bimets:
Time Series and Econometric Modeling in R

Andrea Luciani
Bank of Italy*

Abstract

bimets is an R package developed to ease time series analysis and build up a framework that facilitates the definition, estimation, and simulation of simultaneous equation models.

This package supports daily, weekly, monthly, quarterly, semiannual, and yearly time series. Time series with frequency of 24 and 36 periods per year are also supported. Users can access and modify time series data by date, year-period, and observation index. Advanced time series manipulation and (dis)aggregation capabilities are provided, e.g. time series extension, merging, projection, lag, cumulative and moving product and sum, etc.

Econometric modeling capabilities comprehend advanced model definition (e.g. conditional evaluation of equations, per-equation estimation method and time range), estimation of equations with instrumental variables, coefficient restrictions and error autocorrelation, structural stability analysis, deterministic and stochastic simulation and forecasting of simultaneous equations with exogenizations and add-factors, interim and impact multipliers analysis, endogenous targeting, optimal control and rational expectations.

bimets does not depend on compilers or third-party software so it can be freely downloaded and installed on Linux, MS Windows® and Mac OSX®, without any further requirements.

Keywords: R, system of simultaneous equations, ols, instrumental variables, error autocorrelation, polynomial distributed lag, linear restrictions, incidence matrix, model simulation, forecasting, add-factors, exogenization, multipliers, endogenous targeting, model renormalization, stochastic simulation, optimal control, rational expectations.

1. Introduction

bimets is a software framework developed by using R language and designed for time se-

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

ries 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, also on rational expectations model, and 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.

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

2. 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 and are fully compatible with the base R class `ts()`.

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')

R> print(class(myTS))

[1] "ts"
```

The main **bimets** time series capabilities are:

- *Indexing*, par. 2.1;
- *Aggregation / Disaggregation*, par. 2.2;
- *Manipulation*, par. 2.3;

More details on **bimets** time series capabilities are available in the reference manual.

2.1. 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 year-period*: users can select and modify observations by providing the requested year and period as scalars or as bidimensional arrays, i.e. `ts[[year,period]]`, `ts[[start]]` and `ts[[start,end]]`, given `start <- c(year1,period1); end <- c(year2,period2)`;

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

– *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[[2000,14]]                 #get year 2000 period 14
R> start <- c(2000,20)
R> end <- c(2000,30)
R> myTS[[start]]                   #get year 2000 period 20
R> myTS[[start,end]]               #get from year-period 2000-20 to 2000-30
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
R> myTS[[start]] <- NA             #assign to year-period 2000-20
R> myTS[[start,end]] <- 3.14       #assign from year-period 2000-20 to 2000-30
R> myTS[[start,end]] <- -(1:11)    #assign multiple values
R>                                 #from year-period 2000-20 to 2000-30
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['2000-01-15'] <- NA        #assign to Jan 15, 2000
```

2.2. Time Series Aggregation / Disaggregation

The **bimets** package provides advanced (dis)aggregation capabilities, having linear interpolation capabilities in disaggregation, and several 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 using the average as aggregation fun
R> myYearlyTS <- YEARLY(myMonthlyTS, 'AVE')

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

2.3. 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, i.e. `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
```

3. Econometric Modeling

bimets econometric modeling capabilities comprehend:

- *Model Description Language*, par. 3.1;
- *Estimation*, par. 3.2
- *Structural Stability*, par. 3.4
- *Simulation*, par. 3.5
- *Rational Expectations*, par. 3.6
- *Stochastic Simulation*, par. 3.9
- *Multipliers Analysis*, par. 3.10
- *Endogenous Targeting*, par. 3.11
- *Optimal Control*, par. 3.12

We will go through each item of the list with a simple Klein[1] model example.

For more realistic scenarios, several advanced econometric exercises on the US Federal Reserve FRB/US econometric model (e.g., dynamic simulation in a monetary policy shock, rational expectations, endogenous targeting, stochastic simulation, etc.) are available in the "US Federal Reserve quarterly model (FRB/US) in R with bimets" vignette.

More details on **bimets** econometric capabilities are available in the reference manual.

3.1. Model Description 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 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.

[1] *"Economic Fluctuations in the United States 1921-1941"* by L. R. Klein, Wiley and Sons Inc., New York, 1950

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
   "
```

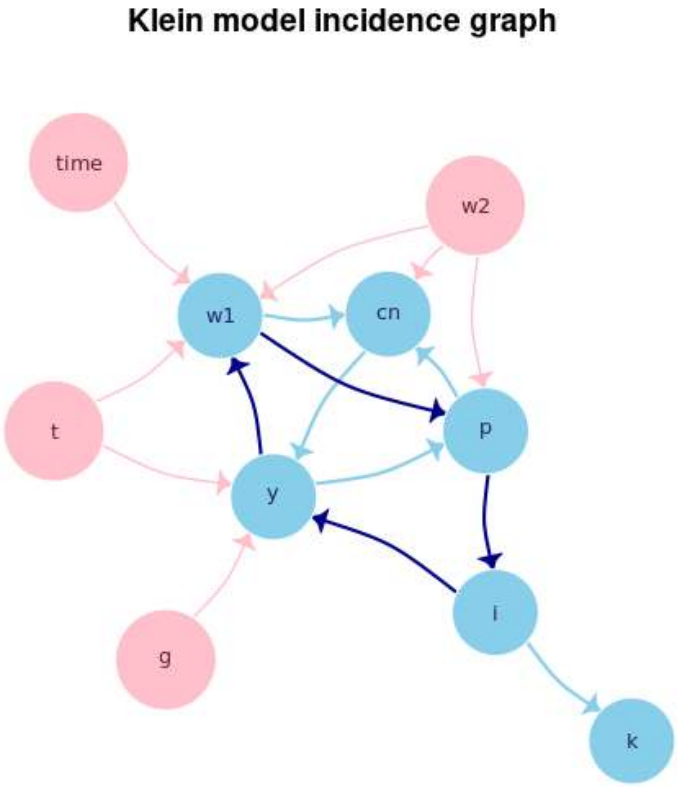
Given:

