEL
ESTIMATE
SIMULATE
STOCHSIMULATE
RENORM
TIMESERIES
BIMETS indexing
BIMETS configuration

Examples

```
#define model
myModelDefinition<-
"MODEL
COMMENT> Klein Model 1 of the U.S. Economy

COMMENT> Consumption
BEHAVIORAL> cn
TSRANGE 1921 1 1941 1
EQ> cn =  a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)
COEFF> a1 a2 a3 a4

COMMENT> Investment
BEHAVIORAL> i
TSRANGE 1921 1 1941 1
EQ> i = b1 + b2*p + b3*TSLAG(p,1) + b4*TSLAG(k,1)
COEFF> b1 b2 b3 b4

COMMENT> Demand for Labor
BEHAVIORAL> w1
TSRANGE 1921 1 1941 1
EQ> w1 = c1 + c2*(y+t-w2) + c3*TSLAG(y+t-w2,1)+c4*time
COEFF> c1 c2 c3 c4

COMMENT> Gross National Product
IDENTITY> y
EQ> y = cn + i + g - t

COMMENT> Profits
IDENTITY> p
EQ> p = y - (w1+w2)

COMMENT> Capital Stock
IDENTITY> k
EQ> k = TSLAG(k,1) + i

END"

#define model data
myModelData<-list(
  cn
  =TIMESERIES(39.8,41.9,45,49.2,50.6,52.6,55.1,56.2,57.3,57.8,55,50.9,
              45.6,46.5,48.7,51.3,57.7,58.7,57.5,61.6,65,69.7,
              START=c(1920,1),FREQ=1),
  g
  =TIMESERIES(4.6,6.6,6.1,5.7,6.6,6.5,6.6,7.6,7.9,8.1,9.4,10.7,10.2,9.3,10,
              10.5,10.3,11,13,14.4,15.4,22.3,
              START=c(1920,1),FREQ=1),
  i
  =TIMESERIES(2.7,-.2,1.9,5.2,3,5.1,5.6,4.2,3,5.1,1,-3.4,-6.2,-5.1,-3,-1.3,
```

```
              2.1,2,-1.9,1.3,3.3,4.9,
              START=c(1920,1),FREQ=1),
  k
  =TIMESERIES(182.8,182.6,184.5,189.7,192.7,197.8,203.4,207.6,210.6,215.7,
              216.7,213.3,207.1,202,199,197.7,199.8,201.8,199.9,
              201.2,204.5,209.4,
              START=c(1920,1),FREQ=1),
  p
  =TIMESERIES(12.7,12.4,16.9,18.4,19.4,20.1,19.6,19.8,21.1,21.7,15.6,11.4,
              7,11.2,12.3,14,17.6,17.3,15.3,19,21.1,23.5,
              START=c(1920,1),FREQ=1),
  w1
  =TIMESERIES(28.8,25.5,29.3,34.1,33.9,35.4,37.4,37.9,39.2,41.3,37.9,34.5,
              29,28.5,30.6,33.2,36.8,41,38.2,41.6,45,53.3,
              START=c(1920,1),FREQ=1),
  y
  =TIMESERIES(43.7,40.6,49.1,55.4,56.4,58.7,60.3,61.3,64,67,57.7,50.7,41.3,
              45.3,48.9,53.3,61.8,65,61.2,68.4,74.1,85.3,
              START=c(1920,1),FREQ=1),
  t
  =TIMESERIES(3.4,7.7,3.9,4.7,3.8,5.5,7,6.7,4.2,4,7.7,7.5,8.3,5.4,6.8,7.2,
              8.3,6.7,7.4,8.9,9.6,11.6,
              START=c(1920,1),FREQ=1),
  time
  =TIMESERIES(NA,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,
              START=c(1920,1),FREQ=1),
  w2
  =TIMESERIES(2.2,2.7,2.9,2.9,3.1,3.2,3.3,3.6,3.7,4,4.2,4.8,5.3,5.6,6,6.1,
              7.4,6.7,7.7,7.8,8,8.5,
              START=c(1920,1),FREQ=1)
)

#load model and model data
myModel<-LOAD_MODEL(modelText=myModelDefinition)
myModel<-LOAD_MODEL_DATA(myModel,myModelData)

#estimate model
myModel<-ESTIMATE(myModel)

#calculate impact multipliers of Government Expenditure 'g' and
#Government Wage Bill 'w2' with respect of Consumption 'cn' and
#Gross National Product 'y' in the Klein model on the year 1941:

myModel<-MULTMATRIX(myModel,
                    symType='STATIC',
                    TSRANGE=c(1941,1,1941,1),
                    INSTRUMENT=c('w2','g'),
                    TARGET=c('cn','y'))

#Multiplier Matrix:    100.00%
#...MULTMATRIX OK

print(myModel$MultiplierMatrix)
```

```
#          w2_1      g_1
#cn_1 0.4540346 1.671956
#y_1  0.2532000 3.653260

#Results show that the impact multiplier of "y"
#with respect to "g" is +3.65
#If we change Government Expenditure 'g' value in 1941
#from 22.3 (its historical value) to 23.3 (+1)
#then the simulated Gross National Product "y"
#in 1941 changes from 95.2 to 99,
#thusly roughly confirming the +3.65 impact multiplier.
#Note that "g" appears only once in the model definition, and only
#in the "y" equation, with a coefficient equal to one. (Keynes would approve)



#multi-period interim multipliers
myModel<-MULTMATRIX(myModel,
                    TSRANGE=c(1940,1,1941,1),
                    INSTRUMENT=c('w2','g'),
                    TARGET=c('cn','y'))

#output multipliers matrix (note the zeros when the period
#of the INSTRUMENT is greater than the period of the TARGET)
print(myModel$MultiplierMatrix)
#           w2_1      g_1      w2_2      g_2
#cn_1  0.4478202 1.582292 0.0000000 0.000000
#y_1   0.2433382 3.510971 0.0000000 0.000000
#cn_2 -0.3911001 1.785042 0.4540346 1.671956
#y_2  -0.6251177 2.843960 0.2532000 3.653260


#multiplier matrix with endogenous variable 'w1' as instrument
#note the ADDFACTOR suffix in the column name, referring to the
#constant adjustment of the endogneous 'w1'
myModel<-MULTMATRIX(myModel,
                     TSRANGE=c(1940,1,1941,1),
                     INSTRUMENT=c('w2','w1'),
                     TARGET=c('cn','y'))

#Multiplier Matrix:    100.00%
#...MULTMATRIX OK
myModel$MultiplierMatrix
#           w2_1 w1_ADDFACTOR_1      w2_2 w1_ADDFACTOR_2
#cn_1  0.4478202      0.7989328 0.0000000      0.0000000
#y_1   0.2433382      0.4341270 0.0000000      0.0000000
#cn_2 -0.3911001     -0.4866248 0.4540346      0.8100196
#y_2  -0.6251177     -0.9975073 0.2532000      0.4517209
```

NAMELIST *Named List of Time Series*

Description

In the case of strings input, this function returns a string array built with the input strings. In the case of time series input, this function returns a list built with the input time series; the output list names will be the variable names passed as arguments.

Usage

```
NAMELIST(...)
```

Arguments

...	List of strings or list of time series. In the case of a list of strings, if an input string is not eligible to be a variable name, e.g. a string composed only with numbers, or with special characters, a warning will be thrown and a message will describe the required change made to the input string in order to make it eligible to be a variable name (see example).

Value

In the case of strings as input, this function returns a string array built with the input strings. In the case of time series as input, this function returns a list built with the input time series; the output list names will be the variable names passed as arguments.

See Also

NOELS
is.bimets
BIMETS indexing
TSERIES
GETYEARPERIOD
LOCS

Examples

```
#NAMELIST with time series...

ts1<-TSERIES(1:10,START=c(2000,1),FREQ=12)
ts2<-TSERIES(10:20,START=c(2002,5),FREQ=12)
myNameList<-NAMELIST(ts1,ts2)
print(myNameList)

#prints a list with $ts1 and $ts2 elements
```

```
#please note that names are 'ts1' and 'ts2'...
#$ts1
#     Jan Feb Mar Apr May Jun Jul Aug Sep Oct
#2000   1   2   3   4   5   6   7   8   9  10

#$ts2
#     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
#2002                  10  11  12  13  14  15  16  17
#2003  18  19  20



#define strange namelist
#print warnings... '' converted in 'X', '9' converted in 'X9'
myNAMELIST<-NAMELIST('pp','oo','ii','','9');

print(myNAMELIST)
```

NOELS *Count Elements*

Description

This function returns a numerical array built with the length of each input argument. Input arguments can be numerical or time series. Input arguments can also be string variables.

Usage

```
NOELS(...)
```

Arguments

`...`	List of input arguments. This function accepts input of class `numerical`, `character`, or BIMETS time series. `character` arguments are exclusive: if an argument is of class `character`, all other arguments must be of class `character`.

Value

This function returns an array of class `c()`.

See Also

TIMESERIES
is.bimets
BIMETS indexing
TSDATES
LOCS
NAMELIST

INTS
TSINFO
TSLOOK
TABIT

Examples

```
out_NOELS<-NOELS(c(1,2,3,4),c(5,6,7))
print(out_NOELS) #print c(4,3)

out_NOELS<-NOELS(TSERIES(c(1,2,3,4),START=c(2000,1),FREQ=1),c(5,6,7))
print(out_NOELS) #print c(4,3)

out_NOELS<-NOELS('aaa','bb')
print(out_NOELS) #print c(3,2)
```

---

normalizeYP *Normalize Year-Period Array*

Description

This function normalizes a numerical array `c(YEAR,PERIOD)`, given a frequency `f` and `PERIOD>=f`. e.g. `normalizeYP(c(2000,15),12) = c(2001,3)`

Usage

```
normalizeYP(x = NULL, f = NULL)
```

Arguments

`x`	Input numerical array `c(YEAR,PERIOD)`
`f`	Frequency of normalization. Must be a positive integer.

Value

This function returns a numerical array `c(YEAR,PERIOD)`

See Also

NUMPERIOD
frequency

Examples

```
#c(2,13) and frequency=4 => c(5,1)
print(normalizeYP(c(2,13),4))
```

NUMPERIOD *Distance Between Two Year-Periods*

Description

This function returns the number of time periods that lie between the provided starting period `x1=c(YEAR1,PRD1)` and the provided ending period `x2=c(YEAR2,PRD2)`, given a frequency `f`.

Usage

```
NUMPERIOD(x1, x2, f = NULL)
```

Arguments

`x1`	Starting period specified as a numerical array `c(YEAR,PRD)`
`x2`	Ending period specified as a numerical array `c(YEAR,PRD)`
`f`	Frequency over the year. It must be a positive integer.

Value

This function returns an integer of class `numeric`.

See Also

`normalizeYP`
`frequency`
`GETDATE`
`LOCS`
`NAMELIST`

Examples

```
# f=5, c(3,4) - c(2,3) = 6 periods
print(NUMPERIOD(c(2,3),c(3,4),5))
```

OPTIMIZE *Optimal control of a BIMETS model*

Description

The OPTIMIZE procedure provides a convenient method for performing optimal control exercises; the procedure maximizes an arbitrary objective-function under the constraints imposed by the econometric model and by user-specified constraints.

An approach to policy evaluation is via a so-called "social welfare function". This approach relaxes the assumptions of the instruments-targets framework, i.e. the RENORM procedure. Rather than assuming specific desired targets for some endogenous variables, it assumes the existence of a social welfare function determining a scalar measure of performance based on both endogenous and policy (exogenous) variables.

The social welfare function can incorporate information about tradeoffs in objectives that are not allowed by the RENORM instruments-targets approach.

BIMETS supplies the OPTIMIZE procedure in order to perform optimal control exercises on econometric models.

The optimization consists of maximizing a social welfare function, i.e. the objective-function, depending on exogenous and (simulated) endogenous variables, subject to user constraints plus the constraints imposed by the econometric model equations. Users are allowed to define constraints and objective-functions of any degree, and are allowed to provide different constraints and objective-functions in different optimization time periods.

The core of the OPTIMIZE procedure is based on a Monte Carlo method that takes advantage of the STOCHSIMULATE procedure. Policy variables, i.e. INSTRUMENT, are uniformly perturbed in the range defined by the user-provided boundaries, then the INSTRUMENT values that i) verify the user-provided constraints and ii) maximize the objective-functions are selected and stored into the optimize element of the output BIMETS model.

The following steps can describe the procedure implemented in OPTIMIZE:

1) check the correctness of input arguments;

2) perform a STOCHSIMULATE by uniformly perturbing the INSTRUMENT variables inside the user-boundaries provided in the OptimizeBounds function argument;

3) during the STOCHSIMULATE, for each period in the optimization TSRANGE: i) discard the stochastic realizations that do not verify the restrictions provided in the OptimizeRestrictions argument; ii) for all the remaining realizations, compute the current value of the objective-functions time series, as defined in the OptimizeFunctions argument, by using the exogenous and (simulated) endogenous stochastic time series;

4) once the STOCHSIMULATE completes, select the stochastic realization that presents the higher

value in the sum of the corresponding objective-function time series values, and return, among other data, the related optimal `INSTRUMENT` time series.

In the following figure, the scatter plot is populated with `2916` objective function stochastic realizations, computed by using the example code at the end of this section; the `210.58` local maximum is highlighted
(i.e. `advancedKleinModel$optimize$optFunMax` in first example).

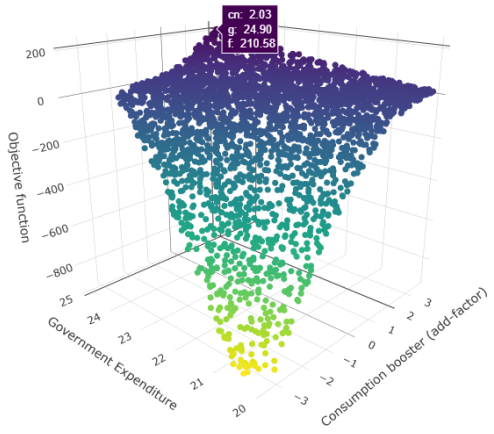
In this example:

i) The objective function definition is:
$f(y, cn, g) = (y - 110) + (cn - 90) * |cn - 90| - \sqrt{g - 20}$
given y as the simulated *Gross National Product*, cn as the simulated *Consumption* and g as the exogenous *Government Expenditure*: the basic idea is to maximize *Consumption*, and secondarily the *Gross National Product*, while reducing the *Government Expenditure*;

ii) The `INSTRUMENT` variables are the cn *Consumption* "booster" (i.e. the add-factor, not to be confused with the simulated *Consumption* in the objective function) and the g *Government Expenditure*, defined over the following domains: $cn \in (-5, 5)$, $g \in (15, 25)$;

iii) The following restrictions are applied to the `INSTRUMENT`: $g + cn^2/2 < 27 \wedge g + cn > 17$, given cn as the *Consumption* "booster" (i.e. the add-factor) and g as the *Government Expenditure*;

Advanced Klein Model: Monte-Carlo optimal control



Objective function stochastic realizations that are computable and verify the restrictions. Local maximum is highlighted. See code example for definitions and formulas.

The figure clearly shows that non-linear restrictions have been applied, and that non-computable objective functions have been discarded, e.g. the stochastic realizations having $g < 20$ due to the square root operation in the objective function, given instrument $g \in (15, 25)$.

Usage

```
OPTIMIZE( model=NULL,
          simAlgo='GAUSS-SEIDEL',
          TSRANGE=NULL,
          simType='DYNAMIC',
          simConvergence=0.01,
          simIterLimit=100,
          ZeroErrorAC=FALSE,
          Exogenize=NULL,
          ConstantAdjustment=NULL,
          verbose=FALSE,
          verboseSincePeriod=0,
          verboseVars=NULL,
          StochReplica=100,
          StochSeed=NULL,
          OptimizeBounds=NULL,
          OptimizeRestrictions=NULL,
          OptimizeFunctions=NULL,
          quietly=FALSE,
          RESCHECKeqList=NULL,
          JACOBIAN_SHOCK=1e-4,
          JacobianDrop=NULL,
          avoidCompliance=FALSE,
          ...)
```

Arguments

model	see SIMULATE
simAlgo	see SIMULATE
TSRANGE	see SIMULATE
simType	see SIMULATE
simConvergence	see SIMULATE
simIterLimit	see SIMULATE
ZeroErrorAC	see SIMULATE
Exogenize	see SIMULATE
ConstantAdjustment	see SIMULATE
verbose	see SIMULATE
verboseSincePeriod	see SIMULATE
verboseVars	see SIMULATE
StochReplica	see STOCHSIMULATE
StochSeed	see STOCHSIMULATE

OptimizeBounds the named list() that defines the search boundaries applied to INSTRUMENT exogenous variables. Each list element must have a name equal to an endogenous or an exogenous model variable.

The list names define the INSTRUMENT.

If a list element name is equal to an exogenous variable, then the boundaries will be applied directly to the related exogenous stochastic time series values. If a list element name is equal to an endogenous variable, then the boundaries will be applied to the stochastic constant adjustment (see STOCHSIMULATE) of the related endogenous variable.

Each list element must be a named list built with the following two named variables:

- TSRANGE: the time range wherein the search boundaries are active. The TSRANGE must be a 4 numerical array,
i.e. TSRANGE=c(start_year, start_period, end_year, end_period) or TSRANGE=TRUE in order to apply the provided boundaries to the whole OPTIMIZE TSRANGE.

- BOUNDS: the boundaries that are applied to the related instrument. These parameters must contain the lower and upper bound of the uniform distribution wherein the search for the objective-functions maximum is performed,
i.e. BOUNDS=c(lower_bound,upper_bound).

See example in order to learn how to build a compliant boundaries structure.

OptimizeRestrictions
the named list() that defines the restrictions applied to INSTRUMENT exogenous variables. This list can be NULL.

Each list element must be a named list built with the following two named variables:

- TSRANGE: the time range wherein the restriction is active. The TSRANGE must be a 4 numerical array,
i.e. TSRANGE=c(start_year, start_period, end_year, end_period) or TSRANGE=TRUE in order to apply the provided restriction to the whole OPTIMIZE TSRANGE.

- INEQUALITY: the inequality expression, i.e. a character variable, that defines the restriction. The INEQUALITY expression can contain exogenous and endogenous variable names, the standard arithmetic and logical operators, parentheses and the MDL functions described in the EQ section of the MDL help page. If in the INEQUALITY expression a variable name refers to an exogenous variable, then that variable will be evaluated by using the related exogenous time series stochastic values. If in the INEQUALITY expression a variable name refers to an endogenous variable, then that variable will be evaluated by using to the stochastic constant adjustment (see argument StochStructure of the STOCHSIMULATE help page) of the related endogenous variable.

Two different OptimizeRestrictions list element can not have overlapping TSRANGE. See example in order to learn how to build a compliant restrictions structure.

OptimizeFunctions
: the named list() that defines the objective functions to be maximized.

  Each list element must be a named list built with the following two named variables:

  - TSRANGE: the time range wherein the objective function is evaluated. The TSRANGE must be a 4 numerical array,
  i.e. TSRANGE=c(start_year, start_period, end_year, end_period) or TSRANGE=TRUE in order to evaluate the objective function in each period of the OPTIMIZE TSRANGE.

  - FUNCTION: the expression, i.e. a character variable, that defines the objective function. The FUNCTION expression can contain exogenous and endogenous variable names, the standard arithmetic and logical operators, parentheses and the MDL functions described in the EQ section of the MDL help page. If in the FUNCTION expression a variable name refers to an exogenous variable, then that variable will be evaluated by using the related exogenous time series stochastic values. If in the FUNCTION expression a variable name refers to an endogenous variable, then that variable will be evaluated by using the stochastic simulated time series (see STOCHSIMULATE) of the related endogenous variable.

  Two different OptimizeFunctions list element can not have overlapping TSRANGE. See example in order to learn how to build a compliant objective functions structure.

quietly
: see SIMULATE

RESCHECKeqList
: see SIMULATE

JACOBIAN_SHOCK
: see SIMULATE

JacobianDrop
: see SIMULATE

avoidCompliance
: see SIMULATE

...
: see SIMULATE

Value

This function will add, into the output BIMETS model object, three new named elements, respectively optimize, simulation_MM and INSTRUMENT_MM.