- `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 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. 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 names of the coefficients 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;

- `TSLEAD(ts,i)`: lead 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;

Note that **bimets** classifies a model as a forward-looking model if any model equation contains the `TSLEAD` time series function. More details about forward-looking models are available in the *Rational Expecations*, par. 3.6.

`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 nor 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 through the use of 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 error autocorrelation coefficients $\rho_{i+1} = \rho_{i+1,1}, ..., \rho_{i+1,n}$ by regressing the new residuals arising from step 4 (similarly 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 `[NOBS x NREG]` size 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 matrix R.

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 a 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 equation.

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 from the Tartaglia's-like triangle:

$$
\begin{array}{cccccc}
1 & -2 & 1 & & & \\
1 & -3 & 3 & -1 & & \\
1 & -4 & 6 & -4 & 1 & \\
1 & -5 & 10 & -10 & 5 & 1 \\
\ldots & \ldots & \ldots & \ldots & &
\end{array}
$$

where in the `i`-th row we will 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 the following example:
  `$This is a comment`

No other keywords are currently allowed in the `MDL` syntax.

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)"

R> kleinModel$behaviorals$cn$tsrange

[1] 1921    1 1941    1

R> kleinModel$behaviorals$cn$eqCoefficientsNames

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

R> kleinModel$behaviorals$cn$eqRegressorsNames

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

R> kleinModel$behaviorals$cn$eqSimExp

expression(cn[2, ] = cn__ADDFACTOR[2, ] + cn__COEFF__a1 * 1 +
    cn__COEFF__a2 * p[2, ] + cn__COEFF__a3 * (p[1, ]) + cn__COEFF__a4 *
    (w1[2, ] + w2[2, ]))
```

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 simulation, in order to guarantee performance and convergence, if any, with the aim of minimizing the number of *feedback* endogenous variables (see the *The Optimal Reordering*, par. 3.7).

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$vblocks[[1]]$vsim

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

R> kleinModel$vblocks[[1]]$vfeed

[1] "y"

R> kleinModel$vblocks[[1]]$vpost

[1] "k"
```

While simulating the Klein's model, **bimets** will iterate on the computation of, in order, `w1 -> p -> i -> cn -> y` (the `vsim` variables in the single block of equations `vblocks[[1]]`), by looking for convergence on `y` (the `vfeed` variable, only one in this example) that is the feedback variable for the block. If the convergence in the block 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 is applied to each block of equations.

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)

Load model data "kleinModelData" into model "klein1.txt"...
LOAD_MODEL_DATA(): warning, there are non-finite values in time series "time".
...LOAD MODEL DATA OK
```

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.

An advanced MDL model example follows (original time series are manually adjusted in order to fit the example):

```
R> lhsKlein1.txt <- "
   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"

R> #adjust the original data in order to estimate and to simulate the model
R> lhsKleinModelData <- within(kleinModelData,{
     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
   })

R> lhsKleinModel <- LOAD_MODEL(modelText = lhsKlein1.txt)
R> lhsKleinModel <- LOAD_MODEL_DATA(lhsKleinModel, lhsKleinModelData)

R> #ESTIMATE and SIMULATE functions are described later
R> lhsKleinModel <- ESTIMATE(lhsKleinModel)
R> lhsKleinModel <- SIMULATE(lhsKleinModel, TSRANGE = c(1925,1,1930,1))
```

3.2. 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*;

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 output omitted):

```
R> kleinModel <- ESTIMATE(kleinModel, quietly = TRUE)
```

Users can also estimate a selection of behavioral equations:

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

ESTIMATE(): warning, there are non-finite values in time series "time".

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
```

3.3. Advanced Estimation

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(): warning, there are non-finite values in time series "time".
...LOAD MODEL DATA OK

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

ESTIMATE(): warning, there are non-finite values in time series "time".

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 reached in  6  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
```

3.4. 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[2], 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`) 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 (i.e. *predictive failure*) 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:

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

[2]G. C. Chow, *Tests of equality between sets of coefficients in two linear regressions.* Econometrica, Vol 28, 4. July 1960

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`, and optionally by providing the argument `CHOWPAR` as an integer array, i.e. `c(year,period)`, built of the requested year and period in the extended `TSRANGE`.

Example:

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

ESTIMATE(): warning, there are non-finite values in time series "time".

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                  =   13.12755
                        T-stat. 6.504605    ***

                    +   0.1669801   p
                        T-stat. 2.183045

                    +   0.08856838  TSLAG(p,1)
                        T-stat. 0.9750418

                    +   0.887964    (w1+w2)
                        T-stat. 12.61002    ***


STATs:
R-Squared                       : 0.9787275
Adjusted R-Squared              : 0.972926
```

```
Durbin-Watson Statistic        : 1.379996
Sum of squares of residuals    : 6.918601
Standard Error of Regression   : 0.7930723
Log of the Likelihood Function : -15.4803
F-statistic                    : 168.7001
F-probability                  : 1.776731e-09
Akaike's IC                    : 40.96061
Schwarz's IC                   : 44.50086
Mean of Dependent Variable     : 50.91333
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            : 4.488731
F-prob(6,17)       : 0.006687229

Predictive Power:

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

      1936, 1   ,  57.7          ,  56.55436      ,  1.145638      ,  1.01181       ,  1.132265
      1937, 1   ,  58.7          ,  59.93099      , -1.230988      ,  1.020099      , -1.206734
      1938, 1   ,  57.5          ,  57.97212      , -0.4721225     ,  0.9686377     , -0.4874087
      1939, 1   ,  61.6          ,  61.52069      ,  0.0793139     ,  1.200479      ,  0.06606853
      1940, 1   ,  65            ,  65.39572      , -0.3957177     ,  1.242267      , -0.3185448
      1941, 1   ,  69.7          ,  73.79655      , -4.096547      ,  1.669299      , -2.454053

...ESTIMATE OK
```

3.5. 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*: in which the historical values for the lagged endogenous variables are used in the solutions of subsequent periods;
- *Dynamic simulations*: 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;
- *Stochastic Simulation*: see par. 3.9;
- *Residuals check*: a single period, single equation simulation; output simulated time series are just the RHS (right-hand-side) computation of their equation, by using the historical values of the involved time series and by accounting for error autocorrelation and PDLs, if any;
- *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, 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 their 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: that can be done in **bimets** by using the `ConstantAdjustment` argument of the `SIMULATE` function, as in step 3;

2. fill in with the estimated residuals all the constant adjustments for the behavioral equations, and fill in with zeroes the constant adjustments for the technical identities: that can be done in **bimets** by using the `SIMULATE` procedure with the option `simType='RESCHECK'`, then by analyzing and using the `ConstantAdjustmentRESCHECK` attribute of the simulated model, as in the following simulation in step 3.
3. perform a simulation of the model: that can be done in **bimets** by using the `SIMULATE` procedure with the option `ConstantAdjustment=ConstantAdjustmentRESCHECK`;
4. compute the difference between the historical and the simulated values for all the endogenous variables;
5. check whether all the differences assessed in step 4 are zero in whole time range, eventually accounting for the error autocorrelation in behaviorals.

An example on `ConstantAdjustmentRESCHECK` usage is available at the end of the `SIMULATE` help page in the reference manual;

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.

Back to Kelin's model example, let's forecast the GNP[3] (i.e. the "`y`" endogenous variable) up to 1944:

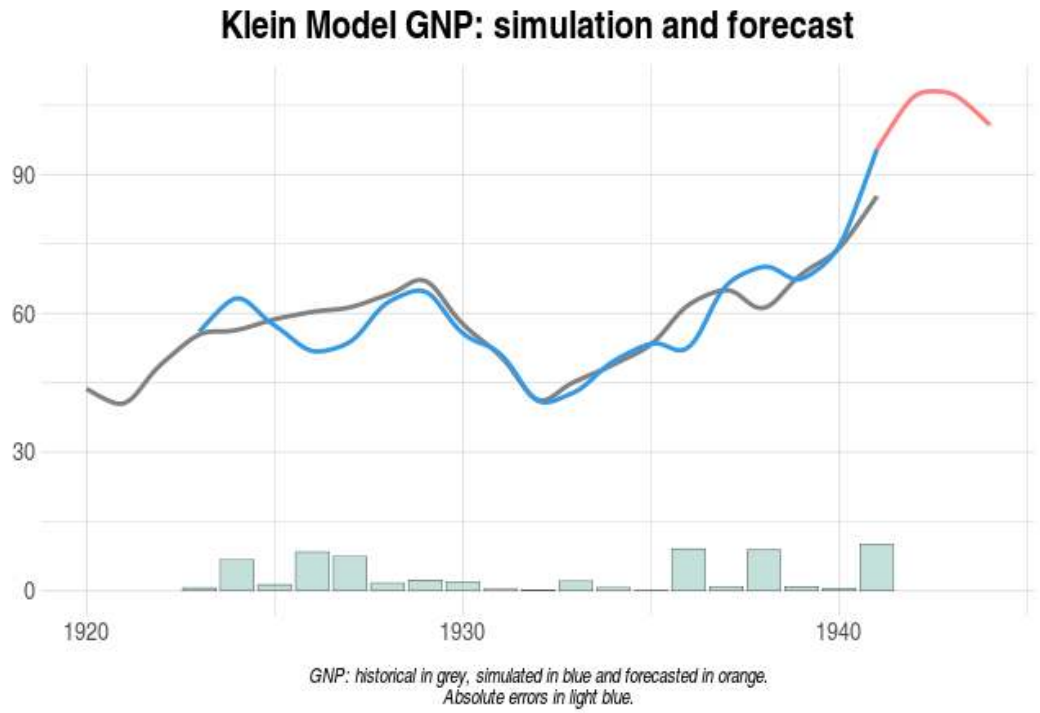
[3]originally referred as "Net national income, measured in billions of 1934 dollars", pag. 141 in *"Economic*

```
R> #FORECAST GNP in 1942:1944
R> #we need to extend exogenous variables in 1942 up to 1944
R> #in this exercise we perform a simple time series extension
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
               ,quietly = TRUE
               )
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
```

Fluctuations in the United States 1921-1941" by L. R. Klein, Wiley and Sons Inc., New York, 1950



3.6. Rational Expectations

bimets classifies a model as a forward-looking model if any model equation contains the `TSLEAD` time series function. Forward-looking models assume that economic agents have complete knowledge of an economic system and calculate the future value of economic variables correctly according to that knowledge. Thus, forward-looking models are called also rational expectations models and, in macro-econometric models, model-consistent expectations.