The optimize element is a named list() that contains the following elements:

- INSTRUMENT: a named list that contains the time series of the instrument exogenous variables that verify the OptimizeRestrictions and that allow the objective OptimizeFunctions to be maximized. This element is populated only if a finite solution exists. List names are equal to

the names of the related exogenous variables. Users can also declare an endogenous variable as INSTRUMENT variable, by using the OptimizeBounds argument: in this case the constant adjustment (see STOCHSIMULATE) related to the provided endogenous variable will be used as instrument exogenous variable, and this output INSTRUMENT list will contains the constant adjustment time series that allow the objective OptimizeFunction to be maximized (see example);

- optFunMax: the scalar value (local maximum) obtained by evaluating the OptimizeFunctions while the model is fed by the optimized INSTRUMENT time series. This element is populated only if a finite solution exists;

- optFunTS: the time series obtained by evaluating the OptimizeFunctions during each period in the OPTIMIZE TSRANGE while the model is fed by the optimized INSTRUMENT time series. Thus, optFunMax==sum(optFunTS). This element is populated only if a finite solution exists;

- optFunAve: the scalar value that is the mean of all the stochastic OptimizeFunctions realizations, filtered by the restrictions imposed by the OptimizeRestrictions argument. This element is populated only if a finite solution exists;

- optFunSd: the scalar value that is the standard deviation of all the stochastic OptimizeFunctions realizations, filtered by the restrictions imposed by the OptimizeRestrictions argument. This element is populated only if a finite solution exists;

- realizationsToKeep: a 1 x StochReplica boolean row array. If the i-th element is TRUE than the related objective function realization is computable and verifies the restrictions imposed by the OptimizeRestricions argument. It can be useful along with optFunResults and INSTRUMENT_MM in order to verify and to refine results;

- optFunResults: the numerical array containing the evaluated OptimizeFunctions for all the (unfiltered) realizations;

- modelData: the whole model input dataset wherein the INSTRUMENT exogenous variables have been modified accordingly to the OPTIMIZE results. This data can be useful in order to verify or to refine results (see example);

- ConstantAdjustment: a modified constant adjustment input list wherein the constant adjustment time series related to a INSTRUMENT endogenous variables have been modified accordingly to the OPTIMIZE results. This data can be useful in order to verify or to refine results (see example);

The arguments passed to the function call during the latest OPTIMIZE run will be inserted into the '__OPT_PARAMETERS__' element of the model optimize list; this data can be helpful in order to replicate the optimization results.

The simulation_MM element is a named list(), having the endogenous variables as names. Each element will contain an R x C matrix, given R the number of observations in the optimization TSRANGE and C=1+StochReplica. The first column of each matrix contains the related endogenous

variable's unperturbed simulated values; the remaining columns will contain all the StochReplica stochastic realizations for the related endogenous variable.

The INSTRUMENT_MM element is a named list(), having INSTRUMENT variables as names. Each element will contain an R x C matrix, given R the number of observations in the optimization TSRANGE and C=1+StochReplica. The first column of each matrix contains the related INSTRUMENT variable's unperturbed values; the remaining columns will contain all the StochReplica stochastic realizations for the related INSTRUMENT variable.

See Also

MDL
LOAD_MODEL
ESTIMATE
STOCHSIMULATE
MULTMATRIX
RENORM
TIMESERIES
BIMETS indexing
BIMETS configuration

Examples

```
#define the advanced Klein model
advancedKleinModelDef <- "
MODEL

COMMENT> Modified Klein Model 1 of the U.S. Economy with PDL,
COMMENT> autocorrelation on errors, restrictions and conditional equation evaluations

COMMENT> Consumption with autocorrelation on errors
BEHAVIORAL> cn
TSRANGE 1923 1 1940 1
EQ> cn =  a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)
COEFF> a1 a2 a3 a4
ERROR> AUTO(2)

COMMENT> Investment with restrictions
BEHAVIORAL> i
TSRANGE 1923 1 1940 1
EQ> i = b1 + b2*p + b3*TSLAG(p,1) + b4*TSLAG(k,1)
COEFF> b1 b2 b3 b4
RESTRICT> b2 + b3 = 1

COMMENT> Demand for Labor with PDL
BEHAVIORAL> w1
```

```
TSRANGE 1923 1 1940 1
EQ> w1 = c1 + c2*(y+t-w2) + c3*TSLAG(y+t-w2,1) + c4*time
COEFF> c1 c2 c3 c4
PDL> c3 1 2

COMMENT> Gross National Product
IDENTITY> y
EQ> y = cn + i + g - t

COMMENT> Profits
IDENTITY> p
EQ> p = y - (w1+w2)

COMMENT> Capital Stock with IF switches
IDENTITY> k
EQ> k = TSLAG(k,1) + i
IF> i > 0
IDENTITY> k
EQ> k = TSLAG(k,1)
IF> i <= 0

END
"

#load the model
advancedKleinModel <- LOAD_MODEL(modelText = advancedKleinModelDef)


#define data
kleinModelData <- list(
    cn  =TIMESERIES(39.8,41.9,45,49.2,50.6,52.6,55.1,56.2,57.3,57.8,
                    55,50.9,45.6,46.5,48.7,51.3,57.7,58.7,57.5,61.6,65,69.7,
                    START=c(1920,1),FREQ=1),
    g   =TIMESERIES(4.6,6.6,6.1,5.7,6.6,6.5,6.6,7.6,7.9,8.1,9.4,10.7,
                    10.2,9.3,10,10.5,10.3,11,13,14.4,15.4,22.3,
                    START=c(1920,1),FREQ=1),
    i   =TIMESERIES(2.7,-.2,1.9,5.2,3,5.1,5.6,4.2,3,5.1,1,-3.4,-6.2,
                    -5.1,-3,-1.3,2.1,2,-1.9,1.3,3.3,4.9,
                    START=c(1920,1),FREQ=1),
    k   =TIMESERIES(182.8,182.6,184.5,189.7,192.7,197.8,203.4,207.6,
                    210.6,215.7,216.7,213.3,207.1,202,199,197.7,199.8,
                    201.8,199.9,201.2,204.5,209.4,
                    START=c(1920,1),FREQ=1),
    p   =TIMESERIES(12.7,12.4,16.9,18.4,19.4,20.1,19.6,19.8,21.1,21.7,
                    15.6,11.4,7,11.2,12.3,14,17.6,17.3,15.3,19,21.1,23.5,
                    START=c(1920,1),FREQ=1),
    w1  =TIMESERIES(28.8,25.5,29.3,34.1,33.9,35.4,37.4,37.9,39.2,41.3,
                    37.9,34.5,29,28.5,30.6,33.2,36.8,41,38.2,41.6,45,53.3,
                    START=c(1920,1),FREQ=1),
    y   =TIMESERIES(43.7,40.6,49.1,55.4,56.4,58.7,60.3,61.3,64,67,57.7,
                    50.7,41.3,45.3,48.9,53.3,61.8,65,61.2,68.4,74.1,85.3,
                    START=c(1920,1),FREQ=1),
    t   =TIMESERIES(3.4,7.7,3.9,4.7,3.8,5.5,7,6.7,4.2,4,7.7,7.5,8.3,5.4,
```

```
                    6.8,7.2,8.3,6.7,7.4,8.9,9.6,11.6,
                    START=c(1920,1),FREQ=1),
    time=TIMESERIES(NA,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,
                    1,2,3,4,5,6,7,8,9,10,
                    START=c(1920,1),FREQ=1),
    w2  =TIMESERIES(2.2,2.7,2.9,2.9,3.1,3.2,3.3,3.6,3.7,4,4.2,4.8,
                    5.3,5.6,6,6.1,7.4,6.7,7.7,7.8,8,8.5,
                    START=c(1920,1),FREQ=1)
);

#load time series into the model object
advancedKleinModel <- LOAD_MODEL_DATA(advancedKleinModel,kleinModelData)

#estimate the model
advancedKleinModel <- ESTIMATE(advancedKleinModel, quietly=TRUE)

#we want to maximize the non-linear objective function:
#f()=(y-110)+(cn-90)*ABS(cn-90)-(g-20)^0.5
#in 1942 by using INSTRUMENT cn in range (-5,5)
#(cn is endogenous so we use the add-factor)
#and g in range (15,25)
#we will also impose the following non-linear restriction:
#g+(cn^2)/2<27 & g+cn>17

#we need to extend exogenous variables up to 1942
advancedKleinModel$modelData <- within(advancedKleinModel$modelData,{
    w2    = TSEXTEND(w2,  UPTO=c(1942,1),EXTMODE='CONSTANT')
    t     = TSEXTEND(t,   UPTO=c(1942,1),EXTMODE='LINEAR')
    g     = TSEXTEND(g,   UPTO=c(1942,1),EXTMODE='CONSTANT')
    k     = TSEXTEND(k,   UPTO=c(1942,1),EXTMODE='LINEAR')
    time  = TSEXTEND(time,UPTO=c(1942,1),EXTMODE='LINEAR')
})

#define INSTRUMENT and boundaries
myOptimizeBounds <- list(
    cn=list(TSRANGE=TRUE,
            BOUNDS=c(-5,5)),
    g=list(TSRANGE=TRUE,
           BOUNDS=c(15,25))
)

#define restrictions
myOptimizeRestrictions <- list(
    myRes1=list(
        TSRANGE=TRUE,
        INEQUALITY='g+(cn^2)/2<27 & g+cn>17')
)

#define objective function
myOptimizeFunctions <- list(
    myFun1=list(
        TSRANGE=TRUE,
        FUNCTION='(y-110)+(cn-90)*ABS(cn-90)-(g-20)^0.5')
```

```
)

#Monte-Carlo optimization by using 10000 stochastic realizations
#and 1E-4 convergence criterion
advancedKleinModel <- OPTIMIZE(advancedKleinModel
                          ,simType = 'FORECAST'
                          ,TSRANGE=c(1942,1,1942,1)
                          ,simConvergence= 1E-4
                          ,simIterLimit  = 1000
                          ,StochReplica  = 10000
                          ,StochSeed = 123
                          ,OptimizeBounds = myOptimizeBounds
                          ,OptimizeRestrictions = myOptimizeRestrictions
                          ,OptimizeFunctions = myOptimizeFunctions)

#OPTIMIZE(): optimization boundaries for the add-factor of endogenous
#            variable "cn" are (-5,5) from year-period 1942-1 to 1942-1.
#OPTIMIZE(): optimization boundaries for the exogenous
#            variable "g" are (15,25) from year-period 1942-1 to 1942-1.
#OPTIMIZE(): optimization restriction "myRes1" is active
#            from year-period 1942-1 to 1942-1.
#OPTIMIZE(): optimization objective function "myFun1" is active
#            from year-period 1942-1 to 1942-1.
#
#Optimize:     100.00 %
#OPTIMIZE(): 2916 out of 10000 objective function realizations (29%)
#            are finite and verify the provided restrictions.
#...OPTIMIZE OK

#print local maximum
advancedKleinModel$optimize$optFunMax
#[1] 210.5755

#print INSTRUMENT that allow local maximum to be achieved
advancedKleinModel$optimize$INSTRUMENT
#$cn
#Time Series:
#Start = 1942
#End = 1942
#Frequency = 1
#[1] 2.032203
#
#$g
#Time Series:
#Start = 1942
#End = 1942
#Frequency = 1
#[1] 24.89773

#LET'S VERIFY RESULTS
#copy into modelData the computed INSTRUMENT
#that allow to maximize the objective function
advancedKleinModel$modelData <- advancedKleinModel$optimize$modelData
```

```
#simulate the model by using the new INSTRUMENT
#note: we used cn add-factor as OPTIMIZE instrument, so we need
#to pass the computed cn add-factor to the SIMULATE call
newConstantAdjustment <- advancedKleinModel$optimize$ConstantAdjustment
advancedKleinModel <- SIMULATE(advancedKleinModel
                  ,simType = 'FORECAST'
                  ,TSRANGE = c(1942,1,1942,1)
                  ,simConvergence = 1E-5
                  ,simIterLimit = 1000
                  ,ConstantAdjustment = newConstantAdjustment
)

#calculate objective function by using the SIMULATE output time series
#(y-110)+(cn-90)*ABS(cn-90)-(g-20)^0.5
y  <- advancedKleinModel$simulation$y
cn <- advancedKleinModel$simulation$cn
g  <- advancedKleinModel$modelData$g
optFunTest <- (y-110)+(cn-90)*abs(cn-90)-(g-20)^0.5

#verify computed max is equal to optimization max
#(in the following command TSPROJECT could be omitted because
#myFun1$TSRANGE = TRUE)
abs(sum(TSPROJECT(optFunTest
              ,TSRANGE=c(1942,1,1942,1)
              ,ARRAY = TRUE)
        ) - advancedKleinModel$optimize$optFunMax)  < 1E-4
#[1] TRUE

#we can also check that the SIMULATE time series
#are equal to the OPTIMIZE realizations that allow to maximize
#the objective function

#get realization index that maximizes the objective function
maximizingRealizationIdx=with(advancedKleinModel$optimize,
                              which.max(optFunResults[realizationsToKeep]))

#get stochastic realizations unfiltered
#(simulation_MM and INSTRUMENT_MM are populated during the OPTIMIZE call)
y_opt  <- advancedKleinModel$simulation_MM$y
cn_opt <- advancedKleinModel$simulation_MM$cn
g_opt  <- advancedKleinModel$INSTRUMENT_MM$g

#filter by restrictions and by finite solutions
#(first column in all matrices is related to the un-perturbed model)
y_opt  <- y_opt[ ,c(FALSE,advancedKleinModel$optimize$realizationsToKeep),drop=FALSE]
cn_opt <- cn_opt[,c(FALSE,advancedKleinModel$optimize$realizationsToKeep),drop=FALSE]
g_opt  <- g_opt[ ,c(FALSE,advancedKleinModel$optimize$realizationsToKeep),drop=FALSE]

#get maximizing realizations
y_opt  <- y_opt[ ,maximizingRealizationIdx,drop=FALSE]
cn_opt <- cn_opt[,maximizingRealizationIdx,drop=FALSE]
g_opt  <- g_opt[ ,maximizingRealizationIdx,drop=FALSE]
```

```
#verify that these variables are equal to the SIMULATE time series
max(abs(y-y_opt))   < 1E-4
#[1] TRUE

max(abs(cn-cn_opt)) < 1E-4
#[1] TRUE

max(abs(g[[1942,1]]-g_opt)) < 1E-4
#[1] TRUE


############################################################
#MULTI RESTRICTIONS, MULTI OBJECTIVE FUNCTIONS EXAMPLE

#load the model (reset stuff)
advancedKleinModel <- LOAD_MODEL(modelText = advancedKleinModelDef)

#load time series into the model object
advancedKleinModel <- LOAD_MODEL_DATA(advancedKleinModel,kleinModelData)

#estimate the model
advancedKleinModel <- ESTIMATE(advancedKleinModel, quietly=TRUE)

#we want to maximize the non-linear objective function:
#f1()=(y-110)+(cn-90)*ABS(cn-90)-(g-20)^0.5
#in 1942 by using INSTRUMENT cn in range (-5,5)
#(cn is endogenous so we use the add-factor)
#and g in range (15,25)
#we want to maximize the non-linear objective function:
#f2()=(y-120)+(cn-100)*ABS(cn-100)-(g-20)^0.5-(w2-8)^0.5
#in 1943 by using INSTRUMENT cn in range (-5,5),
#g in range (15,25)
#and w2 in range (7.5,12.5)
#we will also impose the following non-linear restrictions:
#in 1942: g+(cn^2)/2<27 & g+cn>17
#in 1943: (g^2)/10+(cn^2)/2+w2^2 < 200

#we need to extend exogenous variables up to 1943
advancedKleinModel$modelData <- within(advancedKleinModel$modelData,{
    w2    = TSEXTEND(w2,  UPTO=c(1943,1),EXTMODE='CONSTANT')
    t     = TSEXTEND(t,   UPTO=c(1943,1),EXTMODE='LINEAR')
    g     = TSEXTEND(g,   UPTO=c(1943,1),EXTMODE='CONSTANT')
    k     = TSEXTEND(k,   UPTO=c(1943,1),EXTMODE='LINEAR')
    time  = TSEXTEND(time,UPTO=c(1943,1),EXTMODE='LINEAR')
})

#define INSTRUMENT and boundaries
myOptimizeBounds <- list(
    cn=list(TSRANGE=TRUE,
            BOUNDS=c(-5,5)),
    g=list(TSRANGE=TRUE,
           BOUNDS=c(15,25)),
```

```
    w2=list(TSRANGE=c(1943,1,1943,1),
            BOUNDS=c(7.5,12.5))
)

#define restrictions
myOptimizeRestrictions <- list(
    myRes1=list(
        TSRANGE=c(1942,1,1942,1),
        INEQUALITY='g+(cn^2)/2 < 27 & g+cn > 17'),
    myRes2=list(
        TSRANGE=c(1943,1,1943,1),
        INEQUALITY='(g^2)/10+(cn^2)/2+w2^2 < 200')
)

#define objective functions
myOptimizeFunctions <- list(
    myFun1=list(
        TSRANGE=c(1942,1,1942,1),
        FUNCTION='(y-110)+(cn-90)*ABS(cn-90)-(g-20)^0.5'),
    myFun2=list(
        TSRANGE=c(1943,1,1943,1),
        FUNCTION='(y-120)+(cn-100)*ABS(cn-100)-(g-20)^0.5-(w2-8)^0.5')
)

#Monte-Carlo optimization by using 1000 stochastic realizations
#and 1E-4 convergence
advancedKleinModel <- OPTIMIZE(advancedKleinModel
                          ,simType = 'FORECAST'
                          ,TSRANGE=c(1942,1,1943,1)
                          ,simConvergence=1E-4
                          ,simIterLimit = 500
                          ,StochReplica = 1000
                          ,StochSeed = 123
                          ,OptimizeBounds = myOptimizeBounds
                          ,OptimizeRestrictions = myOptimizeRestrictions
                          ,OptimizeFunctions = myOptimizeFunctions)

#print INSTRUMENT that allow local maximum to be achieved
advancedKleinModel$optimize$INSTRUMENT

#LET'S VERIFY RESULTS
#copy into modelData the computed INSTRUMENT
#that allow to maximize the objective function
advancedKleinModel$modelData <- advancedKleinModel$optimize$modelData

#simulate the model by using the new INSTRUMENT
newConstantAdjustment <- advancedKleinModel$optimize$ConstantAdjustment
advancedKleinModel <- SIMULATE(advancedKleinModel
                  ,simType = 'FORECAST'
                  ,TSRANGE = c(1942,1,1943,1)
                  ,simConvergence = 1E-5
                  ,simIterLimit = 100
                  ,ConstantAdjustment = newConstantAdjustment
```

```
)

#calculate objective functions by using the SIMULATE output time series
y  <- advancedKleinModel$simulation$y
cn <- advancedKleinModel$simulation$cn
g  <- advancedKleinModel$modelData$g
w2 <- advancedKleinModel$modelData$w2
optFunTest1 <- (y-110)+(cn-90)*abs(cn-90)-(g-20)^0.5
optFunTest2 <- (y-120)+(cn-100)*abs(cn-100)-(g-20)^0.5-(w2-8)^0.5

#verify computed max is equal to optimization max
abs(sum(TSPROJECT(optFunTest1
                  ,TSRANGE=c(1942,1,1942,1)
                  ,ARRAY = TRUE)+
        TSPROJECT(optFunTest2
                  ,TSRANGE=c(1943,1,1943,1)
                  ,ARRAY = TRUE)
        ) - advancedKleinModel$optimize$optFunMax)  < 1E-2
#[1] TRUE
```

QUARTERLY *Quarterly (Dis)Aggregation*

Description

This function returns a quarterly (dis)aggregated time series, using as input an annual, semiannual, monthly or daily time series.