In forward-looking models, simulation complexity arises, and all simulation periods must be solved simultaneously because equations can contain references to past and future values. Thus, given `N` simulation periods requested by the user, each model equation must be replicated `N-1` times and modified before the simulation takes place, accounting for lead transformations. Finally, the extended model must be simulated as a single block of equations.

Internal data structures too, such as the incidence and the Jacobian matrix, and the reordering arrays `vpre` and `vblocks` (described later in the *The Optimal Reordering*, par. 3.7) section), grow with the number of simulation periods requested by the user. Therefore, they can only be calculated when a new simulation is requested rather than when the model `MDL` definition is parsed, further extending computational time in simulation.

To understand **bimets** internals when dealing with forward-looking models, please consider the following simple example of a forward-looking model having a single identity:

```
IDENTITY> Y
EQ> Y = TSLEAD(Y) - TSLAG(Y) + X
```

Given `X` as an exogenous variable, if the requested simulation has a `TSRANGE` that spans two periods, then the model will be internally transformed into something like:

```
IDENTITY> Y
EQ> Y = Y__LEAD__1 - TSLAG(Y) + X

IDENTITY> Y__LEAD__1
EQ> Y__LEAD__1 = TSLEAD(Y,2) - Y + TSLEAD(X)
```

Accordingly, the model will be simulated only on the first period of the `TSRANGE`. Please note that `TSLAG(Y)` in the first equation, and `TSLEAD(Y,2)` in the second equation, are a kind of exogenous variables and must be defined in order for the simulation to be completed. Moreover, `Y` and `Y__LEAD__1` are simultaneously involved in the iterative simulation algorithm, and both depend on each other, as also stated in the incidence matrix for the extended model:

```
$incidence_matrix
             Y  Y__LEAD__1
 Y           0           1
 Y__LEAD__1  1           0
```

Due to the mechanism described above, only `DYNAMIC` simulations are allowed in forward-looking models. A Klein-like forward-looking model example is available in the `SIMULATE` help page of the reference manual.

3.7. The Optimal Reordering

In fact, the simulation process takes advantage of an appropriate equations reordering to increase the performances by iteratively solving only one subset of equations, while the others are solved straightforwardly[4].

For backward-looking models, 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 par. 3.1 wherein the content of the `kleinModel$incidence_matrix`

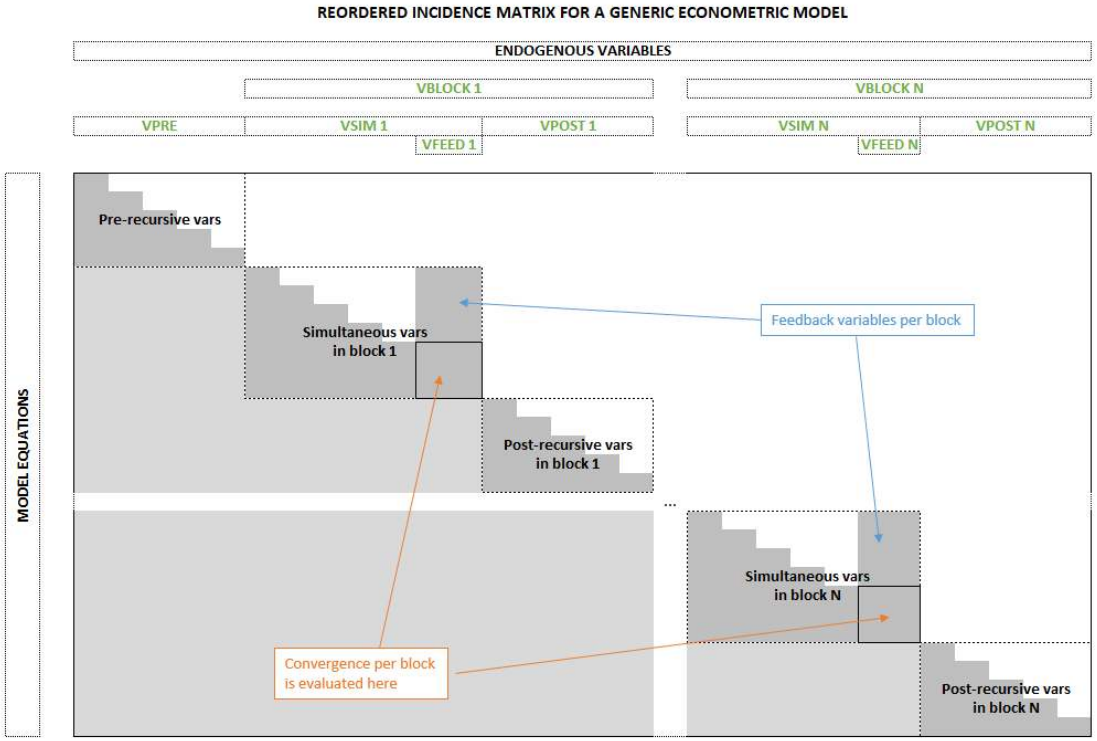
[4] *"...a different ordering of the equations can substantially affect the speed of convergence of the algorithm; indeed some orderings may produce divergence. The less feedback there is, the better the chances for fast convergence..."* - Don, Gallo - Solving large sparse systems of equations in econometric models - Journal of Forecasting 1987.

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 the whole model to be recursive. Indeed the incidence graph is often cyclic, as in the Klein's model that presents the following circular dependencies in the incidence matrix: `p <- w1 <- y <- i <- p`, as shown in the par. 3.1 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 a 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 both 0's and 1's. The 1's in the light gray areas refer to variables already assessed in previous subset of equations, therefore they are known terms within the current subset The 1's in the dark gray areas refer to variables assessed within the current subset.



In **bimets**, the final pattern of an incidence matrix after the equation reordering generally features `N+1` blocks:

1. One a recursive subset of equations, i.e. the pre-recursive `VPRE` in image;

2. `N` blocks of equations, `VBLOCK` in image, each built with a simultaneous `VSIM` and a post-recursive `VPOST` subset of equations;

As said, the pre-recursive and the post-recursive subsets are lower triangular. Therefore the corresponding equations are solvable with a cascade substitution with no iteration. Only the simultaneous equations subset needs an iterative algorithm to be solved. It is important to say that the convergence criterion may also be applied to feedback 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 i) computes the strongly connected component of the related incidence graph by using the Tarjan algorithm[5], and ii) orders the equations in pre-recursive and, for each block of equations, in simultaneous and post-recursive subsets. The simultaneous subsets are then analyzed in order to find a minimal set of feedback variables. This last problem is known to be NP-complete[6].

The optimal reordering of the model equations is programmatically achieved through the

[5]Tarjan, Robert - *Depth-first search and linear graph algorithms* - SIAM Journal on Computing - June 1972

[6]Garey, Johnson - *Computers and Intractability: a Guide to the Theory of NP-completeness* - San Francisco, Freeman 1979

use of an iterative algorithm applied to the incidence matrix that can produce `1+3*N` ordered lists of endogenous variables:

1. One list `vpre` is the ordered list containing the names of the endogenous pre-recursive variables to be sequentially computed (once per simulation period) before the simulation iterative algorithm takes place;
2. For each of the `N` elements in the `vblocks` list:
   (a) One list `vsim` (the simultaneous subset) that is the ordered list containing the names of the endogenous variables to be sequentially computed during each iteration of the simulation iterative algorithm;
   (b) One list `vfeed` that is the list containing the names of the endogenous feedback variables; generally `vfeed` are the last variables of the ordered `vsim` list in the same block;
   (c) One list `vpost` that is the ordered list containing the names of the endogenous post-recursive variables to be sequentially computed (once per simulation period) after the simulation iterative algorithm has found a solution in the previous simultaneous subset in the same block;

Once 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);
- for each block in `vblocks`:
  - iteratively compute the system of simultaneous equations (i.e. the equation of the endogenous variables in the `vsim` ordered list): for each iteration i) update the values of the current endogenous variables, ii) update the feedback variables accordingly to the simulation algorithm in use (see next section for details on simulation algorithms) and iii) 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);
- finally, save the solutions;

Clearly, each endogenous variable is updated accordingly to its related equation `EQ>` in the `MDL` model definition.

In forward-looking models, the incidence matrix and the equations reordering depend on the simulation periods count, therefore the model attributes `incidence_matrix`, `vblocks`, and `vpre` are calculated only after a simulation has been initialized, and will be available to users in the `model$simulation[['__SIM_PARAMETERS__']]` lists.

The reader can see an equations reordering example in the *Model Description Language*, par. 3.1, wherein the content of the `kleinModel$vpre` and `kleinModel$vblocks` variables are printed out.

3.8. The Simulation Algorithms

Given x_j the j-exogenous variable, $j = 1..m$, and $y_{i,k}$ the value of the i-endogenous variable, $i = 1..n$, in the simultaneous block at the iteration k, with i 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 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 block of equations. 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 tested at each iteration's end on the feedback variables.

Newton-Raphson's 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 econome-

ric 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.

The simulation of a non-trivial model, if computed by using the same data but on different hardware, software or numerical libraries, may produce 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.

Below is an example of advanced simulation using the Newton-Raphson algorithm:

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

Below is an example of a Klein model forecasting exercise in three alternative exogenous scenarios:

```
R> #COMPARE FORECAST IN 3 ALTERNATIVE
R> #EXOGENOUS SCENARIOS
```

```
R>
R> #create 3 new models for the 3 scenarios
R> modelScenario1 <- advancedKleinModel
R> modelScenario2 <- advancedKleinModel
R> modelScenario3 <- advancedKleinModel
R> #scenario 1, define exogenous paths
R> modelScenario1$modelData <- within(modelScenario1$modelData,{
                k    = TSEXTEND(k,   UPTO=c(1943,1))
                w2   = TSEXTEND(w2,  UPTO=c(1943,1))
                t    = TSEXTEND(t,   UPTO=c(1943,1))
                g    = TSEXTEND(g,   UPTO=c(1943,1))
                time = TSEXTEND(time,UPTO=c(1943,1)
                                    ,EXTMODE='LINEAR')
   })
R> #scenario 2, define exogenous paths
R> modelScenario2$modelData <- within(modelScenario2$modelData,{
                k    = TSEXTEND(k,   UPTO=c(1943,1))
                w2   = TSEXTEND(w2,  UPTO=c(1943,1))
                t    = TSEXTEND(t,   UPTO=c(1943,1))
                g    = TSEXTEND(g,   UPTO=c(1943,1)
                              ,EXTMODE='LINEAR')
                time = TSEXTEND(time,UPTO=c(1943,1)
                              ,EXTMODE='LINEAR')
   })
R> #scenario 3, define exogenous paths
R> #we also change consumption cn add-factor
R> modelScenario3$modelData <- within(modelScenario3$modelData,{
                k    = TSEXTEND(k,   UPTO=c(1943,1))
                w2   = TSEXTEND(w2,  UPTO=c(1943,1)
                                ,EXTMODE='MEAN4')
                t    = TSEXTEND(t,   UPTO=c(1943,1))
                g    = TSEXTEND(g,   UPTO=c(1943,1)
                                ,EXTMODE='LINEAR')
                time = TSEXTEND(time,UPTO=c(1943,1)
                                ,EXTMODE='LINEAR')
   })
R> constantAdjListScenario3 <- constantAdjList
R> constantAdjListScenario3$cn[[1941,1]] <- c(1,2,3)
R> #simulate the 3 models
R> modelScenario1 <- SIMULATE(modelScenario1
                            ,simAlgo='NEWTON'
                            ,simType='FORECAST'
                            ,TSRANGE=c(1940,1,1943,1)
                            ,simConvergence=1e-5
                            ,simIterLimit=20
                            ,quietly=TRUE)
R> modelScenario2 <- SIMULATE(modelScenario2
                            ,simAlgo='NEWTON'
```

```
                                ,simType='FORECAST'
                                ,TSRANGE=c(1940,1,1943,1)
                                ,simConvergence=1e-5
                                ,simIterLimit=20
                                ,quietly=TRUE)
R> modelScenario3 <- SIMULATE(modelScenario3
                                ,simAlgo='NEWTON'
                                ,simType='FORECAST'
                                ,TSRANGE=c(1940,1,1943,1)
                                ,simConvergence=1e-5
                                ,simIterLimit=20
                                ,ConstantAdjustment=constantAdjListScenario3
                                ,quietly=TRUE)
R> #compare results on GNP
R> TABIT(modelScenario1$simulation$y,
         modelScenario2$simulation$y,
         modelScenario3$simulation$y)

      Date, Prd., modelScenario1$simulation$y, modelScenario2$simulation$y, modelScenario3$simulation$y

      1940, 1   ,  85.90719      ,  85.90719      ,  85.90719
      1941, 1   ,  140.018       ,  140.018       ,  147.4762
      1942, 1   ,  256.1528      ,  233.0955      ,  258.7197
      1943, 1   ,  461.0662      ,  349.0985      ,  404.442
```

3.9. 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 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; 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 distribution and time range for the disturbance. 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 example, we will perform a stochastic forecast of the previously estimated `advancedKleinModel`. The advanced Klein model will be perturbed during the forecast operation by applying a normal disturbance to the endogenous *Consumption* behavioral `cn` 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 its 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> 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)
           )
     )

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
                         ,quietly = TRUE)
```

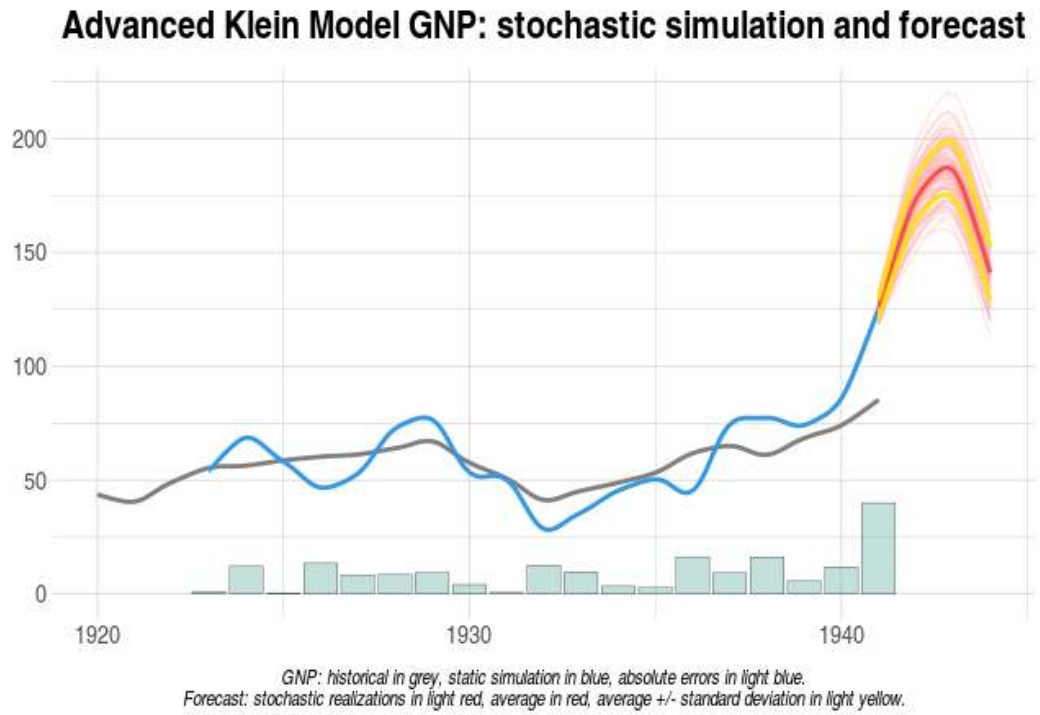
```
R> #print mean and standard deviation of forecasted GNP
R> with(advancedKleinModel$stochastic_simulation, TABIT(y$mean, y$sd))
```