Usage

```
QUARTERLY(x = NULL, fun = NULL, avoidCompliance = FALSE, ...)
```

Arguments

x	Input time series that must satisfy the compliance control check defined in is.bimets.
fun	Only for daily or monthly input time series: **STOCK**: the value of the input time series in the last observation of a quarter is assigned to the same quarter of the output time series. **NSTOCK**: the value of the input time series in the last non-missing observation of a quarter is assigned to the same quarter of the output time series. **SUM**: the sum of input observations in a quarter is assigned to the same quarter of the output time series. **NSUM**: the sum of input non-missing observations in a quarter is assigned to the same quarter of the output time series. **AVE**: the average of input observations in a quarter is assigned to the same quarter of the output time series. **NAVE**: the average of input non-missing observations in a quarter is assigned to

the same quarter of the output time series.

Only for semiannual or annual input time series:
NULL: (default) the output value of each quarterly observation is set equal to the value of the input observation the quarter belongs to (i.e. duplicated values over the period)
INTERP_END: the value of the input time series in a period is copied into the last quarter of the output time series that lies in the same period. Other values are calculated by linear interpolation.
INTERP_CENTER: the value of the input time series in a period is copied into the median quarter of the output time series that lies in the same period. Other values are calculated by linear interpolation.
INTERP_BEGIN: the value of the input time series in a period is copied into the first quarter of the output time series that lies in the same period. Other values are calculated by linear interpolation.

`avoidCompliance`
: If `TRUE`, compliance control check of input time series will be skipped. See `is.bimets`

`...`
: Backward compatibility.

Value

This function returns a quarterly BIMETS time series.

See Also

`YEARLY`
`SEMIANNUAL`
`MONTHLY`
`DAILY`

Examples

```
#TS YEARLY TO QUARTERLY
n<-10
ts1<-TSERIES(1:n,START=c(2000,1),FREQ=1)
ts1[5]<-NA
TABIT(QUARTERLY(ts1,fun='INTERP_CENTER'));

#TS DAILY TO QUARTERLY
n<-600
ts1<-TSERIES(1:n,START=c(2000,1),FREQ='D')
ts1[25]<-NA
TABIT(QUARTERLY(ts1,fun='SUM'))
```

RENORM *Endogenous targeting of a BIMETS model.*

Description

The endogenous targeting of econometric models (a.k.a. "renormalization") consists of solving the model while interchanging the role of one or more endogenous variables with an equal number of exogenous variables.

This procedure determines the values for the INSTRUMENT exogenous variables that allow the objective TARGET endogenous values to be achieved, with respect to the constraints given by the model equations (see MDL).

This is an approach to economic and monetary policy analysis, and is based on two assumptions:

1. there exists a desired level for a set of the n endogenous variables defined as TARGET;
2. there exists a set of the n exogenous variables defined as INSTRUMENT;

Given these premises, the endogenous targeting process consists in determining the values of the exogenous variables chosen as INSTRUMENT allowing us to achieve the desired values for the endogenous variables designated as TARGET. In other words the procedure allows users to exchange the role of exogenous and endogenous among a set of variables pairs.

Given a list of exogenous INSTRUMENT variables and a list of TARGET endogenous time series, the iterative procedure can be split into the following steps:

1. Computation of the multipliers matrix MULTMAT of the TARGET endogenous variables with respect to the INSTRUMENT exogenous variables (this is a square matrix by construction);

2. Solution of the linear system (if any):

$V_{exog}(i+1) = V_{exog}(i) +$ MULTMAT $^{-1} * (V_{endog}(i) -$ TARGET), where $V_{exog}(i)$ are the exogenous variables in the INSTRUMENT list and $V_{endog}(i)$ are the endogenous variables that have a related target in the TARGET list, given i the current iteration;

3. Simulation of the model with the new set of exogenous variables computed in step 2, then a convergence check by comparing the subset of endogenous variables arising from this simulation and the related time series in TARGET list. If the convergence condition is satisfied, or the maximum number of iterations is reached, the algorithm will stop, otherwise it will go back to step 1;

Users can also declare an endogenous variable as an INSTRUMENT variable. In this case, the constant adjustment (see SIMULATE) related to the provided endogenous variable will be used as the instrument exogenous variable. This procedure is particularly suited for the automatic computation of the add-factors needed to fine tune the model into a baseline path and to improve the forecasting accuracy.

If the convergence condition is satisfied, the RENORM procedure will return the INSTRUMENT time

series allowing us to achieve the desired values for the endogenous variables designated as TARGET.

Usage

```
RENORM(model=NULL,
       simAlgo='GAUSS-SEIDEL',
       TSRANGE=NULL,
       simType='DYNAMIC',
       simConvergence=0.01,
       simIterLimit=100,
       ZeroErrorAC=FALSE,
       Exogenize=NULL,
       ConstantAdjustment=NULL,
       verbose=FALSE,
       verboseSincePeriod=0,
       verboseVars=NULL,
       renormIterLimit=10,
       renormConvergence=1e-4,
       TARGET=NULL,
       INSTRUMENT=NULL,
       MM_SHOCK=0.00001,
       quietly=FALSE,
       tol=1e-28,
       JACOBIAN_SHOCK=1e-4,
       JacobianDrop=NULL,
       avoidCompliance=FALSE,
       ...
       )
```

Arguments

model	see SIMULATE
simAlgo	see SIMULATE
TSRANGE	see SIMULATE
simType	see SIMULATE
simConvergence	see SIMULATE
simIterLimit	see SIMULATE
ZeroErrorAC	see SIMULATE
Exogenize	see SIMULATE
ConstantAdjustment	see SIMULATE
verbose	see SIMULATE
verboseSincePeriod	see SIMULATE

verboseVars see SIMULATE

renormIterLimit
: The value representing the maximum number of iterations to be performed. The iterative renormalization procedure will stop when renormIterLimit is reached or the TARGET variables satisfy the renormConvergence criterion

renormConvergence
: The convergence value requested for the iterative renormalization process, that stops when the Euclidean distance between each TARGET time series and the related simulated endogenous variable is less than the renormConvergence value

TARGET
: A named list that specifies the target endogenous variables. List names must be equal to the names of the target endogenous variables involved in the renormalization; each list element must contain the time series of the desired target endogenous values; time series must be compliant with the compliance control check defined in is.bimets (see example)

INSTRUMENT
: A character array built with the names of the instrument exogenous variables involved in the renormalization. User can also declare an endogenous variable as INSTRUMENT variable: in this case the constant adjustment (see SIMULATE) related to the provided endogenous variable will be used as instrument exogenous variable (see example)

tol
: the tolerance for detecting linear dependencies in the columns of a matrix while an inversion is requested.

MM_SHOCK see MULTMATRIX

quietly see SIMULATE

JACOBIAN_SHOCK see SIMULATE

JacobianDrop see SIMULATE

avoidCompliance
: see SIMULATE

... see SIMULATE

Value

This function will add a new named element renorm into the output BIMETS model object.

This new renorm element is a named list that contains the following elements:

- INSTRUMENT: a named list that contains the INSTRUMENT exogenous time series that allow the objective TARGET endogenous values to be achieved. This element is populated only if the convergence is reached. List names are equal to the names of the related exogenous variables. Users can also declare an endogenous variable as INSTRUMENT variable: in this case the constant adjustment (see SIMULATE) related to the provided endogenous variable will be used as instrument exogenous variable, and this INSTRUMENT output list will contains the constant adjustment time series that allow the objective TARGET endogenous values to be achieved (see example);

- TARGET: a named list built with the achieved TARGET endogenous time series. List names are equal to the target endogenous variable names;

- unConvergedTARGET: the names array of the endogenous TARGET variables that failed the convergence. This element is populated only if the convergence has not been reached;

- modelData: the whole model input dataset wherein the INSTRUMENT exogenous variables have been modified accordingly to the RENORM results. This data can be useful in order to refine results or to verify that the model, fed with the proper INSTRUMENT exogenous time series, produces the desired TARGET endogenous values. This element is populated only if the convergence is achieved (see example);

- ConstantAdjustment: a modified constant adjustment input list (see SIMULATE) wherein the constant adjustment time series related to a INSTRUMENT endogenous variables have been modified accordingly to the RENORM results. This data can be useful in order to refine results or to verify that the model, fed with the proper INSTRUMENT exogenous time series (therefore with the proper ConstantAdjustment time series, if any), produces the desired TARGET endogenous values (see example). This element is populated only if the convergence is achieved;

- __RENORM_PARAMETERS__: a named list that contains the arguments passed to the function call during the latest RENORM run, e.g. TSRANGE, INSTRUMENT, TARGET, renormIterLimit, renormConvergence, ConstantAdjustment, Exogenize, etc.: this data can be useful in order to replicate renorm results.

See Also

MDL
LOAD_MODEL
ESTIMATE
SIMULATE
STOCHSIMULATE
MULTMATRIX
OPTIMIZE
TIMESERIES
BIMETS indexing
BIMETS configuration

Examples

```
#define model
myModelDefinition<-
"MODEL
COMMENT> Klein Model 1 of the U.S. Economy

COMMENT> Consumption
BEHAVIORAL> cn
TSRANGE 1921 1 1941 1
EQ> cn =  a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)
COEFF> a1 a2 a3 a4
```

```
COMMENT> Investment
BEHAVIORAL> i
TSRANGE 1921 1 1941 1
EQ> i = b1 + b2*p + b3*TSLAG(p,1) + b4*TSLAG(k,1)
COEFF> b1 b2 b3 b4

COMMENT> Demand for Labor
BEHAVIORAL> w1
TSRANGE 1921 1 1941 1
EQ> w1 = c1 + c2*(y+t-w2) + c3*TSLAG(y+t-w2,1) + c4*time
COEFF> c1 c2 c3 c4

COMMENT> Gross National Product
IDENTITY> y
EQ> y = cn + i + g - t

COMMENT> Profits
IDENTITY> p
EQ> p = y - (w1+w2)

COMMENT> Capital Stock
IDENTITY> k
EQ> k = TSLAG(k,1) + i

END"

#define model data
myModelData<-list(
  cn
  =TIMESERIES(39.8,41.9,45,49.2,50.6,52.6,55.1,56.2,57.3,57.8,55,50.9,
              45.6,46.5,48.7,51.3,57.7,58.7,57.5,61.6,65,69.7,
              START=c(1920,1),FREQ=1),
  g
  =TIMESERIES(4.6,6.6,6.1,5.7,6.6,6.5,6.6,7.6,7.9,8.1,9.4,10.7,10.2,9.3,10,
              10.5,10.3,11,13,14.4,15.4,22.3,
              START=c(1920,1),FREQ=1),
  i
  =TIMESERIES(2.7,-.2,1.9,5.2,3,5.1,5.6,4.2,3,5.1,1,-3.4,-6.2,-5.1,-3,-1.3,
              2.1,2,-1.9,1.3,3.3,4.9,
              START=c(1920,1),FREQ=1),
  k
  =TIMESERIES(182.8,182.6,184.5,189.7,192.7,197.8,203.4,207.6,210.6,215.7,
              216.7,213.3,207.1,202,199,197.7,199.8,201.8,199.9,
              201.2,204.5,209.4,
              START=c(1920,1),FREQ=1),
  p
  =TIMESERIES(12.7,12.4,16.9,18.4,19.4,20.1,19.6,19.8,21.1,21.7,15.6,11.4,
              7,11.2,12.3,14,17.6,17.3,15.3,19,21.1,23.5,
              START=c(1920,1),FREQ=1),
  w1
  =TIMESERIES(28.8,25.5,29.3,34.1,33.9,35.4,37.4,37.9,39.2,41.3,37.9,34.5,
              29,28.5,30.6,33.2,36.8,41,38.2,41.6,45,53.3,
              START=c(1920,1),FREQ=1),
```

```
  y
  =TIMESERIES(43.7,40.6,49.1,55.4,56.4,58.7,60.3,61.3,64,67,57.7,50.7,41.3,
              45.3,48.9,53.3,61.8,65,61.2,68.4,74.1,85.3,
              START=c(1920,1),FREQ=1),
  t
  =TIMESERIES(3.4,7.7,3.9,4.7,3.8,5.5,7,6.7,4.2,4,7.7,7.5,8.3,5.4,6.8,7.2,
              8.3,6.7,7.4,8.9,9.6,11.6,
              START=c(1920,1),FREQ=1),
  time
  =TIMESERIES(NA,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,
              START=c(1920,1),FREQ=1),
  w2
  =TIMESERIES(2.2,2.7,2.9,2.9,3.1,3.2,3.3,3.6,3.7,4,4.2,4.8,5.3,5.6,6,6.1,
              7.4,6.7,7.7,7.8,8,8.5,
              START=c(1920,1),FREQ=1)
)

#load model and model data
myModel<-LOAD_MODEL(modelText=myModelDefinition)
myModel<-LOAD_MODEL_DATA(myModel, myModelData)

#estimate model
myModel<-ESTIMATE(myModel)

#we want an arbitrary value of 66 on Consumption 'cn' in 1940 and 78 in 1941
#we want an arbitrary value of 77 on GNP 'y' in 1940 and 98 in 1941
kleinTargets<-list(
  cn = TSERIES(66,78,START=c(1940,1),FREQ=1),
  y  = TSERIES(77,98,START=c(1940,1),FREQ=1)
)

#Then, we can perform the model endogenous targeting by using Government Wage Bill 'w2'
#and Government Expenditure 'g' as
#INSTRUMENT in the years 1940 and 1941:
myModel<-RENORM(myModel
                    ,INSTRUMENT = c('w2','g')
                    ,TARGET = kleinTargets
                    ,TSRANGE = c(1940,1,1941,1)
                    ,simIterLimit = 100
)


with(myModel,TABIT(modelData$w2,
                   renorm$INSTRUMENT$w2,
                   modelData$g,
                   renorm$INSTRUMENT$g))

#  Date, Prd., modelData$w2, renorm$INSTRUMENT$w2, modelData$g, renorm$INSTRUMENT$g
# etc.
#  1938, 1   ,          7.7,                     ,          13,
#  1939, 1   ,          7.8,                     ,        14.4,
#  1940, 1   ,            8,              7.41333,        15.4,             16.1069
#  1941, 1   ,          8.5,               9.3436,        22.3,             22.6599
```

```
#So, if we want to achieve on Consumption 'cn'
#an arbitrary simulated value of 66
#in 1940 and 78 in 1941, and if we want
#to achieve on GNP 'y' an arbitrary
#simulated value of 77 in 1940 and 98 in 1941,
#we need to change exogenous 'w2' from 8 to 7.41
#in 1940 and from 8.5 to 9.34 in 1941,
#and we need to change exogenous 'g'
#from 15.4 to 16.1 in 1940 and from 22.3 to 22.66 in 1941

#Let's verify:
#create a new model
kleinRenorm<-myModel

#get instruments to be used
newInstruments=myModel$renorm$INSTRUMENT

#change exogenous by using new instruments
kleinRenorm$modelData<-within(kleinRenorm$modelData,
                {
                  w2[[1940,1]]=newInstruments$w2[[1940,1]]
                  w2[[1941,1]]=newInstruments$w2[[1941,1]]
                  g[[1940,1]] =newInstruments$g[[1940,1]]
                  g[[1941,1]] =newInstruments$g[[1941,1]]
                 }
               )
#users can also replace last two commands with:
#kleinRenorm$modelData<-kleinRenorm$renorm$modelData

#simulate the new model
kleinRenorm<-SIMULATE(kleinRenorm
                     ,TSRANGE=c(1940,1,1941,1)
                     ,simConvergence=0.00001
                     ,simIterLimit=100
                     )
#Simulation: 100.00%
#...SIMULATE OK

#verify targets are achieved
with(kleinRenorm$simulation,
     TABIT(cn,y)
     )

#Date, Prd., cn        , y
#1940,  1  , 66.01116  , 77.01772
#1941,  1  , 78.02538  , 98.04121



######################################################################


#now use 'i' endogenous variable as an instrument
```

```
#first, define the related exogenous constant adjustment
myCA<-list(i = myModel$modelData$i*0+0.1)

#run renorm with endogenous 'i' as instrument
myModel<-RENORM(myModel
                ,INSTRUMENT = c('w2','i')
                ,TARGET = kleinTargets
                ,TSRANGE = c(1940,1,1941,1)
                ,simIterLimit = 100
                ,ConstantAdjustment = myCA
)

#get the values of the constant adjustment for the endogenous 'i'
#in 1940-1941 that allow achieving the target values for 'cn' and 'y'
myModel$renorm$ConstantAdjustment
#$i
#Time Series:
#Start = 1920
#End = 1941
#Frequency = 1
# [1] 0.1000000 0.1000000 0.1000000 ...
#[20] 0.1000000 0.7069039 0.4388811

#these values are also reported in the INSTRUMENT output list
myModel$renorm$INSTRUMENT$i
#Time Series:
#Start = 1940
#End = 1941
#Frequency = 1
#[1] 0.7069039 0.4388811
```

SEMIANNUAL *Semiannual (Dis)Aggregation*

Description

This function returns a semi-annual (dis)aggregated time series, by using as input an annual, quarterly, monthly or daily time series.

Usage

```
SEMIANNUAL(x = NULL, fun = NULL, avoidCompliance = FALSE, ...)
```

Arguments

x	Input time series that must satisfy the compliance control check defined in is.bimets.

fun — Only for daily or monthly or quarterly input time series:
STOCK: the value of the input time series in the last observation of a half-year is assigned to the same half-year of the output time series.
NSTOCK: the value of the input time series in the last non-missing observation of a half-year is assigned to the same half-year of the output time series.
SUM: the sum of input observations in a half-year is assigned to the same half-year of the output time series.
NSUM: the sum of input non-missing observations in a half-year is assigned to the same half-year of the output time series.
AVE: the average of input observations in a half-year is assigned to the same half-year of the output time series.
NAVE: the average of input non-missing observations in a half-year is assigned to the same half-year of the output time series.

Only for annual input time series:
NULL: (default) the output value of each half-year observation is set equal to the value of the input observation the half-year belongs to (i.e. duplicated values over the period)
INTERP_END: the value of the input time series in a period is copied into the last half-year of the output time series that lies in the same period. Other values are calculated by linear interpolation.
INTERP_CENTER: the value of the input time series in a period is copied into the median half-year of the output time series that lies in the same period. Other values are calculated by linear interpolation.
INTERP_BEGIN: the value of the input time series in a period is copied into the first half-year of the output time series that lies in the same period. Other values are calculated by linear interpolation.

avoidCompliance — If TRUE, compliance control check of input time series will be skipped. See is.bimets

... — Backward compatibility.

Value

This function returns a semi-annual BIMETS time series.

See Also

ANNUAL
QUARTERLY
MONTHLY
DAILY

Examples

```
#TS QUARTERLY TO SEMIANNUAL
n<-14
```

```
xArr<-(n:0)
ts1<-TSERIES(xArr,START=c(2000,1),FREQ='Q')
print(SEMIANNUAL(ts1,fun='NAVE'))

#TS ANNUAL TO SEMIANNUAL
ts1<-TSERIES((1:10),START=c(2000,1),FREQ=1)
print(SEMIANNUAL(ts1,fun='INTERP_END'))
```

SIMULATE *Simulation of a BIMETS model*

Description

The simulation of an econometric model basically consists in solving the system of the equations describing the model for each time period in the specified time interval. Since the equations may not be linear in the variables, and since the graph derived from the incidence matrix may be cyclic, the usual methods based on linear algebra are not applicable. The simulation must be solved by using an iterative algorithm.

BIMETS simulation capabilities support:

- *Static simulations*: a static multiple equation simulation, in which the historical values for the lagged endogenous variables are used in the solutions of subsequent periods;

- *Dynamic simulations*: a dynamic simulation, in which the simulated values for the lagged endogenous variables are used in the solutions of subsequent periods;

- *Residuals check*: a single period, single equation simulation; simulated time series in output are just the computation of the RHS (right-hand-side) of their equation, by using the historical values of the involved time series and by accounting for error autocorrelation and PDLs, if any;

- *Forecast simulations*: similar to dynamic simulation, but during the initialization of the iterative algorithm the starting values of endogenous variables in a period are set equal to the simulated values of the previous period. This allows the simulation of future endogenous observations, i.e. the forecast;

- *Stochastic Simulation*: see STOCHSIMULATE;

- *Partial or total exogenization of endogenous variables*: in the provided time interval (i.e. partial exog.) or in whole simulation time range (i.e. total exog.), the values of the selected endogenous variables can be definitely set equal to their historical values, by excluding their equations from the iterative algorithm of simulation;

- *Constant adjustment of endogenous variables (add-factors)*: adds up a new exogenous time series - the "constant adjustment" - in the equation of the selected endogenous variables.