```
      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]))
```

```
         [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
[1,] 125.2511 121.3591 123.7998 120.3449 121.0243 123.0448
[2,] 172.3125 170.2987 174.5269 170.0456 169.3925 168.6419
[3,] 185.2961 186.5037 187.3361 185.0368 177.2287 186.3369
[4,] 140.8831 145.1024 139.5191 139.5570 135.5024 151.5389
```



At the moment, all the disturbances are i.i.d. and are not transformed into a congruent autoregressive scheme in case the related perturbed endogenous behavioral presents an autocorrelation for the errors in its `MDL` definition, e.g. `ERROR> AUTO(n)`

Users can also pass an arbitrary matrix to the stochastic simulation algorithm, by using the `TYPE='MATRIX'` directive in the `StochStructure` argument: in this case `PARS=matrix` with `matrix` as a `[ TSRANGE x StochReplica ]` matrix. An example follows:

```
R> TSRANGE <- c(1935,1,1940,1)
R> StochReplica <- 100

R> #we will perturb simulation by using regression residuals
R> #get cn and i residuals in TSRANGE
R> cn_residuals <- TSPROJECT(advancedKleinModel$behaviorals$cn$residuals,
                             TSRANGE=TSRANGE,
                             ARRAY = TRUE)
R> i_residuals <- TSPROJECT(advancedKleinModel$behaviorals$i$residuals,
                            TSRANGE=TSRANGE,
                            ARRAY = TRUE)

R> #define stochastic matrices
R> cn_matrix <- c()
R> i_matrix <- c()
R> #populate matrices
```

```
R> 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)
   }

R> #define stochastic structure
R> myStochStructure <- list(
     cn=list(
       TSRANGE=TRUE,
       TYPE='MATRIX',
       PARS=cn_matrix
     ),
     i=list(
       TSRANGE=TRUE,
       TYPE='MATRIX',
       PARS=i_matrix
     )
   )

R> #stochastic simulation
R> advancedKleinModel <- STOCHSIMULATE(advancedKleinModel
                                       ,TSRANGE=TSRANGE
                                       ,StochStructure=myStochStructure
                                       ,quietly = TRUE)

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

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

      1935, 1   ,  50.22247      ,  3.077131
      1936, 1   ,  39.53867      ,  21.64031
      1937, 1   ,  35.53497      ,  23.75123
      1938, 1   ,  45.1226       ,  12.42018
      1939, 1   ,  58.64743      ,  9.648934
      1940, 1   ,  76.91834      ,  23.7928
```

3.10. 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 "Simulation" section, 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`;

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 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 $X_{shocked}(t) = X(t)(1 + MM\_SHOCK)$

bimets users can also declare an endogenous variable as the `INSTRUMENT` variable. In this

case, the constant adjustment (see *Simulation* 3.5) 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
                            ,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 (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`" 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:      50.00 %
Multiplier Matrix:     100.00 %
...MULTMATRIX OK

R> #output multipliers matrix (note the zeros when 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
```

3.11. Endogenous Targeting

The endogenous targeting[7] 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()` function 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.

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 `n` endogenous variables defined as `TARGET`;
2. there exists a set of `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 time series 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:
   $V_{exog}(i+1) = V_{exog}(i) + \texttt{MULTMAT}^{-1} * (V_{endog}(i) - \texttt{TARGET})$, 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 *Simulation* 3.5) 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` function will return the `INSTRUMENT` time series allowing us to achieve the desired values for the endogenous variables designated as `TARGET`.

[7] On the Theory of Economic Policy - Tinbergen J. 1952

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 (output omitted):

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

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
                           ,quietly = TRUE)