- *Gauss-Seidel* and *Newton-Raphson* simulation algorithms: the Gauss-Seidel algorithm is simple, robust, and works well for many backward-looking macro-econometric models. Equations are

evaluated as-is in a proper order until the convergence, if any, is verified on feedback variables. It is slower than Newton-Raphson algorithms for a very low convergence criterion, and fails to find a convergence for a small set of econometric models, even when a convergence exists. The Newton-Raphson algorithm allows users to solve a broader set of macro-econometric models than the Gauss-Seidel algorithm. Moreover, it is usually faster than the Gauss-Seidel algorithm (on modern computers, users must simulate an extensive econometric model with a low convergence criterion to appreciate the speedup). This type of algorithm requires the construction and the inversion of the Jacobian matrix for the feedback variables; thus, in some scenarios, numerical issues can arise, and users are required to manually exclude some feedback variables from the Jacobian matrix by using the `JacobianDrop` argument of the `SIMULATE` procedure.

In details, the generic model suitable for simulation in BIMETS can be written as:

$y_1 = f_1(\bar{x}, \bar{y})$
...
$y_n = f_n(\bar{x}, \bar{y})$

being:

n the number of equations in the model;
$\bar{y} = [y_1, ..., y_n]$ the `n`-dimensional vector of the endogenous variables;
$\bar{x} = [x_1, ..., x_m]$ the `m`-dimensional vector of the exogenous variables;
$f_i(...), i = 1..n$ any kind of functional expression able to be written by using the `MDL` syntax;

As described later on, in BIMETS a modified Gauss-Seidel iterative algorithm, or a Newton-Raphson algorithm, can solve the system of equations. The convergence properties may vary depending on the model specifications. In some conditions, the algorithm may not converge for a specific model or a specific set of data.

A convergence criterion and a maximum number of iterations to be performed are provided by default. Users can change these criteria by using the `simConvergence` and `simIterLimit` arguments of the `SIMULATE` function.

The general conceptual scheme of the simulation process (for each time period) is the following:

1. initialize the solution for the current simulation period;
2. iteratively solve the system of equations;
3. save the solution, if any;

Step 2 means that for each iteration, the operations are:

2.1 update the values of the current endogenous variables;
2.2 verify that the convergence criterion is satisfied or that the maximum number of allowed iterations has been reached;

The initial solution for the iterative process (step 1) can be given alternatively by:

- the historical value of the endogenous variables for the current simulation period (the default);

- the simulated value of the endogenous variables from the previous simulation period (this alternative is driven by the `simType='FORECAST'` argument of the `SIMULATE` function);

In the "dynamic" simulations (i.e. simulations performed by using either the default `simType='DYNAMIC'` or the `simType='FORECAST'`), whenever lagged endogenous variables are needed in the computation, the simulated values of the endogenous variables $\bar{y}$ assessed in the previous time periods are used. In this case, the simulation results in a given time period depend on the simulation results in the previous time periods. This kind of simulation is defined as "multiple equation, multiple period".

As an alternative, the actual historical values can be used in the "static" simulations (i.e. simulations performed by using `simType='STATIC'`) rather than simulated values whenever lagged endogenous variables are needed in the computations. In this case, the simulation results in a given time period do not depend on the simulation results in the previous time periods. This kind of simulation is defined as "multiple equation, single period".

The last simulation type available is the residual check (`simType='RESCHECK'`). With this option, a "single equation, single period" simulation is performed. In this case, no iteration must be carried out. The endogenous variables are assessed for each time period by using historical values for each variable on the right-hand side of the equation, for both lagged and current periods. This kind of simulation helps debug and check of the logical coherence of the equations and the data, and can be used as a simple tool to compute the add-factors.

The debugging of the logical coherence of equations and data is carried out through a *Residual Check* procedure.

It consists of the following steps:

1. add another exogenous variable - the constant adjustment - to every equation of the model, both behavioral and technical identity (i.e. by using the `ConstantAdjustment` argument of the `SIMULATE` function);
2. fill in with the estimated residuals all the constant adjustments for the behavioral equations;
3. fill in with zeroes the constant adjustments for the technical identities;
4. perform a simulation of the model with the `simType='RESCHECK'` option;
5. compute the difference between the historical and the simulated values for all the endogenous variables;
6. check whether all the differences assessed in step 5 are zero in whole time range;

If a perfect tracking of the history is obtained, then the equations have been written coherently with the data, otherwise a simulated equation not tracking the historical values is an unambiguous symptom of data inconsistent with the model definition.

Aside from the residual check, the add-factors constitute an important tool to significantly improve the accuracy of forecasts made through an econometric model. Considering the following model:

$y_1 = f_1(\bar{x}, \bar{y}) + z_1$
...
$y_n = f_n(\bar{x}, \bar{y}) + z_n$

the add-factors $\bar{z} = [z_1, ..., z_n]$ can be interpreted as estimates of the disturbance terms' future values or as adjustments of the intercepts in each equation. These add-factors round out the forecasts, by summarizing the effects of all the systematic factors not included in the model. One choice for the computation of the add-factors is given by past estimation residuals and past forecast errors or by an average of these errors. This consideration suggests an easy way of computing the add-factors:

1. add the constant adjustments to every equation of the model, both behavioral and technical identity;
2. fill in with zeroes all the constant adjustments;
3. solve the model, with the `simType='RESCHECK'` option, in a time interval including some periods beyond the estimation sample;
4. compute the difference between the historical and the simulated values for each the endogenous variables;
5. average, or process in a suitable way, the difference arising from point 4 in the time periods beyond the estimation sample to compute the constant value to be used as an add-factor in the following forecasting exercises;

Please note that, in the case of equation that presents an LHS function, the add-factor will be added before the application of the inverse function, i.e., during the simulation, the following:

$g_1(y_1) = f_1(\bar{x}, \bar{y}) + z_1$
...
$g_n(y_n) = f_n(\bar{x}, \bar{y}) + z_n$

will be solved as:

$y_1 = g_1^{-1}(f_1(\bar{x}, \bar{y}) + z_1)$
...
$y_n = g_n^{-1}(f_n(\bar{x}, \bar{y}) + z_n)$

If a linear dependence between the simulated endogenous and the add-factor is preferred, users can manually insert an auxiliary equation w_i into the model definition, e.g. the following:

$g_i(y_i) = f_i(\bar{x}, \bar{y})$

can be replaced by:

$w_i = f_i(\bar{x}, \bar{y})$
$y_i = g_i^{-1}(w_i)$

During the simulation, the add-factors (if requested by the user) will be applied as in the following:

$w_i = f_i(\bar{x}, \bar{y}) + v_i$
$y_i = g_i^{-1}(w_i) + z_i$

given v_i, z_i as add-factors and the linear dependence from z_i and y_i.

THE OPTIMAL REORDERING

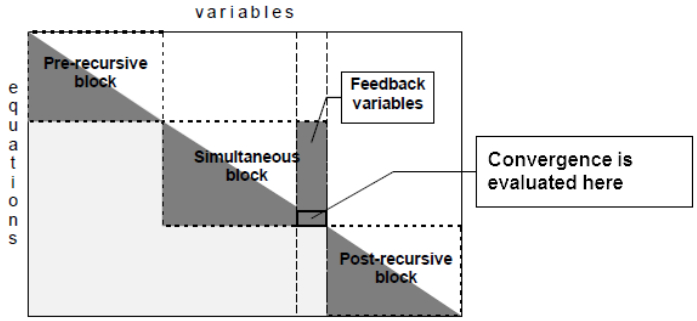
In fact, the simulation process takes advantage of an appropriate ordering of the equations to increase the performances by iteratively solving only one subset of equations, while the others are solved straightforwardly. *"...a different ordering of the equations can substantially affect the speed of convergence of the algorithm; indeed some orderings may produce divergence. The less feedback there is, the better the chances for fast convergence..."* - Don, Gallo - Solving large sparse systems of equations in econometric models - Journal of Forecasting 1987.

The `LOAD_MODEL` function builds the model's incidence matrix, then defines the proper equation reordering. The incidence matrix is built from the equations of the model; it is a square matrix in which each row and each column represent an endogenous variable. If the `(i,j)` element is equal to 1 then in the model definition the current value of the endogenous variable referred by the `i`-row depends directly from the current value of the endogenous variable referred by the `j`-column. The reader can see an incidence matrix example in the section `"BIMETS package"` of this manual wherein the content of the `kleinModel$incidence_matrix` variable is printed out.

In econometric models, the incidence matrix is usually very sparse. Only a few of the total set of endogenous variables are used in each equation. In this situation, ordering the equation in a particular sequence will lead to a sensible reduction of the number of iterations needed to achieve convergence. Reordering the equations is equivalent to rearranging rows and columns of the incidence matrix. In this way, the incidence matrix might be made lower triangular for a subset of the equations. For this subset, an endogenous variable determined in a specific equation has no *incidence* in any equation above it, although the same variable might have incidence in equations below it. Such a subset of equations is called recursive. Recursive systems are easy to solve. It is only necessary to solve each equation once if this is done in the proper order. On the other hand, it is unlikely for whole model to be recursive. Indeed the incidence graph is often cyclic, as in the Klein's model that presents the following circular dependecies in the incidence matrix: `p <- w1 <- y <- p` as shown in the `"BIMETS package"` section figure.

For a subset of the equations, some 1's will occur in the upper triangle of the incidence matrix for all possible orderings. Such subset of equations is called *simultaneous*. To solve the endogenous variables in the simultaneous block of equations, an iterative algorithm has to be used. Nevertheless, the equations in the simultaneous block may be ordered so that the pattern of the 1's in the upper triangle of the incidence matrix forms a spike. The variables corresponding to the 1's in the upper triangle are called *feedback* variables.

A qualitative graphical example of an ordered incidence matrix is given in the following figure. The white areas are all 0's, the gray areas contain 0's and 1's. The 1's in the light gray areas refer to variables already evaluated in previous blocks, therefore they are known terms within the block. The 1's in the dark gray areas refer to variables evaluated within the block.



The final pattern of an incidence matrix after the equation reordering generally features three blocks:

1. a recursive block (the pre-recursive block);
2. a simultaneous block;
3. another recursive block (the post-recursive block);

As said, the pre-recursive and the post-recursive blocks are lower triangular. Therefore the corresponding equations are solvable with a cascade substitution with no iteration. Just the simultaneous equations set needs an iterative algorithm to be solved. It is important to say that the convergence criterion may also be applied to these variables only: when the feedback variables converge, the rest of the simultaneous variables also do.

BIMETS builds and analyzes the model's incidence matrix, and then it orders the equations in pre-recursive, post-recursive and simultaneous blocks. The simultaneous block is then analyzed in order to find a minimal set of feedback variables. This last problem is known to be NP-complete (Ref: Garey, Johnson - *Computers and Intractability: a Guide to the Theory of NP-completeness* - San Francisco, Freeman 1979).

The optimal reordering of the model equations is programmatically achieved through the use of an iterative algorithm applied to the incidence matrix that can produce 4 ordered lists of endogenous variables:

1. `vpre` is the ordered list containing the names of the endogenous pre-recursive variables to be sequentially computed (once per simulation period, by using their `EQ>` definition in the `MDL`) before the simulation iterative algorithm takes place;
2. `vsim` (the simultaneous block) is the ordered list containing the names of the endogenous variables to be sequentially computed during each iteration of the simulation iterative algorithm;
3. `vfeed` is the list containing the names of the endogenous feedback variables; generally `vfeed` are the last variables of the ordered `vsim` list;
4. `vpost` is the ordered list containing the names of the endogenous post-recursive variables to be sequentially computed (once per simulation period, by using their `EQ>` definition in the `MDL`) after the simulation iterative algorithm has found a solution in the simultaenous block;

If equations are reordered, the previous conceptual scheme is modified as follows:

- initialize the solution for the current simulation period;
- compute the pre-recursive equations (i.e. the equation of the endogenous variables in the `vpre`

ordered list);
- iteratively compute the system of simultaneous equations (i.e. the equation of the endogenous variables in the `vsim` ordered list); for each iteration update the values of the current endogenous variables and verify that the convergence criterion is satisfied on the feedback variables `vfeed` or that the maximum number of iterations has been reached;
- compute the post-recursive equations (i.e. the equation of the endogenous variables in the `vpost` ordered list);
- save the solutions;

THE SIMULATION ALGORITHMS

Given x_j the j-exogenous variable, $j = 1..m$, and $y_{i,k}$ the i-endogenous variable in the simultaneous block, at the iteration k, with $i = 1..n$ the position of the equation in a reordered model, the modified Gauss-Seidel method takes for the approximation of the endogenous variable $y_{i,k}$ the solution of the following:

$$y_{i,k} = f_i(x_1, ..., x_m, y_{1,k}, ..., y_{i-1,k}, y_{i,k-1}, ..., y_{n,k-1})$$

Newton-Raphson's methods can be seen as an extension of the modified Gauss-Seidel algorithm, and a further step is required: in Newton-Raphson, feedback variables are updated not by using their model equations, but by using the inverse of the Jacobian matrix and the following assignment:

$$\bar{y}_k^F \leftarrow \bar{y}_{k-1}^F + (I - J)^{-1}[\bar{y}_k^F - \bar{y}_{k-1}^F]$$

given the vector of feedback variables values $\bar{y}_k^F = [y_{n-F+1,k}, ..., y_{n,k}]$ at iteration k, the identity matrix I, and the Jacobian matrix J, with $I, J \in R^{F,F}$ and F equal to the number of feedback variables for the given model. Please note that the modified Gauss-Seidel algorithm can be seen as a reduced form of a Netwotn algorithm, given $J = 0$.

In Newton-Raphson methods, the Jacobian matrix J is calculated as follows:
1 - shock the feedback variables one at a time by a small amount;
2 - for each shocked feedback variable, evaluate the shocked solution of the simultaneous block;
3 - calculate the derivatives (i.e. the column in the Jacobian matrix related to the shocked feedback variable) using the difference quotients between the shocked and the base solution of the simultaneous block.

As said, the convergence is always tested at each iteration's end on the feedback variables.

Newton-Raphson methods on a reordered model require the calculation of the Jacobian matrix on the feedback endogenous variables, i.e. at least $F + 2$ iterations per simulation period, with F as the number of feedback variables. For large models (i.e. more than 30 feedback variables) if the overall required convergence is greater than $10^{-6}\%$ the speedup over the Gauss-Seidel method is small or negative, if the Jacobian matrix is recalculated at each iteration. Moreover, the Gauss-Seidel method does not require a matrix inversion; therefore, it is more robust against algebraical and numerical issues. For small models, both methods are fast on modern computers. On the other hand, Gausse-Seidel fails to find a convergence for a small set of econometric models, even when

a convergence exists. In general, given a system of equations $Ax = b$, with $x, b \in R^n, n > 0$ and $A \in R^{n,n}$, the Gauss-Seidel algorithm is known to converge if either:
- A is symmetric positive-definite;
- A is strictly or irreducibly diagonally dominant.

To improve simulation speed, BIMETS evaluates the Newton-Raphson algorithm performance during simulation, and, at each iteration, a new Jacobian matrix is calculated *only if* the convergence speed is slower than a predefined threshold. In a vectorized simulation (e.g., STOCHSIMULATE, OPTIMIZE, RENORM), the Jacobian matrix is calculated only on the unperturbed model, then applied to all realizations.

The simulation of a non-trivial model, if computed by using the same data but on different hardware, software or numerical libraries, produces numerical differences. Therefore a convergence criterion smaller than $10^{-7}\%$ frequently leads to a local solution.

See *Numerical methods for simulation and optimal control of large-scale macroeconomic models - Gabay, Nepomiastchy, Rachidi, Ravelli - 1980* for further information.

Usage

```
SIMULATE( model=NULL,
          simAlgo='GAUSS-SEIDEL',
          TSRANGE=NULL,
          simType='DYNAMIC',
          simConvergence=0.01,
          simIterLimit=100,
          ZeroErrorAC=FALSE,
          BackFill=0,
          Exogenize=NULL,
          ConstantAdjustment=NULL,
          verbose=FALSE,
          verboseSincePeriod=0,
          verboseVars=NULL,
          MULTMATRIX=FALSE,
          RENORM=FALSE,
          TARGET=NULL,
          INSTRUMENT=NULL,
          MM_SHOCK=0.00001,
          STOCHSIMULATE=FALSE,
          StochStructure=NULL,
          StochReplica=100,
          StochSeed=NULL,
          OPTIMIZE=FALSE,
          OptimizeBounds=NULL,
          OptimizeRestrictions=NULL,
          OptimizeFunctions=NULL,
          quietly=FALSE,
          RESCHECKeqList=NULL,
          JACOBIAN_SHOCK=1e-4,
```

```
JacobianDrop=NULL,
avoidCompliance=FALSE,
...)
```

Arguments

`model` The BIMETS model object to be simulated. The simulation requires that all the model behaviorals, if any, have been previously estimated: all the behavioral coefficients (i.e. the regression coefficients and the autoregression coefficients for the errors, if any) must be numerically defined in the model object. (see also ESTIMATE)

`simAlgo` The simulation algorithm to be used to solve the system of model equations for each time period in the simulation TSRANGE.
The options are:
GAUSS-SEIDEL: (default) the Gauss-Seidel algorithm is simple, robust, and works well for many backward-looking macro-econometric models. Equations are evaluated as-is in a proper order until the convergence, if any, is verified on feedback variables (see *THE SIMULATION ALGORITHMS* section up here). It is slower than Newton-Raphson algorithms for a very low convergence criterion, and fails to find a convergence for a small set of econometric models, even when a convergence exists. In general, given a system of equations $Ax = b$, with $x, b \in R^n, n > 0$ and $A \in R^{n,n}$, the Gauss-Seidel algorithm is known to converge if either:
- A is symmetric positive-definite;
- A is strictly or irreducibly diagonally dominant.
NEWTON: the Newton-Raphson algorithm allows users to solve a broader set of macro-econometric models than the Gauss-Seidel algorithm. Moreover, it is faster than the Gauss-Seidel algorithm in most cases (on modern computers, users must simulate an extensive econometric model with a low convergence criterion to appreciate the speedup). This type of algorithm requires the construction and the inversion of the Jacobian matrix for the feedback variables; thus, in some scenarios (e.g., equations with unverified IF> condition, implicit exogenizations, etc.), numerical issues can arise, and users are required to manually exclude some feedback variables from the Jacobian matrix by using the JacobianDrop argument. To improve simulation speed, BIMETS evaluates the Newton-Raphson algorithm performance during simulation, and, at each iteration, a new Jacobian matrix is calculated *only if* the convergence speed is slower than a predefined threshold. In a vectorized simulation (e.g., STOCHSIMULATE, OPTIMIZE, RENORM), the Jacobian matrix is calculated only on the unperturbed model, then applied to all realizations. As in the Gauss-Seidel algorithm, the convergence is evaluated on feedback variables
FULLNEWTON: similar to the "base" Newton case, but when using this option, a different Jacobian matrix is calculated for each stochastic realization in a vectorized simulation (e.g., STOCHSIMULATE, OPTIMIZE, RENORM). All the Jacobian matrices are referred to as the Jacobian array. Generally, a "full" Newton simulation requires more computational time, mainly due to the increased number of Jacobian matrices to be calculated. However, the "base" Newton al-

gorithm fails to converge when the model is highly non-linear and perturbances are significant (see examples in STOCHSIMULATE help page). Therefore, a "full" Newton algorithm is required in these cases.

`TSRANGE` The time range of the simulation, as a four dimensional numerical array, i.e. `TSRANGE=c(start_year, start_period, end_year, end_period)`

`simType` The simulation type requested:
DYNAMIC: the default, whenever lagged endogenous variables are needed in the computations, the simulated values of the endogenous variables evaluated in the previous time periods are used;
STATIC: rather than the simulated values, the actual historical values are used whenever lagged endogenous variables are needed in the computations;
FORECAST: similar to the `'DYNAMIC'` option, but the initial solutions for the iterative process are given by the simulated values of the endogenous variables in the previous period. In this case there is no need for historical values of the endogenous variables in whole provided `TSRANGE`;
RESCHECK: in this case there is no iteration to carry out. The endogenous variables are evaluated for each single time period by using the historical values for all the variables on the right-hand side of the equation, both lagged and current period;

`simConvergence` The percentage convergence value requested for the iterative process, which stops when the percentage difference of all the feedback variables between iterations is less than `simConvergence` in absolute value

`simIterLimit` The value representing the maximum number of iterations to be performed. The iterative procedure will stop when `simIterLimit` is reached or the feedback variables satisfy the `simConvergence` criterion

`ZeroErrorAC` If `TRUE` it zeroes out all the autoregressive terms, if any, in the behavioral equations

`BackFill` Defined as an `integer`, it is the length of historical data prior to the simulation `TSRANGE` to be saved along with the solutions

`Exogenize` A named list that specifies the endogenous variables to be exogenized. During the simulation and inside the provided time range, the exogenized endogenous variables will be assigned to their historical values. List names must be the names of the endogenous variables to be exogenized; each element of this list contains the time range of the exogenization for the related endogenous variable, in the form of a 4-dimensional integer array, i.e. start_year, start_period, end_year, end_period. A list element can also be assigned `TRUE`: in this case the related endogenous variable will be exogenized in whole simulation `TSRANGE` (see example)

`ConstantAdjustment`
A named list that specifies the constant adjustments (i.e. add factors) to be added to the selected equations of the model. Each constant adjustment can be see as a new exogenous variable added to the equation of the specified endogenous variable. The list names are the names of the involved endogenous variables; each element of this is list contains the time series to be added to the equation of the related endogenous variable. Each provided time series must verify the compliance control check defined in `is.bimets` (see example)

verbose — If TRUE some verbose output will be activated. Moreover the values of all endogenous variables will be printed out during each iteration of the convergence algorithm for all time periods in the simulation TSRANGE

verboseSincePeriod — An integer that activates the verbose output, during the iterative process, only after the provided number of simulation periods

verboseVars — A character array with the names of the endogenous variables for which the verbose output will be activated in the iterative process

MULTMATRIX — It is TRUE when the parent call is a multiplier matrix operation requested by a MULTMATRIX procedure

RENORM — It is TRUE when the parent call is an endogenous targeting operation requested by a RENORM procedure

TARGET — see MULTMATRIX

INSTRUMENT — see MULTMATRIX

MM_SHOCK — see MULTMATRIX

STOCHSIMULATE — It is TRUE when the parent call is a stochastic simulation requested by a STOCHSIMULATE operation

StochStructure — The list() that defines the disturbance structure applied to the model. See STOCHSIMULATE

StochReplica — An integer value that sets the number of stochastic simulation replications to be performed. See STOCHSIMULATE

StochSeed — A number used to initialize the pseudo-random number generator. It can be useful in order to replicate stochastic results. See STOCHSIMULATE

OPTIMIZE — It is TRUE when the parent call is an optimize operation requested by a procedure

OptimizeBounds — see OPTIMIZE

OptimizeRestrictions — see OPTIMIZE

OptimizeFunctions — see OPTIMIZE

quietly — If TRUE, information messages will be suppressed, e.g. an information message will be printed out if any time series has a missing value in the extended TSRANGE, that is the simulation TSRANGE backward extended by model$max_lag periods (See LOAD_MODEL for info on model max lag)

RESCHECKeqList — If simType=RESCHECK, by using this argument users can select a subset of target endogenous variables: the simulation will be performed only for the selected variables. It must be provided as an array of endogenous names, e.g. c('endo1','endo2',...)

JACOBIAN_SHOCK — The value of the shock added to feedback variables in the derivative calculation of the Jacobian matrix. The default value is 1e-4 times the value of the feedback variable

JacobianDrop — The array built with feedback variables names to be excluded from the Jacobian matrix calulation

avoidCompliance — If TRUE, compliance control check of model time series will be skipped. See is.bimets

... — Backward compatibility

Value

This function will add a new element named `simulation` into the output BIMETS model object.

The new `simulation` element is a named list; the names of the `simulation` list are the names of the endogenous variables of the model; each element of the `simulation` list contains the simulated time series of the related endogenous variable (see example).

The `simulation` list also contains the `'__SIM_PARAMETERS__'` element that contains the arguments passed to the function call during the latest `SIMULATE` run, e.g. `TSRANGE`, `symType`, `simConvergence`, `symIterLimit`, `Exogenize`, etc.: this data can be helpful in order to replicate the simulation results.

See Also

MDL
LOAD_MODEL
ESTIMATE
STOCHSIMULATE
MULTMATRIX
RENORM
OPTIMIZE
TIMESERIES
BIMETS indexing
BIMETS configuration

Examples

```
#define model
myModelDefinition<-
"MODEL
COMMENT> Klein Model 1 of the U.S. Economy

COMMENT> Consumption
BEHAVIORAL> cn
TSRANGE 1921 1 1941 1
EQ> cn =  a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)
COEFF> a1 a2 a3 a4

COMMENT> Investment
BEHAVIORAL> i
TSRANGE 1921 1 1941 1
EQ> i = b1 + b2*p + b3*TSLAG(p,1) + b4*TSLAG(k,1)
COEFF> b1 b2 b3 b4

COMMENT> Demand for Labor
BEHAVIORAL> w1
TSRANGE 1921 1 1941 1
EQ> w1 = c1 + c2*(y+t-w2) + c3*TSLAG(y+t-w2,1) + c4*time
```

```
COEFF> c1 c2 c3 c4

COMMENT> Gross National Product
IDENTITY> y
EQ> y = cn + i + g - t

COMMENT> Profits
IDENTITY> p
EQ> p = y - (w1+w2)

COMMENT> Capital Stock
IDENTITY> k
EQ> k = TSLAG(k,1) + i

END"

#define model data
myModelData<-list(
   cn
   =TIMESERIES(39.8,41.9,45,49.2,50.6,52.6,55.1,56.2,57.3,57.8,55,50.9,
               45.6,46.5,48.7,51.3,57.7,58.7,57.5,61.6,65,69.7,
               START=c(1920,1),FREQ=1),
   g
   =TIMESERIES(4.6,6.6,6.1,5.7,6.6,6.5,6.6,7.6,7.9,8.1,9.4,10.7,10.2,9.3,10,
               10.5,10.3,11,13,14.4,15.4,22.3,
               START=c(1920,1),FREQ=1),
   i
   =TIMESERIES(2.7,-.2,1.9,5.2,3,5.1,5.6,4.2,3,5.1,1,-3.4,-6.2,-5.1,-3,-1.3,
               2.1,2,-1.9,1.3,3.3,4.9,
               START=c(1920,1),FREQ=1),
   k
   =TIMESERIES(182.8,182.6,184.5,189.7,192.7,197.8,203.4,207.6,210.6,215.7,
               216.7,213.3,207.1,202,199,197.7,199.8,201.8,199.9,
               201.2,204.5,209.4,
               START=c(1920,1),FREQ=1),
   p
   =TIMESERIES(12.7,12.4,16.9,18.4,19.4,20.1,19.6,19.8,21.1,21.7,15.6,11.4,
               7,11.2,12.3,14,17.6,17.3,15.3,19,21.1,23.5,
               START=c(1920,1),FREQ=1),
   w1
   =TIMESERIES(28.8,25.5,29.3,34.1,33.9,35.4,37.4,37.9,39.2,41.3,37.9,34.5,
               29,28.5,30.6,33.2,36.8,41,38.2,41.6,45,53.3,
               START=c(1920,1),FREQ=1),
   y
   =TIMESERIES(43.7,40.6,49.1,55.4,56.4,58.7,60.3,61.3,64,67,57.7,50.7,41.3,
               45.3,48.9,53.3,61.8,65,61.2,68.4,74.1,85.3,
               START=c(1920,1),FREQ=1),
   t
   =TIMESERIES(3.4,7.7,3.9,4.7,3.8,5.5,7,6.7,4.2,4,7.7,7.5,8.3,5.4,6.8,7.2,
               8.3,6.7,7.4,8.9,9.6,11.6,
               START=c(1920,1),FREQ=1),
   time
   =TIMESERIES(NA,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,
```

```
              START=c(1920,1),FREQ=1),
   w2
   =TIMESERIES(2.2,2.7,2.9,2.9,3.1,3.2,3.3,3.6,3.7,4,4.2,4.8,5.3,5.6,6,6.1,
              7.4,6.7,7.7,7.8,8,8.5,
              START=c(1920,1),FREQ=1)
 )

#load model and model data
myModel<-LOAD_MODEL(modelText=myModelDefinition)
myModel<-LOAD_MODEL_DATA(myModel,myModelData)

#estimate model
myModel<-ESTIMATE(myModel)

#DYNAMIC SIMULATION

#simulate model
myModel<-SIMULATE(myModel
                 ,TSRANGE=c(1923,1,1941,1)
                 ,simConvergence=0.00001
                 ,simIterLimit=100
  )
#
#Simulation:    100.00%
#...SIMULATE OK

#get simulated time series "cn" and "y"
TABIT(myModel$simulation$cn)
#
#      Date, Prd., myModel$simulation$cn
#
#      1923, 1   ,     50.338
#      1924, 1   ,    55.6994
#      1925, 1   ,    56.7111
#      ...
#      1940, 1   ,    66.7799
#      1941, 1   ,     75.451
#

TABIT(myModel$simulation$y)
#
#      Date, Prd., myModel$simulation$y
#
#      1923, 1   ,    56.0305
#      1924, 1   ,    65.8526
#      1925, 1   ,     64.265
#      ...
#      1940, 1   ,    76.8049
#      1941, 1   ,    93.4459
#

#get latest simulation parameters
print(myModel$simulation$'__SIM_PARAMETERS__')
```

```
#$TSRANGE
#[1] 1923    1 1941    1
#
#$simType
#[1] "DYNAMIC"
#
#$simConvergence
#[1] 1e-05
#
#$simIterLimit
#[1] 100
#
#$ZeroErrorAC
#[1] FALSE
#
#...etc etc


#########################################################
#RESCHECK SIMULATION

#simulate model
myModel<-SIMULATE(myModel
                  ,simType='RESCHECK'
                  ,TSRANGE=c(1923,1,1941,1)
                  ,simConvergence=0.00001
                  ,simIterLimit=100
 )
#
#Simulation:    100.00%
#...SIMULATE OK

#get consumption simulated vs historical differences
TABIT(myModel$simulation$cn-myModel$modelData$cn)
#
#      Date, Prd., myModel$simulation$cn - myModel$modelData$cn
#
#      1923, 1   ,    1.56574
#      1924, 1   ,   0.493503
#      1925, 1   , -0.0076079
#      ...
#      1939, 1   ,  -0.989201
#      1940, 1   ,  -0.785077
#      1941, 1   ,    2.17345
#

#########################################################
#FORECAST GNP in 1942 and 1943

#we need to extend exogenous variables in 1942 and 1943
myModel$modelData$w2   = TSEXTEND(myModel$modelData$w2,  UPTO=c(1943,1))
myModel$modelData$t    = TSEXTEND(myModel$modelData$t,   UPTO=c(1943,1))
myModel$modelData$g    = TSEXTEND(myModel$modelData$g,   UPTO=c(1943,1))
```

```
myModel$modelData$time = TSEXTEND(myModel$modelData$time,UPTO=c(1943,1)
                                  ,EXTMODE='LINEAR')

#simulate model
myModel<-SIMULATE(myModel
                  ,simType='FORECAST'
                  ,TSRANGE=c(1940,1,1943,1)
                  ,simConvergence=0.00001
                  ,simIterLimit=100
)
#
#Simulation:    100.00%
#...SIMULATE OK

#get forecasted GNP
TABIT(myModel$simulation$y)
#
#      Date, Prd., myModel$simulation$y
#
#      1940, 1   ,    74.5781
#      1941, 1   ,    94.0153
#      1942, 1   ,    133.969
#      1943, 1   ,    199.913
#

#########################################################
#VERBOSE SIMULATION

myModel<-SIMULATE(myModel
                  ,TSRANGE=c(1923,1,1941,1)
                  ,simConvergence=0.00001
                  ,simIterLimit=100
                  ,verbose=TRUE
                  ,verboseSincePeriod=19
                  ,verboseVars=c('cn')
 )
#CHECK_MODEL_DATA(): warning, there are missing values in series "time".
#
#Simulation:      5.26%
#Prd.    1 convergence reached in iter 34
#Simulation:     10.53%
#Prd.    2 convergence reached in iter 42
#Simulation:     15.79%
#Prd.    3 convergence reached in iter 40
#Simulation:     21.05%
#
#...
#
#Prd.   17 convergence reached in iter 40
#Simulation:     94.74%
#Prd.   18 convergence reached in iter 37
#Simulation:    100.00%
#VPRE eval
```

```
#Prd.   19 Iter    0         cn            69.7
#VSIM eval
#Prd.   19 Iter    1         cn     71.7197097805143
#VSIM eval
#Prd.   19 Iter    2         cn     72.7386970702054
#... etc etc
#Prd.   19 Iter   39         cn     75.4510243396176
#VSIM eval
#Prd.   19 Iter   40         cn     75.4510298916399
#VPOST eval
#Prd.   19 Iter   40         cn      75.4510299
#Prd.   19 convergence reached in iter 40
#
#...SIMULATE OK


##########################################################
#DYNAMIC NEWTON SIMULATION
#WITH EXOGENIZATION AND CONSTANT ADJUSTMENTS

#define exogenization list
#'cn' exogenized in 1923-1925
#'i' exogenized in whole TSRANGE
exogenizeList<-list(
                cn = c(1923,1,1925,1),
                i  = TRUE
)

#define add factor list
constantAdjList<-list(
               cn = TIMESERIES(1,-1,START=c(1923,1),FREQ='A'),
               y  = TIMESERIES(0.1,-0.1,-0.5,START=c(1926,1),FREQ='A')
)

#simulate model
myModel<-SIMULATE(myModel
                  ,simAlgo='NEWTON'
                  ,simType='DYNAMIC'
                  ,TSRANGE=c(1923,1,1941,1)
                  ,simConvergence=0.00001
                  ,simIterLimit=100
                  ,Exogenize=exogenizeList
                  ,ConstantAdjustment=constantAdjList
 )
#SIMULATE(): endogenous variable "cn" has been exogenized from year-period 1923-1 to 1925-1
#SIMULATE(): endogenous variable "i" has been exogenized from year-period 1923-1 to 1941-1
#SIMULATE(): endogenous variable "cn" has a constant adjustment from year-period 1923-1 to 1924-1
#SIMULATE(): endogenous variable "y" has a constant adjustment from year-period 1926-1 to 1928-1
#
#Simulation:    100.00%
#...SIMULATE OK

##########################################################
```

```
#EXAMPLE OF MODEL THAT FAILS GAUSS CONVERGENCE

#define model
myNewtonModelDefinition<-
"
MODEL

COMMENT> Consumption
BEHAVIORAL> cn
TSRANGE 1922 1 1931 1
EQ> cn =  a1 + a2*p + a3*LAG(p,1) + a4*(w1+w2)
COEFF> a1 a2 a3 a4

COMMENT> Investment
BEHAVIORAL> i
TSRANGE 1922 1 1931 1
EQ> i = b1 + b2*p + b3*LAG(p,1) + b4*LAG(k,1)
COEFF> b1 b2 b3 b4

COMMENT> Demand for Labor
BEHAVIORAL> w1
TSRANGE 1922 1 1931 1
EQ> w1 = c1 + c2*(z+y+t-w2) + c3*LAG(z+y+t-w2,1) + c4*time
COEFF> c1 c2 c3 c4

COMMENT> Gross National Product
IDENTITY> y
EQ> y = cn + i + g - t

COMMENT> Simple copy of y in z
IDENTITY> z
EQ> z = cn + i + g - t

COMMENT> Profits
IDENTITY> p
EQ> p = z + y - (w1+w2)

COMMENT> Capital Stock
IDENTITY> k
EQ> k = LAG(k,1) + i

END
"

#add data to model
myModelData$z <- myModelData$y
myNewtonModel<-LOAD_MODEL(modelText=myNewtonModelDefinition)
myNewtonModel<-LOAD_MODEL_DATA(myNewtonModel,myModelData)

#estimate model
myNewtonModel<-ESTIMATE(myNewtonModel)

#GAUSS simulation fails to converge...
```

```
myNewtonModel <- SIMULATE(myNewtonModel,
                        TSRANGE = c(1921, 1, 1930, 1),
                        simConvergence = 1e-7,
                        verbose=TRUE)

#...while NEWTON converge
myNewtonModel <- SIMULATE(myNewtonModel,
                        simAlgo='NEWTON',
                        TSRANGE = c(1921, 1, 1930, 1),
                        simConvergence = 1e-7,
                        verbose=TRUE)

#########################################################
#EXAMPLE OF MODEL THAT REQUIRES
#A VARIABLE EXCLUSION FROM JACOBIAN MATRIX

#define model
myNewtonWithDropModelDefinition<-
"
MODEL

COMMENT> Consumption
BEHAVIORAL> cn
TSRANGE 1922 1 1931 1
EQ> cn =  a1 + a2*p + a3*LAG(p,1) + a4*(w1+w2)
COEFF> a1 a2 a3 a4

COMMENT> Investment
BEHAVIORAL> i
TSRANGE 1922 1 1931 1
EQ> i = b1 + b2*p + b3*LAG(p,1) + b4*LAG(k,1)
COEFF> b1 b2 b3 b4

COMMENT> Demand for Labor
BEHAVIORAL> w1
TSRANGE 1922 1 1931 1
EQ> w1 = c1 + c2*(z+y+t-w2) + c3*LAG(z+y+t-w2,1) + c4*time
COEFF> c1 c2 c3 c4

COMMENT> Gross National Product
IDENTITY> y
EQ> y = cn + i + g - t

COMMENT> Simple copy of y in z
IDENTITY> z
EQ> z = cn + i + g - t

COMMENT> Profits
IDENTITY> p
EQ> p = z + y - (w1+w2)
IF> y < 0

COMMENT> Capital Stock
```

```
IDENTITY> k
EQ> k = LAG(k,1) + i

END
"

#add data to model
myModelData$z <- myModelData$y
myNewtonModel <- LOAD_MODEL(modelText=myNewtonWithDropModelDefinition)
myNewtonModel <- LOAD_MODEL_DATA(myNewtonModel,myModelData)

#estimate model
myNewtonModel <- ESTIMATE(myNewtonModel)

#"p" variable must be removed from Jacobian beacuse of unverified IF>
myNewtonModel <- SIMULATE(myNewtonModel,
                        simAlgo='NEWTON',
                        JacobianDrop='p',
                        TSRANGE = c(1921, 1, 1930, 1),
                        simConvergence = 1e-7,
                        verbose=TRUE)
```

STOCHSIMULATE *Stochastic simulation of a BIMETS model*

Description

The STOCHSIMULATE operation performs a stochastic simulation. The simulation algorithms are the same as those used by the SIMULATE operation.

Forecasts produced by structural econometric models are subject to several sources of error, such as random disturbance term of each stochastic equation, errors in estimated coefficients, errors in forecasts of exogenous variables, errors in preliminary data and mis-specification of the model.

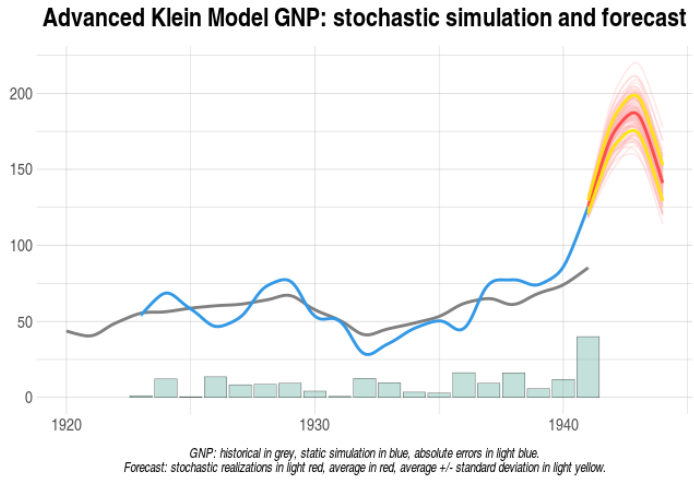
The forecast error depending on the structural disturbances can be analyzed by using the stochastic simulation procedure.