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
```

3.12. 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, (see *Endogenous Targeting*, par. 3.11) . 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` (see *Stochastic Simulation*, par. 3.9) 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.

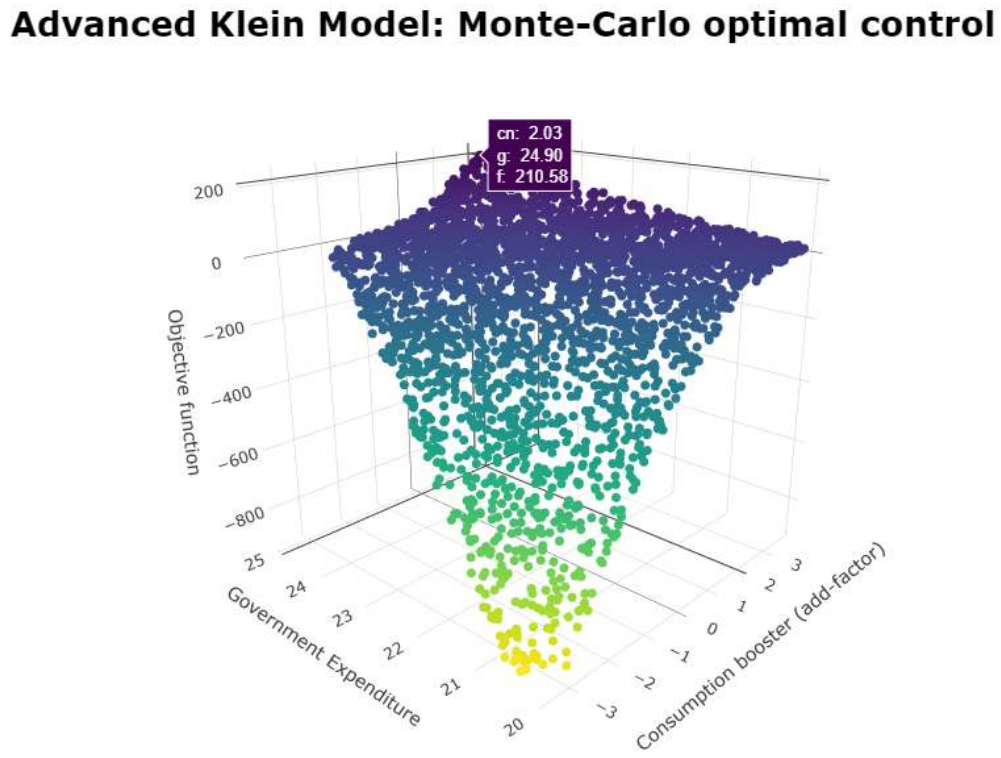
In the following figure, the scatter plot is populated with `2916` objective function stochastic realizations, computed by using the example code at the end of this section; the `210.58` local maximum is highlighted (i.e. `advancedKleinModel$optimize$optFunMax` in code example).

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 $g \in (15, 25)$.

Optimal control example of the previously defined `advancedKleinModel` follows:

```
R> #load the advanced model
R> advancedKleinModel <- LOAD_MODEL(modelText = advancedKlein1.txt
                                    ,quietly = TRUE)

R> #load time series into the model object
R> advancedKleinModel <- LOAD_MODEL_DATA(advancedKleinModel
                                         ,kleinModelData
                                         ,quietly = TRUE)

R> #estimate the model
R> advancedKleinModel <- ESTIMATE(advancedKleinModel
                                  ,quietly = TRUE)

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
                             ,quietly = TRUE)

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
                     ,quietly = TRUE
   )

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
```

A more complex example is available in the `OPTIMIZE` help page of the reference manual.

4. Computational Details

The iterative simulation procedure is the most time-consuming operation of the **bimets** package. For small models, this operation is quite immediate; on the other hand, the simulation of models that count hundreds of equations could last for minutes, especially if the requested operation involves a parallel simulation having hundreds of realizations per equation. This could be the case for the endogenous targeting, the stochastic simulation and the optimal control.

The `SIMULATE` code has been optimized in order to minimize the execution time in these cases. In terms of computational efficiency, the procedure takes advantage of the fact that multiple datasets are bound together in matrices, therefore in order 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 `SIMULATE` algorithm transforms time series into matrices and consequently can easily bind multiple datasets by column. At the same time, the single run ensures a fast code execution, while each column in the output matrices represents a stochastic or perturbed realization.

The above approach is even faster if R has been compiled and linked to optimized multi-threaded numerical libraries, e.g. Intel® MKL, OpenBlas, Microsoft® R Open, etc.

Finally, model equations are pre-fetched into sorted R expressions, and an optimized R environment is defined and reserved to the `SIMULATE` algorithm; this approach removes the overhead usually caused by expression parsing and by the R looking for variables inside nested environments.

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.

Several advanced econometric exercises on the US Federal Reserve FRB/US econometric model (e.g., dynamic simulation in a monetary policy shock, rational expectations, endogenous targeting, stochastic simulation, etc.) are available in the "US Federal Reserve quarterly model (FRB/US) in R with bimets" vignette.

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® and Mac OSX®, without any further requirements.

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)
```

References

[1] F. J. Henk Don and Giampiero M. Gallo *Solving large sparse systems of equations in econometric models.* Journal of Forecasting, 6(3):167-180, 1987.

[2] Jan Tinbergen *On the theory of economic policy.* North-Holland, Amsterdam, 1952.

[3] Daniel Gabay, Pierre Nepomiastchy, M'Hamed Rachdi and Alain Ravelli *Numerical methods for simulation and optimal control of large-scale macroeconomic models.* Applied stochastic control in econometrics and management science:115-158, 1980

[4] M. R. Garey, D. S. Johnson *Computers and Intractability: a Guide to the Theory of NP-completeness.* San Francisco, Freeman 1979

[5] G. C. Chow, *Tests of equality between sets of coefficients in two linear regressions.* Econometrica, Vol 28, 4. July 1960

[6] L. R. Klein *Economic Fluctuations in the United States.* Wiley and Sons Inc., New York, 1950

[7] Board of governors of The Federal Reserve System - *pyfrbus: a Python-based platform to run simulations with the FRB/US model.*. python package version 1.0.0, https://www.federalreserve.gov/econres/us-models-about.htm

Affiliation:

Andrea Luciani
Bank of Italy
Directorate General for Economics, Statistics and Research
Via Nazionale, 91
00184, Rome - Italy
E-mail: andrea.luciani@bancaditalia.it