The deterministic simulation is the simultaneous solution of an econometric model obtained by applying, for each stochastic (behavioral) equation, the expected values of the structural disturbances, which are all zero by assumption. In the BIMETS STOCHSIMULATE stochastic simulation, the structural disturbances are given values that have specified stochastic properties. The error terms of the estimated behavioral equation of the model are appropriately perturbed. Identity equations and exogenous variables can be as well perturbed by disturbances that have specified stochastic properties. The model is then solved for each data set with different values of the disturbances. Finally, mean and standard deviation are computed for each simulated endogenous variable.

In terms of computational efficiency, the procedure takes advantage of the fact that multiple datasets are bound together in matrices. Therefore, to achieve a global convergence, the iterative simulation algorithm is executed once for all perturbed datasets. This solution can be viewed as a sort of a SIMD (i.e. Single Instruction Multiple Data) parallel simulation: the `STOCHSIMULATE` function transforms time series into matrices and consequently can easily bind multiple datasets by column. At the same time, a single run ensures a fast code execution. Finally, each column in the output matrices represents a stochastic realization.

By using the `StochStructure` argument of this function, users can define a stochastic structure for the disturbances. For each variable of the model, users can provide a distinct probability distribution for the disturbance, and a specific time range of application. Mean and standard deviation for each simulated endogenous time series will be stored in the `stochastic_simulation` element of the output model object; all the stochastic realizations will be stored in the `simulation_MM` element of the output model object as named matrices.

In the following figure, the advanced Klein model (see code example), has been perturbed during the forecast operation by applying a normal disturbance to the endogenous *Consumption* behavioral `cn` add-factor in year 1942, and a uniform disturbance to the exogenous *Government Expenditure* time series `g` along all the simulation `TSRANGE`. The normal disturbance applied to the `cn` behavioral has a zero mean and a standard deviation equal to the behavioral regression standard error, i.e. `advancedKleinModel$behaviorals$cn$statistics$StandardErrorRegression`, thus roughly replicating the `ESTIMATE` regression error during the current perturbation (not accounting for inter-equations cross-covariance).



At the moment, all the disturbances are i.i.d. and are not transformed into a congruent autoregressive scheme in the case the related perturbed endogenous behavioral presents an autocorrelation for the errors in its `MDL` definition, e.g. `ERROR> AUTO(n)`

Usage

```
STOCHSIMULATE(model=NULL,
              simAlgo='GAUSS-SEIDEL',
              TSRANGE=NULL,
              simType='DYNAMIC',
              simConvergence=0.01,
              simIterLimit=100,
              ZeroErrorAC=FALSE,
              Exogenize=NULL,
              ConstantAdjustment=NULL,
              verbose=FALSE,
              verboseSincePeriod=0,
              verboseVars=NULL,
              StochStructure=NULL,
              StochReplica=100,
              StochSeed=NULL,
              quietly=FALSE,
              RESCHECKeqList=NULL,
              JACOBIAN_SHOCK=1e-4,
              JacobianDrop=NULL,
              avoidCompliance=FALSE,
              ...)
```

Arguments

model	see SIMULATE
simAlgo	see SIMULATE
TSRANGE	see SIMULATE
simType	see SIMULATE
simConvergence	see SIMULATE
simIterLimit	see SIMULATE
ZeroErrorAC	see SIMULATE
Exogenize	see SIMULATE
ConstantAdjustment	see SIMULATE
verbose	see SIMULATE
verboseSincePeriod	see SIMULATE
verboseVars	see SIMULATE
StochStructure	the named `list()` that defines the disturbance structure applied to the model. Each list element must have a name equal to an endogenous or an exogenous model variable. List names define the `INSTRUMENT`. If a list element name is equal to an exogenous variable, the disturbance will

be applied to the related exogenous time series values. If a list element name is equal to an endogenous variable, the disturbance will be applied to the constant adjustment time series (see SIMULATE) of the related endogenous variable.

Each list element must be a named list built with the following three element:

- TSRANGE: the time range wherein the disturbance is active and additive to the related model variable. The TSRANGE must be a 4 numerical array, i.e. TSRANGE=c(start_year, start_period, end_year, end_period) or TSRANGE=TRUE in order to apply the provided disturbance to whole STOCHSIMULATE TSRANGE.

- TYPE: the type of disturbance distribution. At the moment, only normal, i.e. TYPE='NORM', and uniform, i.e. TYPE='UNIF', disturbance distributions are allowed;

- PARS: the parameters that shape the disturbance.

In the case of a TYPE='NORM' distribution, these parameters must contain the mean and the standard deviation of the normal distribution, i.e. PARS=c(mean,sd);

in the case of a TYPE='UNIF' distribution, these parameters must contain the lower and upper bound of the uniform distribution, i.e. PARS=c(min,max);

in the case of a TYPE='MATRIX', these parameters must contain the user-built matrix of realized disturbances, i.e. PARS=matrix, with matrix as a [TSRANGE x StochReplica] matrix.

See example in order to learn how to build a compliant stochastic structure.

StochReplica — an integer value that sets the number of stochastic realizations to be performed

StochSeed — a number used to initialize the pseudo-random number generator. It can be helpful in order to replicate stochastic results

quietly — see SIMULATE

RESCHECKeqList — see SIMULATE

JACOBIAN_SHOCK — see SIMULATE

JacobianDrop — see SIMULATE

avoidCompliance — see SIMULATE

... — see SIMULATE

Value

This function will add, into the output BIMETS model object, three new named elements, respectively stochastic_simulation, simulation_MM and INSTRUMENT_MM.

The stochastic_simulation element is a named list(), having endogenous variables as names. Each element will contain two time series: the mean and the standard deviation of the related

stochastic simulated endogenous time series.

The arguments passed to the function call during the latest STOCHSIMULATE run will be inserted into the '__STOCH_SIM_PARAMETERS__' element of the stochastic_simulation list; this data can be helpful in order to replicate the stochastic simulation results.

The simulation_MM element is a named list(), having the endogenous variables as names. Each element will contain an R x C matrix, given R the number of observations in the simulation TSRANGE and C=1+StochReplica. The first column of each matrix contains the related endogenous variable's unperturbed simulated values; the remaining columns will contain all the StochReplica stochastic realizations for the related endogenous variable (see example).

The INSTRUMENT_MM element is a named list(), having INSTRUMENT variables as names. Each element will contain an R x C matrix, given R the number of observations in the simulation TSRANGE and C=1+StochReplica. The first column of each matrix contains the related INSTRUMENT variable's unperturbed values; the remaining columns will contain all the StochReplica stochastic realizations for the related INSTRUMENT variable.

See Also

MDL
LOAD_MODEL
ESTIMATE
SIMULATE
RENORM
OPTIMIZE
TIMESERIES
BIMETS indexing
BIMETS configuration

Examples

```
#define the advanced Klein model
advancedKleinModelDef <- "
MODEL

COMMENT> Modified Klein Model 1 of the U.S. Economy with PDL,
COMMENT> autocorrelation on errors, restrictions and
COMMENT> conditional equation evaluations

COMMENT> Consumption with autocorrelation on errors
BEHAVIORAL> cn
TSRANGE 1923 1 1940 1
EQ> cn =  a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)
COEFF> a1 a2 a3 a4
ERROR> AUTO(2)
```

```
COMMENT> Investment with restrictions
BEHAVIORAL> i
TSRANGE 1923 1 1940 1
EQ> i = b1 + b2*p + b3*TSLAG(p,1) + b4*TSLAG(k,1)
COEFF> b1 b2 b3 b4
RESTRICT> b2 + b3 = 1

COMMENT> Demand for Labor with PDL
BEHAVIORAL> w1
TSRANGE 1923 1 1940 1
EQ> w1 = c1 + c2*(y+t-w2) + c3*TSLAG(y+t-w2,1) + c4*time
COEFF> c1 c2 c3 c4
PDL> c3 1 2

COMMENT> Gross National Product
IDENTITY> y
EQ> y = cn + i + g - t

COMMENT> Profits
IDENTITY> p
EQ> p = y - (w1+w2)

COMMENT> Capital Stock with IF switches
IDENTITY> k
EQ> k = TSLAG(k,1) + i
IF> i > 0
IDENTITY> k
EQ> k = TSLAG(k,1)
IF> i <= 0

END
"

#load the model
advancedKleinModel <- LOAD_MODEL(modelText = advancedKleinModelDef)


#define data
kleinModelData <- list(
  cn  =TIMESERIES(39.8,41.9,45,49.2,50.6,52.6,55.1,56.2,57.3,57.8,
                  55,50.9,45.6,46.5,48.7,51.3,57.7,58.7,57.5,61.6,65,69.7,
                  START=c(1920,1),FREQ=1),
  g   =TIMESERIES(4.6,6.6,6.1,5.7,6.6,6.5,6.6,7.6,7.9,8.1,9.4,10.7,
                  10.2,9.3,10,10.5,10.3,11,13,14.4,15.4,22.3,
                  START=c(1920,1),FREQ=1),
  i   =TIMESERIES(2.7,-.2,1.9,5.2,3,5.1,5.6,4.2,3,5.1,1,-3.4,-6.2,
                  -5.1,-3,-1.3,2.1,2,-1.9,1.3,3.3,4.9,
                  START=c(1920,1),FREQ=1),
  k   =TIMESERIES(182.8,182.6,184.5,189.7,192.7,197.8,203.4,207.6,
                  210.6,215.7,216.7,213.3,207.1,202,199,197.7,199.8,
                  201.8,199.9,201.2,204.5,209.4,
                  START=c(1920,1),FREQ=1),
```

```
  p   =TIMESERIES(12.7,12.4,16.9,18.4,19.4,20.1,19.6,19.8,21.1,21.7,
                  15.6,11.4,7,11.2,12.3,14,17.6,17.3,15.3,19,21.1,23.5,
                  START=c(1920,1),FREQ=1),
  w1  =TIMESERIES(28.8,25.5,29.3,34.1,33.9,35.4,37.4,37.9,39.2,41.3,
                  37.9,34.5,29,28.5,30.6,33.2,36.8,41,38.2,41.6,45,53.3,
                  START=c(1920,1),FREQ=1),
  y   =TIMESERIES(43.7,40.6,49.1,55.4,56.4,58.7,60.3,61.3,64,67,57.7,
                  50.7,41.3,45.3,48.9,53.3,61.8,65,61.2,68.4,74.1,85.3,
                  START=c(1920,1),FREQ=1),
  t   =TIMESERIES(3.4,7.7,3.9,4.7,3.8,5.5,7,6.7,4.2,4,7.7,7.5,8.3,5.4,
                  6.8,7.2,8.3,6.7,7.4,8.9,9.6,11.6,
                  START=c(1920,1),FREQ=1),
  time=TIMESERIES(NA,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,
                  1,2,3,4,5,6,7,8,9,10,
                  START=c(1920,1),FREQ=1),
  w2  =TIMESERIES(2.2,2.7,2.9,2.9,3.1,3.2,3.3,3.6,3.7,4,4.2,4.8,
                  5.3,5.6,6,6.1,7.4,6.7,7.7,7.8,8,8.5,
                  START=c(1920,1),FREQ=1)
);

#load time series into the model object
advancedKleinModel <- LOAD_MODEL_DATA(advancedKleinModel, kleinModelData)

#estimate the model
advancedKleinModel <- ESTIMATE(advancedKleinModel, quietly=TRUE)

#we want to perform a stochastic forecast of the GNP up to 1944
#we will add normal disturbances to endogenous Consumption 'cn'
#in 1942 by using its regression standard error
#we will add uniform disturbances to exogenous Government Expenditure 'g'
#in whole TSRANGE
myStochStructure <- list(
  cn=list(
        TSRANGE=c(1942,1,1942,1),
        TYPE='NORM',
        PARS=c(0,advancedKleinModel$behaviorals$cn$statistics$StandardErrorRegression)
        ),
  g=list(
        TSRANGE=TRUE,
        TYPE='UNIF',
        PARS=c(-1,1)
        )
  )

#we need to extend exogenous variables up to 1944
advancedKleinModel$modelData <- within(advancedKleinModel$modelData,{
    w2    = TSEXTEND(w2,  UPTO=c(1944,1),EXTMODE='CONSTANT')
    t     = TSEXTEND(t,   UPTO=c(1944,1),EXTMODE='LINEAR')
    g     = TSEXTEND(g,   UPTO=c(1944,1),EXTMODE='CONSTANT')
    k     = TSEXTEND(k,   UPTO=c(1944,1),EXTMODE='LINEAR')
    time  = TSEXTEND(time,UPTO=c(1944,1),EXTMODE='LINEAR')
  })
```

```
#stochastic model forecast
advancedKleinModel <- STOCHSIMULATE(advancedKleinModel
                      ,simType='FORECAST'
                      ,TSRANGE=c(1941,1,1944,1)
                      ,StochStructure=myStochStructure
                      ,StochSeed=123
                      )

#print mean and standard deviation of forecasted GNP
with(advancedKleinModel$stochastic_simulation,TABIT(y$mean, y$sd))

#      Date, Prd., y$mean         , y$sd
#
#      1941, 1   ,  125.5045      ,  4.250935
#      1942, 1   ,  173.2946      ,  9.2632
#      1943, 1   ,  185.9602      ,  11.87774
#      1944, 1   ,  141.0807      ,  11.6973

#print the unperturbed forecasted GNP along with the
#first 5 perturbed realizations
with(advancedKleinModel$simulation_MM,print(y[,1:6]))

#####################################################
#EXAMPLE WITH TYPE='MATRIX'

TSRANGE <- c(1935,1,1940,1)
StochReplica <- 100

#we will perturb simulation by using regression residuals
#get cn and i residuals in TSRANGE
cn_residuals <- TSPROJECT(advancedKleinModel$behaviorals$cn$residuals,
                          TSRANGE=TSRANGE,
                          ARRAY = TRUE)
i_residuals <- TSPROJECT(advancedKleinModel$behaviorals$i$residuals,
                         TSRANGE=TSRANGE,
                         ARRAY = TRUE)

#define stochastic matrices
cn_matrix <- c()
i_matrix <- c()

#populate matrices
for (idx in 1:StochReplica)
{
  rand <- rnorm(1,0,1)
  cn_matrix <- cbind(cn_matrix,rand*cn_residuals)
  i_matrix <- cbind(i_matrix,rand*i_residuals)
}

#define stochastic structure
myStochStructure <- list(
  cn=list(
    TSRANGE=TRUE,
```

```
    TYPE='MATRIX',
    PARS=cn_matrix
  ),
  i=list(
    TSRANGE=TRUE,
    TYPE='MATRIX',
    PARS=i_matrix
  )
)

#stochastic simulation
advancedKleinModel <- STOCHSIMULATE(advancedKleinModel
                                    ,TSRANGE=TSRANGE
                                    ,StochStructure=myStochStructure
)

#print GNP mean and sd
with(advancedKleinModel$stochastic_simulation,TABIT(y$mean, y$sd))

#########################################################
#EXAMPLE OF MODEL THAT REQUIRES THE FULL NEWTON ALGORITHM
#see profit equation

myFullNewtonDefinition<-
"MODEL

COMMENT> Consumption
BEHAVIORAL> cn
TSRANGE 1922 1 1929 1
EQ> cn =  a1 + a2*p + a3*LAG(p,1) + a4*(w1+w2+w3)
COEFF> a1 a2 a3 a4

COMMENT> Investment
BEHAVIORAL> i
TSRANGE 1922 1 1929 1
EQ> i = b1 + b2*p + b3*LAG(p,1) + b4*LAG(k,1)
COEFF> b1 b2 b3 b4

COMMENT> Demand for Labor
BEHAVIORAL> w1
TSRANGE 1922 1 1929 1
EQ> w1 = c1 + c2*(z+y+t-w2) + c3*LAG(z+y+t-w2,1) + c4*time
COEFF> c1 c2 c3 c4

COMMENT> Demand for Labor
BEHAVIORAL> w3
TSRANGE 1922 1 1929 1
EQ> w3 = c1 + c2*(z+y+t-w2) + c3*LAG(z+y+t-w2,1) + c4*time
COEFF> c1 c2 c3 c4

COMMENT> Gross National Product
IDENTITY> y
EQ> y = cn + i + g - t
```

```
IDENTITY> z
EQ> z = cn + i + g - t

COMMENT> Profits with cubic dependence on cn
IDENTITY> p
EQ> p = cn^3/1000 + z + y - (w1+w2+w3)

COMMENT> Capital Stock
IDENTITY> k
EQ> k = LAG(k,1) + i

END
"


#define model data
myModelData<-list(
   cn
   =TIMESERIES(39.8,41.9,45,49.2,50.6,52.6,55.1,56.2,57.3,57.8,55,50.9,
               45.6,46.5,48.7,51.3,57.7,58.7,57.5,61.6,65,69.7,
               START=c(1920,1),FREQ=1),
   g
   =TIMESERIES(4.6,6.6,6.1,5.7,6.6,6.5,6.6,7.6,7.9,8.1,9.4,10.7,10.2,9.3,10,
               10.5,10.3,11,13,14.4,15.4,22.3,
               START=c(1920,1),FREQ=1),
   i
   =TIMESERIES(2.7,-.2,1.9,5.2,3,5.1,5.6,4.2,3,5.1,1,-3.4,-6.2,-5.1,-3,-1.3,
               2.1,2,-1.9,1.3,3.3,4.9,
               START=c(1920,1),FREQ=1),
   k
   =TIMESERIES(182.8,182.6,184.5,189.7,192.7,197.8,203.4,207.6,210.6,215.7,
               216.7,213.3,207.1,202,199,197.7,199.8,201.8,199.9,
               201.2,204.5,209.4,
               START=c(1920,1),FREQ=1),
   p
   =TIMESERIES(12.7,12.4,16.9,18.4,19.4,20.1,19.6,19.8,21.1,21.7,15.6,11.4,
               7,11.2,12.3,14,17.6,17.3,15.3,19,21.1,23.5,
               START=c(1920,1),FREQ=1),
   w1
   =TIMESERIES(28.8,25.5,29.3,34.1,33.9,35.4,37.4,37.9,39.2,41.3,37.9,34.5,
               29,28.5,30.6,33.2,36.8,41,38.2,41.6,45,53.3,
               START=c(1920,1),FREQ=1),
   y
   =TIMESERIES(43.7,40.6,49.1,55.4,56.4,58.7,60.3,61.3,64,67,57.7,50.7,41.3,
               45.3,48.9,53.3,61.8,65,61.2,68.4,74.1,85.3,
               START=c(1920,1),FREQ=1),
   t
   =TIMESERIES(3.4,7.7,3.9,4.7,3.8,5.5,7,6.7,4.2,4,7.7,7.5,8.3,5.4,6.8,7.2,
               8.3,6.7,7.4,8.9,9.6,11.6,
               START=c(1920,1),FREQ=1),
   time
   =TIMESERIES(NA,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,
```

```
                START=c(1920,1),FREQ=1),
  w2
  =TIMESERIES(2.2,2.7,2.9,2.9,3.1,3.2,3.3,3.6,3.7,4,4.2,4.8,5.3,5.6,6,6.1,
                7.4,6.7,7.7,7.8,8,8.5,
                START=c(1920,1),FREQ=1)
)


#add data to model
myModelData$z <- myModelData$y
myModelData$w3 <- (myModelData$w1)

myFullNewtonModel <- LOAD_MODEL(modelText=myFullNewtonDefinition)
myFullNewtonModel <- LOAD_MODEL_DATA(myFullNewtonModel,myModelData)

myFullNewtonModel <- ESTIMATE(myFullNewtonModel)

#simple Newton will fail, due to
#large variance in normal disturbances
#...while full Newton will converge
myFullNewtonModel <- STOCHSIMULATE(myFullNewtonModel,
                      simAlgo='FULLNEWTON',
                      TSRANGE = c(1921, 1, 1923, 1),
                      simConvergence = 1e-5,
                      simIterLimit = 250,
                      StochReplica = 100,
                      StochSeed=123,
                      StochStructure = list(
                        cn=list(
                          TSRANGE=TRUE,
                          TYPE='NORM',
                          PARS=c(0,10)
                        )
                      )
                      ,verbose=TRUE
                      )
```

---

summary.BIMETS_MODEL *Print basic information about a BIMETS model*

Description

This function prints basic information about a BIMETS model, e.g. behaviorals and identities count, coefficients count, the presence of estimated coefficients or simulated time series.

Usage

```
## S3 method for class 'BIMETS_MODEL'
summary(object,...)
```

```
## S3 method for class 'BIMETS_MODEL'
print(x,...)
```

Arguments

object	A BIMET model.
x	A BIMET model.
...	Arguments list for the generic method.

Value

This function prints basic information about a BIMETS model, i.e.:

- the name of the model;
- the behaviorals count;
- the identities count;
- the coefficients count;
- the check for the compliance of the model data;
- the check for the coefficients definition in all the behaviorals;
- the check for the definition of a simulated time series for each related endogenous variable of the model;

See Also

MDL
LOAD_MODEL
SIMULATE
MULTMATRIX
RENORM
TIMESERIES
BIMETS indexing
BIMETS configuration

Examples

```
#define model
myModelDefinition<-
"MODEL

COMMENT> Modified Klein Model 1 of the U.S. Economy with PDL,
COMMENT> autocorrelation on errors, restrictions and conditional evaluations

COMMENT> Consumption
BEHAVIORAL> cn
TSRANGE 1925 1 1941 1
```

```
EQ> cn = a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)
COEFF> a1 a2 a3 a4
ERROR> AUTO(2)

COMMENT> Investment
BEHAVIORAL> i
TSRANGE 1923 1 1941 1
EQ> i = b1 + b2*p + b3*TSLAG(p,1) + b4*TSLAG(k,1)
COEFF> b1 b2 b3 b4
RESTRICT> b2 + b3 = 1

COMMENT> Demand for Labor
BEHAVIORAL> w1
TSRANGE 1925 1 1941 1
EQ> w1 = c1 + c2*(y+t-w2) + c3*TSLAG(y+t-w2,1) + c4*time
COEFF> c1 c2 c3 c4
PDL> c3 1 3

COMMENT> Gross National Product
IDENTITY> y
EQ> y = cn + i + g - t

COMMENT> Profits
IDENTITY> p
EQ> p = y - (w1+w2)

COMMENT> Capital Stock with switches
IDENTITY> k
EQ> k = TSLAG(k,1) + i
IF> i > 0
IDENTITY> k
EQ> k = TSLAG(k,1)
IF> i <= 0

END"

#define model data
myModelData<-list(
  cn
  =TIMESERIES(39.8,41.9,45,49.2,50.6,52.6,55.1,56.2,57.3,57.8,55,50.9,
              45.6,46.5,48.7,51.3,57.7,58.7,57.5,61.6,65,69.7,
              START=c(1920,1),FREQ=1),
  g
  =TIMESERIES(4.6,6.6,6.1,5.7,6.6,6.5,6.6,7.6,7.9,8.1,9.4,10.7,10.2,9.3,10,
              10.5,10.3,11,13,14.4,15.4,22.3,
              START=c(1920,1),FREQ=1),
  i
  =TIMESERIES(2.7,-.2,1.9,5.2,3,5.1,5.6,4.2,3,5.1,1,-3.4,-6.2,-5.1,-3,-1.3,
              2.1,2,-1.9,1.3,3.3,4.9,
              START=c(1920,1),FREQ=1),
  k
  =TIMESERIES(182.8,182.6,184.5,189.7,192.7,197.8,203.4,207.6,210.6,215.7,
              216.7,213.3,207.1,202,199,197.7,199.8,201.8,199.9,
```

```
            201.2,204.5,209.4,
            START=c(1920,1),FREQ=1),
  p
  =TIMESERIES(12.7,12.4,16.9,18.4,19.4,20.1,19.6,19.8,21.1,21.7,15.6,11.4,
            7,11.2,12.3,14,17.6,17.3,15.3,19,21.1,23.5,
            START=c(1920,1),FREQ=1),
  w1
  =TIMESERIES(28.8,25.5,29.3,34.1,33.9,35.4,37.4,37.9,39.2,41.3,37.9,34.5,
            29,28.5,30.6,33.2,36.8,41,38.2,41.6,45,53.3,
            START=c(1920,1),FREQ=1),
  y
  =TIMESERIES(43.7,40.6,49.1,55.4,56.4,58.7,60.3,61.3,64,67,57.7,50.7,41.3,
            45.3,48.9,53.3,61.8,65,61.2,68.4,74.1,85.3,
            START=c(1920,1),FREQ=1),
  t
  =TIMESERIES(3.4,7.7,3.9,4.7,3.8,5.5,7,6.7,4.2,4,7.7,7.5,8.3,5.4,6.8,7.2,
            8.3,6.7,7.4,8.9,9.6,11.6,
            START=c(1920,1),FREQ=1),
  time
  =TIMESERIES(NA,-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,
            START=c(1920,1),FREQ=1),
  w2
  =TIMESERIES(2.2,2.7,2.9,2.9,3.1,3.2,3.3,3.6,3.7,4,4.2,4.8,5.3,5.6,6,6.1,
            7.4,6.7,7.7,7.8,8,8.5,
            START=c(1920,1),FREQ=1)
)


#load model
myModel<-LOAD_MODEL(modelText=myModelDefinition)

#model summary
summary(myModel)

#BIMETS MODEL
#----------------------------------
#name:             myModelDefinition
#behaviorals:      3
#identities:       3
#coefficients:     12
#model data:       not OK
#.CHECK_MODEL_DATA(): model has no data. Please use LOAD_MODEL_DATA().
#fully estimated:  FALSE
#simulated:        FALSE

#load data into the model
myModel<-LOAD_MODEL_DATA(myModel,myModelData,showWarnings = TRUE)

#estimate the model
myModel<-ESTIMATE(myModel)

#model summary
print(myModel)
```

```
#BIMETS MODEL
#----------------------------------
#name:             myModelDefinition
#behaviorals:      3
#identities:       3
#coefficients:     12
#model data:       OK
#fully estimated:  TRUE
#simulated:        FALSE
```

TABIT *Print Time Series Data*

Description

This function prints, in a human-readable format, the content of a list of time series. Time series can be subsetted with the projection argument TSRANGE.

Usage

```
TABIT(...,
      TSRANGE=NULL,
      digits=getOption('digits'),
      avoidCompliance=FALSE)
```

Arguments

...	Input list of time series that must satisfy the compliance control check defined in is.bimets.
TSRANGE	The optional date range of data presentation. TSRANGE must be specified as a numerical array composed by starting year, staring period, ending year and ending period of projection, i.e. TSRANGE=c(START_YEAR,START_PERIOD,END_YEAR,END_PERIOD).
digits	Controls the number of digits to print out. Valid values are 1 to 22 with a default of 7.
avoidCompliance	If TRUE, compliance control check of input time series will be skipped. See is.bimets

Value

This function prints out time series data and returns a NULL value.

See Also

TSPROJECT
MOVAVG
TSDELTA
TSLAG
TSPROJECT
TSEXTEND
TSLEAD

Examples

```
#create monthly series
ts1<-TSERIES(INTS(1,15),START=c(2000,1),FREQ=12)
ts2<-TSERIES(INTS(1,15),START=c(2001,1),FREQ=12)
ts3<-TSERIES(rnorm(15),START=c(2002,1),FREQ=12)
ts4<-TSERIES(rep(NA,15),START=c(2001,4),FREQ=12)

TABIT(ts1,ts2,ts3,ts4)
# print...
#
#      Date, Prd., ts1            , ts2            , ts3            , ts4
#
#  Jan 2000, 1   ,  1             ,                ,                ,
#  Feb 2000, 2   ,  2             ,                ,                ,
#  Mar 2000, 3   ,  3             ,                ,                ,
#  ...
#  Dec 2000, 12  ,  12            ,                ,                ,
#  Jan 2001, 1   ,  13            ,  1             ,                ,
#  Feb 2001, 2   ,  14            ,  2             ,                ,
#  ...
#  Dec 2001, 12  ,                ,  12            ,                ,  NA
#  Jan 2002, 1   ,                ,  13            , -1.419782      ,  NA
#  Feb 2002, 2   ,                ,  14            , -1.070188      ,  NA
#  Mar 2002, 3   ,                ,  15            ,  0.889571      ,  NA
#  Apr 2002, 4   ,                ,                ,  0.9583392     ,  NA
#  ...
#  Feb 2003, 2   ,                ,                , -0.3444237     ,
#  Mar 2003, 3   ,                ,                , -0.3073225     ,


#create quarterly series, set TSRANGE then print with 3 digits

ts1<-TSERIES(INTS(1,15),START=c(2000,1),FREQ=4)
ts2<-TSERIES(INTS(1,15),START=c(2001,1),FREQ=4)
ts3<-TSERIES(rnorm(15),START=c(2002,1),FREQ=4)
ts4<-TSERIES(rep(NA,15),START=c(2001,4),FREQ=4)

TABIT(ts1,ts2,ts3,ts4,TSRANGE=c(1991,3,2003,2),digits=3)
```

```
#print...
#
#      Date, Prd., ts1        , ts2        , ts3        , ts4
#
#   2000 Q1, 1   ,  1         ,            ,            ,
#   2000 Q2, 2   ,  2         ,            ,            ,
#   2000 Q3, 3   ,  3         ,            ,            ,
#   2000 Q4, 4   ,  4         ,            ,            ,
#   2001 Q1, 1   ,  5         ,  1         ,            ,
#   2001 Q2, 2   ,  6         ,  2         ,            ,
#   2001 Q3, 3   ,  7         ,  3         ,            ,
#   2001 Q4, 4   ,  8         ,  4         ,            ,  NA
#   2002 Q1, 1   ,  9         ,  5         ,  0.729     ,  NA
#   2002 Q2, 2   ,  10        ,  6         ,  0.923     ,  NA
#   2002 Q3, 3   ,  11        ,  7         , -0.81      ,  NA
#   2002 Q4, 4   ,  12        ,  8         , -0.0748    ,  NA
#   2003 Q1, 1   ,  13        ,  9         ,  0.248     ,  NA
#   2003 Q2, 2   ,  14        ,  10        , -0.347     ,  NA


#create daily series and set TSRANGE

ts1<-TSERIES(INTS(1,25),START=c(2000,1),FREQ=366)
ts2<-TSERIES(INTS(1,25),START=c(2000,10),FREQ=366)
ts3<-TSERIES(rnorm(25),START=c(2000,20),FREQ=366)
ts4<-TSERIES(rep(NA,25),START=c(2000,30),FREQ=366)

TABIT(ts1,ts2,ts3,ts4,TSRANGE=c(2000,5,2000,35))

#...print data
```

TSDELTA *Time Series Lag Differences (Delta)*

Description

This function returns the O-order, L-lag differences of the input time series.

Usage

```
TSDELTA(x = NULL, L = 1, O = 1, avoidCompliance = FALSE, ...)
```

Arguments

x	Input time series that must satisfy the compliance control check defined in is.bimets.
L	Lag.
O	Order of the difference.

avoidCompliance
: If TRUE, compliance control check of input time series will be skipped. See is.bimets

...
: Backward compatibility.

Value

This function returns a BIMETS time series.

See Also

TSDELTAP
TSDELTALOG
TSLAG
MOVAVG
INTS
CUMSUM

Examples

```
#random TS
n<-10
xArr<-rnorm(n)
ts1<-TSERIES(xArr,START=c(2000,1),FREQ='A')
TABIT(ts1,TSDELTA(ts1,1,1),TSDELTA(ts1,1,2),TSDELTA(ts1,1,3))
```

TSDELTALOG *Time Series Lag Logarithmic Differences*

Description

This function returns the L-lag logarithmic differences of the input time series.

Usage

```
TSDELTALOG(x = NULL, L = 1,  avoidCompliance = FALSE, ...)
```

Arguments

x
: Input time series that must satisfy the compliance control check defined in is.bimets.

L
: Lag.

avoidCompliance
: If TRUE, compliance control check of input time series will be skipped. See is.bimets

...
: Backward compatibility.

Value

This function returns a BIMETS time series.

See Also

TSDELTAP
TSLAG
MOVAVG
INTS
CUMSUM

Examples

```
#sequence TS
n<-10
ts1<-TSERIES(1:n,START=c(2000,1),FREQ='A')
TABIT(ts1,TSDELTALOG(ts1,1))
```

TSDELTAP *Time Series Percentage Lag Differences (Delta Percentage)*

Description

This function returns the 0-order, L-lag percentage differences of the input time series. If the input time series frequency is a multiple of the L lag argument, then it is possible to set the argument ANNUALIZE=TRUE in order to have the percent changes returned at annual rates, i.e. raised to the power of frequency/L.

Usage

```
TSDELTAP(x = NULL, L = 1, ANNUALIZE = FALSE, avoidCompliance = FALSE, ...)
```

Arguments

x	Input time series that must satisfy the compliance control check defined in is.bimets.
L	Lag.
ANNUALIZE	If TRUE the percent changes are returned as annual rates, i.e. raised to the power of frequency/L
avoidCompliance	If TRUE, compliance control check of input time series will be skipped. See is.bimets
...	Backward compatibility.

Value

This function returns a BIMETS time series.

See Also

TSDELTA
TSLAG
MOVAVG
INDEXNUM

Examples

```
#TS Q
n<-10;
ts1<-TSERIES(n:0,START=c(2000,1),FREQ='Q')
TABIT(ts1,TSDELTAP(ts1,1))

#TS 366
ts1<-TSERIES(seq(1,length=10,by=-0.001),START=c(2000,1),FREQ=366)
TABIT(ts1,TSDELTAP(ts1,1,ANNUALIZE=TRUE))
```

TSERIES *Create a Time Series*

Description

This function returns a time series that is compliant with BIMETS compliance control defined in is.bimets. Users can provide observation values, frequency, and the starting period. Moreover, users can provide metadata information that will be stored into the time series object as attributes.

TIMESERIES is an alias for TSERIES.

Usage

```
TIMESERIES(..., START = c(2000,1), FREQ = 1,
           SOURCE = NULL, TITLE = NULL, UNITS = NULL, SCALEFAC = 0,
           class=NULL, avoidCompliance = FALSE)

TSERIES(..., START = c(2000,1), FREQ = 1,
        SOURCE = NULL, TITLE = NULL, UNITS = NULL, SCALEFAC = 0,
        class=NULL, avoidCompliance = FALSE)
```

Arguments

...	List of values to be inserted into the output time series. This function accepts arguments of class ts() and xts(), that must be BIMETS compliant as defined in is.bimets. It also accepts numerical arrays. Please note that for daily and weekly time series, as in the default R time series class ts(), this function will insert provided input values always filling up to the 366th period in the daily case and up to the 53rd period in the weekly case. (see example)
START	This argument defines the start of the time series. Starting period can be specified as c(YEAR,PERIOD), or as Date(), or as yearmon() if the frequency FREQ=12, or as yearqtr() if the frequency FREQ=4. Please note that the time series must lie in the year range 1800-2199: conversion between date and year-period has been optimized and hard-coded for all frequencies within this time range.
FREQ	The frequency of the time series. Frequency can be FREQ=1, 2, 3, 4, 12, 24, 36, 53, or 366. Frequency can also be defined by using the char 'A' or 'Y' for annual/yearly, 'S' for semiannual, 'Q' for quarterly, 'M' for monthly, 'W' for weekly, and 'D' for daily time series.
SOURCE	Set the metadata string that represents the source of the data. Metadata will be lost if the current time series is transformed by any function that changes its values.
TITLE	Set the metadata string that represents the description of the data. Metadata will be lost if the current time series is transformed by any function that changes its values.
UNITS	Set the metadata string that represents the unit of measure of the data. Metadata will be lost if the current time series is transformed by any function that changes its values.
SCALEFAC	Set the numerical value that represents the scale factor of the data. Users may eventually want to use this value in code. Metadata will be lost if the current time series is transformed by any function that changes its values.
class	If class='XTS' this function will return a time series based on the xts() class. If class='TS' this function will return a time series based on the ts() class. If class=NULL (default) the output base class will be the one defined in the global BIMETS option 'BIMETS_CONF_CCT' (see BIMETS configuration). Please note that package functions only accept time series belonging to the same class as the one defined in the global option 'BIMETS_CONF_CCT'. Users can change any global option directly in the code. Please note that BIMETS package performs better with class='TS' or more generally with BIMETS_CONF_CCT='TS'
avoidCompliance	If TRUE, compliance control check of input time series will be skipped. See is.bimets

Value

This function returns a BIMETS time series that is compliant with the BIMETS compliance control defined in is.bimets.

See Also

is.bimets
as.bimets
BIMETS indexing
BIMETS configuration
fromBIMETStoTS
fromBIMETStoXTS
NOELS
TSDATES
INTS
TABIT

Examples

```
#day and month names can change depending on locale
Sys.setlocale('LC_ALL','C')
Sys.setlocale('LC_TIME','C')

#create a simple R ts
n<-10
ts1<-ts((1:n),start=c(2000,1),frequency=1)

#create a bimets ts annual
#having the following values: 5, (all data in ts1), NA, 8
#starting from Jan 2020, and having custom metadata
out_tseries<-TIMESERIES(5,ts1,NA,8,START=c(2020,1),FREQ=1,
SOURCE='mySource',TITLE='myTitle',UNITS='myUnits',SCALEFAC=2)

#print out
TABIT(out_tseries)

#use Date() as start date
TABIT(TIMESERIES(1:10,START=as.Date('2000-01-01'),FREQ='A'))
TABIT(TIMESERIES(1:10,START=as.Date('2000-01-01'),FREQ='Y'))
TABIT(TIMESERIES(1:10,START=as.Date('2000-07-08'),FREQ='D'))
TABIT(TIMESERIES(1:10,START=as.Date('2018-01-01'),FREQ='W'))

#use yearmon()/yearqtr() as start date
TABIT(TIMESERIES(1:10,START=as.yearmon('Mar 2001'),FREQ='M'))
TABIT(TIMESERIES(1:10,START=as.yearqtr('2000 Q3'),FREQ='Q'))

#create ts monthly with metadata
out_tseries<-TIMESERIES(5,ts1,NA,8,START=c(2020,1),FREQ='M',
SOURCE='mySource',TITLE='myTitle',UNITS='myUnits',SCALEFAC=2)

#print out
TABIT(out_tseries)

#create daily
```

```
out_tseries<-TSERIES(5,ts1,NA,8,START=c(2000,1),FREQ=366,
SOURCE='mySource',TITLE='myTitle',UNITS='myUnits',SCALEFAC=2)

#print out
TABIT(out_tseries)

#insert values skipping 366 in non-bissextile
myLength<-400
myValues<-1:myLength
myDates<-as.Date('2001-01-01')+0:(myLength-1)
ts<-as.bimets(xts(myValues,order.by = myDates))
TABIT(ts) #366 observation will be a duplicated of 365, see as.bimets() help
```

TSEXTEND *Extend Time Series*

Description

This function extends the time series definition range by using the directives specified in the EXTMODE and FACTOR arguments.

Usage

```
TSEXTEND(x = NULL, BACKTO = NULL, UPTO = NULL, EXTMODE = "GROWTH",
       FACTOR = NA, avoidCompliance = FALSE, ...)
```

Arguments

x	Input time series that must satisfy the compliance control check defined in is.bimets.
BACKTO	Define the new start of the time series, which must be provided as c(YEAR,PERIOD). It is possible to convert a Date(), or a yearmon(), or a yearqtr() to the related c(YEAR,PERIOD) by using the functions date2yp, ym2yp, and yq2yp.
UPTO	Define the new end of the time series, which must be provided as c(YEAR,PERIOD). It is possible to convert a Date(), or a yearmon(), or a yearqtr() to the related c(YEAR,PERIOD) by using the functions date2yp, ym2yp, and yq2yp.
EXTMODE	It must be one of the following: **MISSING**: extend the time series using missings values NA. **ZERO**: extend the time series by using 0 (zero) values. **CONSTANT**: extend the time series by using the closest non-missing observation. **MEAN4**: extend the time series by using the mean of the closest four non-missing observations. **LINEAR**: extend the time series by using the same increment of the closest couple of observations. **QUADRATIC**: extend the time series by using the same quadratic increment of the closest eight observations.

GROWTH: extends the time series by using the closest growth rate.
GROWTH4: extend the time series by using the factor
`r=(mean(x[-1:-4])/mean(x[-5:-8]))**(1/4)`
MYCONST: extend the time series by using the value defined in `FACTOR`
MYRATE: extend the time series by using the increment defined in `FACTOR`

`FACTOR` User-defined value used by some options of the `EXTMODE` argument.

`avoidCompliance`
If `TRUE`, compliance control check of input time series will be skipped. See `is.bimets`

`...` Backward compatibility.

Value

This function returns a BIMETS time series built by extending the input time series.

See Also

`TSLAG`
`TSJOIN`
`TSMERGE`
`TSPROJECT`
`CUMSUM`
`INDEXNUM`
`TSTRIM`

Examples

```
n<-10;
ts1<-TIMESERIES(1:n,START=c(2000,1),FREQ='A')
ts2<-TSEXTEND(ts1,BACKTO=c(1990,1),UPTO=c(2020,1),EXTMODE='GROWTH4')
TABIT(ts1,ts2)

xArr<-c(0.5,5.6,4.8,3.8,7.3,9.9,7.8,3.7,8.2,10)
ts1<-TIMESERIES(xArr,START=c(2000,1),FREQ='A')
ts2<-TSEXTEND(ts1,BACKTO=c(1990,1),UPTO=c(2020,1),EXTMODE='QUADRATIC')
TABIT(ts1,ts2)

xArr<-(1:n)
dateArr<-seq(as.Date('2000/12/31'),by='year',length=n)
dataF<-data.frame(dateArr,xArr)
ts1<-TIMESERIES(xArr,START=c(2000,1),FREQ='A')
ts2<-TSEXTEND(ts1,BACKTO=c(1990,1),UPTO=c(2020,1),EXTMODE='MYRATE',FACTOR=2.5)
TABIT(ts1,ts2)
```

TSINFO *Get Time Series Info*

Description

This function returns detailed information about the input time series list. The requested information is defined in the argument MODE.

Usage

```
TSINFO(..., MODE = NULL, avoidCompliance=FALSE)
```

Arguments

...	Input time series list. Each time series must satisfy the compliance control check defined in is.bimets.
MODE	Select the information to be retrieved from the list of time series. MODE can be set to: **STARTY**: the output will be a numerical array built with the starting year of each time series in the input list. **ENDY**: the output will be a numerical array built with the ending year of each time series in the input list. **STARTP**: the output will be a numerical array built with the starting period of each time series in the input list. **ENDP**: the output will be a numerical array built with the ending period of each time series in the input list. **START**: the output will be a numerical array built with the value x=START_YEAR+START_PERIOD/FREQ calculated on each time series in the input list, where FREQ is the time series frequency. **END**: the output will be a numerical array built with the value x=END_YEAR+END_PERIOD/FREQ calculated on each time series in the input list. **START2**: the output will be a numerical matrix. For each time series in the input list the output matrix will have a row=c(START_YEAR,START_PERIOD) with related values. **END2**: the output will be a numerical matrix. For each time series in the input list the output matrix will have a row=c(END_YEAR,END_PERIOD) with related values. **FREQ**: the output will be a numerical array built with the frequency of each time series in the input list. **FACTOR**: the output will be a numerical array built with the SCALEFAC metadata value of each time series in the input list. **UNITS**: the output will be a string array built with the UNITS metadata string of each time series in the input list. **TITLE**: the output will be a string array built with the TITLE metadata string of each time series in the input list. **SOURCE**: the output will be a string array built with the SOURCE metadata

string of each time series in the input list.

avoidCompliance

If TRUE, compliance control check of input time series will be skipped. See is.bimets

Value

This function returns an array built of the requested information about the input time series list. In the case of MODE=START2 or MODE=END2 the output will be of class matrix().

See Also

NOELS
is.bimets
BIMETS indexing
fromBIMETStoTS
fromBIMETStoXTS
GETYEARPERIOD
INTS
TSLOOK
TABIT

Examples

```
#create ts
ts1<-TIMESERIES(INTS(1,10),START=c(2004,2),FREQ=2,
UNITS='myUnits',TITLE='myTitle',SOURCE='mySource')
ts2<-TIMESERIES(INTS(1,20),START=c(2006,3),FREQ=4,SCALEFAC=1)
ts3<-TIMESERIES(INTS(1,30),START=c(2008,7),FREQ=12)

print(TSINFO(ts1,ts2,ts3,MODE='STARTY')) #print ... c(2004,2006,2008)
print(TSINFO(ts1,ts2,ts3,MODE='ENDP')) #print ... c(1,2,12)
print(TSINFO(ts1,ts2,ts3,MODE='FREQ')) #print ... c(2,4,12)

print(TSINFO(ts1,ts2,ts3,MODE='START2'))
#print ...
#[,1] [,2]
#[1,] 2004    2
#[2,] 2006    3
#[3,] 2008    7

print(TSINFO(ts1,ts2,ts3,MODE='END')) #print ... c(2009.5, 2011.5, 2011.0)
print(TSINFO(ts1,ts2,ts3,MODE='FACTOR')) #print ...  c(0,1,0)
print(TSINFO(ts1,ts2,ts3,MODE='UNITS')) #print ...  c('myUnits','','')
```

TSJOIN *Join Time Series*

Description

This function returns the join of the two input time series. If the first time series overlaps the second time series, output data is taken from the first time series up to the second time series's starting date, the remainder of the data being taken from the second time series.
A different joining period can be specified by using the JPRD argument.
The two time series must have the same frequency.

Usage

```
TSJOIN(x = NULL, y = NULL, JPRD = NULL, ALLOWGAP = FALSE,
       WARN = FALSE, avoidCompliance = FALSE, ...)
```

Arguments

x	First input time series that must satisfy the compliance control check defined in is.bimets.
y	Second input time series that must satisfy the compliance control check defined in is.bimets.
JPRD	This argument defines a joining period other than the starting period of the second time series. It must be defined as JPRD=c(YEAR,PERIOD) and must lie in the time range of the second time series. Users can convert a Date(), or a yearmon() or a yearqtr() to a c(YEAR,PERIOD) by using date2yp, ym2yp, yq2yp.
ALLOWGAP	if TRUE, the possible gap between the two time series is filled with missing values NA, otherwise, if the two time ranges do not overlap, an error will be thrown.
WARN	Print a warning message if the two time series do not overlap or if the first time series starts after the JPRD. The warning is shown only if ALLOWGAP=TRUE.
avoidCompliance	If TRUE, compliance control check of input time series will be skipped. See is.bimets
...	Backward compatibility.

Value

This function returns a BIMETS time series that is built by joining the two input time series.

See Also

TSLAG
TSEXTEND
TSMERGE
TSPROJECT

Examples

```
#day and month names can change depending on locale
Sys.setlocale('LC_ALL','C')
Sys.setlocale('LC_TIME','C')

#TS
ts1<-TSERIES((1:10),START=c(1985,1),FREQ=1)
ts2<-TSERIES((1:10),START=c(2000,1),FREQ=1)
TABIT(ts1,ts2,TSJOIN(ts1,ts2,ALLOWGAP=TRUE))

#XTS
setBIMETSconf('BIMETS_CONF_CCT','XTS')
n<-10
xArr<-(0:n)
dateArr<-as.yearqtr('1997 Q1')+ 0:n/4
dataF<-data.frame(dateArr,xArr)
ts1<-xts(dataF[,2],order.by=dataF[,1])
dateArr<-as.yearqtr('2000 Q1')+ 0:n/4
dataF<-data.frame(dateArr,xArr)
ts2<-xts(dataF[,2],order.by=dataF[,1])
TABIT(ts1,ts2,TSJOIN(ts1,ts2,ALLOWGAP=TRUE,JPRD=yq2yp(as.yearqtr("2001 Q3"))))

#restore default
setBIMETSconf('BIMETS_CONF_CCT','TS')
```

TSLAG *Lag Time Series*

Description

This function lags the input time series by the specified number of time periods.

Usage

```
TSLAG(x = NULL, L = 1, avoidCompliance = FALSE, ...)
```

Arguments

x	Input time series that must satisfy the compliance control check defined in is.bimets.
L	Lag. Must be an integer, positive or negative.
avoidCompliance	If TRUE, compliance control check of input time series will be skipped. See is.bimets
...	Backward compatibility.

Value

This function returns a BIMETS time series built by lagging the input time series.

See Also

TSJOIN
TSEXTEND
TSMERGE
MOVAVG
GETYEARPERIOD
TSLEAD

Examples

```
#DEFINE TS
n<-10
ts1<-TSERIES(n:1,START=c(2000,1),FREQ=1)
ts1[5]<-NA

#print TSLAG
TABIT(ts1,TSLAG(ts1,5))

n<-10
ts1<-TSERIES(n:1,START=c(2000,1),FREQ='D')
ts1[5]<-NA

#print TSLAG
TABIT(ts1,TSLAG(ts1,5))
```

TSLEAD *Lead Time Series*

Description

This function leads the input time series by the specified number of time periods.

Usage

```
TSLEAD(x = NULL, L = 1, avoidCompliance = FALSE, ...)
```

Arguments

x	Input time series that must satisfy the compliance control check defined in is.bimets.
L	Lead. Must be an integer, positive or negative.
avoidCompliance	If TRUE, compliance control check of input time series will be skipped. See is.bimets
...	Backward compatibility.

Value

This function returns a BIMETS time series built by leading the input time series.

See Also

TSJOIN
TSEXTEND
TSMERGE
MOVAVG
GETYEARPERIOD
TSLAG

Examples

```
#DEFINE TS
n<-10

ts1<-TSERIES(n:1,START=c(2000,1),FREQ=1)
ts1[5]<-NA

#print TSLEAD
TABIT(ts1,TSLEAD(ts1,5))
```

TSLOOK *Lookup a Time Series*

Description

This function returns the time range and the frequency of an input time series.

Usage

```
TSLOOK(x=NULL, avoidCompliance=FALSE, ...)
```

Arguments

x	Input time series that must satisfy the compliance control check defined in is.bimets.
avoidCompliance	If TRUE, compliance control check of input time series will be skipped. See is.bimets.
...	Backward compatibility.

Value

This function returns a list of numeric arrays built with the following elements:
STARTY will contain the value of the starting year
STARTP will contain the value of the starting period
ENDY will contain the value of the ending year
ENDP will contain the value of the ending period
FREQ will contain the value of the time series frequency

See Also

NOELS
is.bimets
BIMETS indexing
fromBIMETStoXTS
fromBIMETStoTS
GETYEARPERIOD
INTS
TSINFO

Examples

```
#create series
ts1<-TSERIES(INTS(1,10),START=c(2000,1),FREQ=12)

ts1Look<-TSLOOK(ts1)

print(ts1Look$STARTY) #print...2000
print(ts1Look$STARTP) #print...1
print(ts1Look$ENDY) #print...2000
print(ts1Look$ENDP) #print...10
print(ts1Look$FREQ) #print...12
```

TSMERGE *Merge Time Series*

Description

This function merges and concatenates two or more time series of the same frequency. The output time series will be defined over the union of dates for which the input time series are defined, from the earliest starting date to the latest ending date.

For each period, the output value will be set equal to the first non-missing value found in the input time series list by using the order of the arguments. If all the input time series are missing at a period, then the output time series will be set to the missing value NA in the same period. Note

that if the input time series' date spans do not intersect, TSMERGE(X1,X2,...,XN) returns a simple concatenation of X1, X2,..., XN.

By defining the argument fun, the value of the output time series can also be computed as a function of the values of the input time series in the same period (see example).

Usage

```
TSMERGE(..., fun = NULL, MV = FALSE, avoidCompliance = FALSE)
```

Arguments

...	Input list of time series that must satisfy the compliance control check defined in is.bimets.
fun	By defining the argument fun, the value of the output time series in a period can be computed as a function of the input time series values in the same period. fun can assume the following string values: **AVE**: the value of the output time series in a period will be set equal to the average of all input time series values in the same period. **SUM**: the value of the output time series in a period will be set equal to the sum of all input time series values in the same period. **MAX**: the value of the output time series in a period will be set equal to the maximum of all input time series values in the same period. **MIN**: the value of the output time series in a period will be set equal to the minimum of all input time series values in the same period.
MV	If FALSE, the function defined in the argument fun will skip any missing values found in the input time series.
avoidCompliance	If TRUE, compliance control check of input time series will be skipped. See is.bimets

Value

This function returns a BIMETS time series built by merging two or more input time series.

See Also

TSJOIN
TSEXTEND
TSLAG
TSPROJECT
TSLEAD
TSTRIM

Examples

```
#DEFINE TS
n<-10
ts1<-TSERIES(n:1,START=c(1995,1),FREQ=1)
ts2<-TSERIES(n:1,START=c(2000,1),FREQ=1)

ts2[5]<-NA
ts1[10]<-NA

#print TSMERGE
TABIT(ts1,ts2,TSMERGE(ts1,ts2,fun='SUM',MV=TRUE))

#TS D
n<-20
ts1<-TSERIES(n:1,START=c(1999,360),FREQ='D')
ts2<-TSERIES(n:1,START=c(2000,1),FREQ='D')

ts2[5]<-NA
ts1[10]<-NA

#print TSMERGE
TABIT(ts1,ts2,TSMERGE(ts1,ts2,fun='SUM',MV=TRUE))
```

TSPROJECT *Project a Time series*

Description

This function projects the input time series into a time interval. The output class can be either a time series (default) or a one-dimensional array if the argument ARRAY=TRUE.

Usage

```
TSPROJECT(x=NULL, TSRANGE=NULL, ARRAY=FALSE,
          EXTEND=FALSE, avoidCompliance=FALSE,...)
```

Arguments

x	Input time series that must satisfy the compliance control check defined in is.bimets.
TSRANGE	Date range of data projection. TSRANGE must be specified as a numerical array composed by starting year, starting period, ending year and ending period of projection, i.e. TSRANGE=c(START_YEAR,START_PERIOD,END_YEAR,END_PERIOD).
ARRAY	If TRUE this function will return a numerical array built with observation values that lie in the specified time range. If FALSE (default) the output will be a time series.

EXTEND — If TRUE and in the case that the input time series does not overlap with the provided TSRANGE, the output time series will be extended over the TSRANGE by inserting missing values NA into the new observations.

avoidCompliance — If TRUE, compliance control check of input time series will be skipped. See is.bimets

... — Backward compatibility.

Value

This function returns a BIMETS time series, or a numerical array if the argument ARRAY=TRUE, built by projecting the input time series into the provided TSRANGE.

See Also

TSJOIN
TSEXTEND
TSMERGE
MOVAVG
GETYEARPERIOD
CUMSUM
TSTRIM

Examples

```
#create yearly ts
ts1<-TSERIES((1:10),START=c(2000,1),FREQ=1)

print(TSPROJECT(ts1,TSRANGE=c(2002,1,2005,1))) #print projected ts from 2002 to 2005

print(TSPROJECT(ts1,TSRANGE=c(2001,1,2004,1),ARRAY=TRUE)) #print c(2,3,4,5)

print(TSPROJECT(ts1,TSRANGE=c(1998,1,2002,1),EXTEND=TRUE,ARRAY=TRUE))
#print c(NA,NA,1,2,3)
```

TSTRIM *Trim a Time Series*

Description

This function removes trailing or leading missing values NA from the input array or the input time series. Users can provide the value to be removed other than NA missing value by using the argument VALUE.

Usage

```
TSTRIM(x=NULL, VALUE=NA, TRAIL=TRUE, LEAD=TRUE, avoidCompliance=FALSE, ...)
```

Arguments

x	Input numerical array or time series that must satisfy the compliance control check defined in is.bimets.
VALUE	Target value to be removed. Default to missing value NA.
TRAIL	If TRUE this function will remove trailing target values from the input time series.
LEAD	If TRUE this function will remove leading target values from the input time series.
avoidCompliance	If TRUE, compliance control check of input time series will be skipped. See is.bimets
...	Backward compatibility.

Value

This function returns an object of the same class of the input, i.e. an array or a BIMETS time series, built by removing leading and trailing user-defined values.

See Also

TSLAG
TSJOIN
TSMERGE
TSPROJECT
CUMSUM
INDEXNUM

Examples

```
#TS A
n<-10
ts1<-TSERIES(c(NA,1:n,NA),START=c(2000,1),FREQ='A')
TABIT(ts1,TSTRIM(ts1))

ts1<-TSERIES(c(NA,1:n,NA),START=c(2000,1),FREQ='A')
TABIT(ts1,TSTRIM(ts1,TRAIL=FALSE))

ts1<-TSERIES(c(NA,1:n,NA),START=c(2000,1),FREQ='A')
TABIT(ts1,TSTRIM(ts1,LEAD=FALSE))

ts1<-TSERIES(c(0,0,NA,1:n,NA,0),START=c(2000,1),FREQ='A')
TABIT(ts1,TSTRIM(ts1,0))
```

VERIFY_MAGNITUDE *Time Series Magnitude Test*

Description

Given a time series list, this function returns list indices that refer to time serie having a magnitude greater than the user provided value. Magnitude `M` is defined as `M = sqrt(sum(X_i^2))`, given `X_i` the time series values. Missing values will be discarded with a warning message.

This function can be useful in comparing list of time series, e.g. differences between historical and simulated endogenous variables of an econometric model.

Usage

```
VERIFY_MAGNITUDE(x=list(),
                 magnitude=10e-7,
                 verbose=TRUE,
                 ...)
```

Arguments

`x`	Input list, having elements as time series of class `ts` or `xts`.
`magnitude`	A positive number that is the maximum magnitude allowed.
`verbose`	If `TRUE`, output messages will be printed out.
`...`	Backward compatibility.

Value

This function returns the list indices related to time series that have a magnitude grater than the `magnitude` user argument.

See Also

`TSJOIN`
`TSEXTEND`
`TSMERGE`
`MOVAVG`
`GETYEARPERIOD`
`CUMSUM`

Examples

```
#create ts
ts1=TSERIES((1:10)*0.1, START=c(2000,1), FREQ=1)
ts2=TSERIES((1:10)*0.01, START=c(2001,1), FREQ=4)
ts3=TSERIES(c((1:10)*0.001,NA), START=c(2002,1), FREQ=12)
```

```
myList=list(t1=ts1,t2=ts2,t3=ts3)

print(VERIFY_MAGNITUDE(myList,magnitude=0.1))
```

---

ym2yp *yearmon to Year-Period Conversion*

Description

This function transforms an input variable of class `yearmon()` into an equivalent two-dimensional numerical array of type `c(YEAR,PERIOD)`.

Usage

```
ym2yp(x = NULL)
```

Arguments

`x`	Input of class `yearmon()`

Value

This function returns a two-dimensional numerical array of type `c(YEAR,PERIOD)`.

See Also

`date2yp`
`yq2yp`
`GETDATE`

Examples

```
#day and month names can change depending on locale
Sys.setlocale('LC_ALL','C')
Sys.setlocale('LC_TIME','C')

print(ym2yp(as.yearmon("Dec 2013"))); #print c(2013,12)
print(ym2yp(c(as.yearmon('Jan 2000'),as.yearmon('Dec 1987'),
      as.yearmon('Jan 2003'),as.yearmon('Mar 2012'))))
```

yq2yp *yearqtr to Year-Period Conversion*

Description

This function transforms an input variable of class `yearqtr()` into an equivalent two-dimensional numerical array of type `c(YEAR,PERIOD)`.

Usage

```
yq2yp(x = NULL)
```

Arguments

x	Input of class `yearmon()`

Value

This function returns a two-dimensional numerical array of type `c(YEAR,PERIOD)`.

See Also

date2yp
ym2yp
GETDATE

Examples

```
#day and month names can change depending on locale
Sys.setlocale('LC_ALL','C')
Sys.setlocale('LC_TIME','C')

print(yq2yp(as.yearqtr('2001 Q3'))); #print c(2001,3)
print(yq2yp(c(as.yearqtr('2000 Q2'),as.yearqtr('1987 Q4'),as.yearqtr('2003 Q1'))))
```

Index