bimets:
Time Series and Econometric Modeling in R

Andrea Luciani
Bank of Italy*

Abstract

bimets is an R package developed with the aim of easing time series analysis and building 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, static and dynamic simulation of simultaneous equations with exogenizations and add-factors, interim and impact multipliers analysis, endogenous targeting with model "renormalization".

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, model renormalization.

1. Introduction

bimets is a software framework developed by using R language and designed for time series analysis and econometric modeling, which allows creating and manipulating time series, specifying simultaneous equation models of any size by using a kind of high-level description

*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.*

language, and performing model estimation, simulation and forecasting.

In addition, **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.

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 we can find 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 using different commercial software has the same average magnitude.

bimets stands for Bank of Italy Model Easy Time Series; it does not depend on compilers or third-party software so it can be freely downloaded and installed on Linux, MS Windows® 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)
```

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 supproted. The time series are created by the `TIMESERIES()` function.

Example:

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

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

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

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

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

Example:

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

R> myTS[1:3]                      #get first three obs.
R> myTS['2000-01-12']             #get Jan 12, 2000 data
R> myTS['2000-02-03/2000-02-14']  #get Feb 3 up to Feb 14
R> myTS[[2000,14]]                #get year 2000 period 14
R> myTS['2000-01-15'] <- NA        #assign to Jan 15, 2000
R> myTS[[2000,42]] <- NA           #assign to Feb 11, 2000
R> myTS[[2000,100]] <- c(-1,-2,-3) #extend time series starting from period 100
```

2.2. Time Series Aggregation/Disaggregation

The **bimets** package provides advanced (dis)aggregation capabilities, with several linear interpolation capabilities in disaggregation, and many 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 annual time series using the average as aggregation fun
R> myAnnualTS <- ANNUAL(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()`
- Lag differences, absolute and percentage `TSDELTA()` `TSDELTAP()`
- 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, PER, myMovAveTS     , myTS1

  ago 2004, 8  ,               ,  56
  set 2004, 9  ,               ,  57
  ott 2004, 10 ,  3.849002     ,  58
  nov 2004, 11 ,  3.776275     ,  59
  dic 2004, 12 ,  3.706247     ,  60
```

3. Econometric Modeling

bimets econometric modeling capabilities comprehend:

- *Model Definition Language*
- *Estimation*
- *Simulation*
- *Multipliers Analysis*
- *Renormalization (Tinbergen Classification)*

We will go through each item of the list with a simple Tinbergen-Klein model example.

3.1. Model Definition Language

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

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

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

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

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

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

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

```
COEFF> c1 c2 c3 c4

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

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

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

END
"
```

Please note that there are circular dependencies between equations of the model, i.e. `cn <- w1 <- y <- cn`. Circular dependencies imply that the model simulation must be solved with an iterative algorithm.

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

- **EQUATION>** or **BEHAVIORAL>** indicate the beginning of a series of keyword statements describing a behavioral equation. The behavioral statement general form is:
  `BEHAVIORAL> name [TSRANGE startYear, startPeriod, endYear, endPeriod]`
  where `name` is the name of the behavioral equation and the optional `TSRANGE` specifies that the provided time interval must be used in the coefficients estimation. 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> name = coeff1*f1 + coeff2*f2 + coeff3*f3 + ...`
where `name` is the name 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> name = f1 + f2 + f3 + ...`
where `name` is the name of the identity variable and
`f1, f2, f3, ...` are functions of variables.

The mathematical expression can include the standard arithmetic operators, parentheses and `MDL` functions described later in this section. `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 is available for the specified error structure: `n` periods of data prior to the time interval specified by `TSRANGE` are requested 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 the 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
```

Ccoefficients `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) In a similar way, 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 an 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 the left to the 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`

The elements of this matrix 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:

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

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 the value of a logical expression. 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 `MDL` functions listed below 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)
```

– **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. A comment 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.

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

- TSLAG(ts,i): lag the ts time series by i-periods;
- TSDELTA(ts,i): i-periods difference of the ts time series;
- 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;

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, the user can ask **bimets** about any details of this model.

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

```
R> kleinModel$behaviorals$cn

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

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

$eqCoefficientsNamesOriginal
[1] "a1" "a2" "a3" "a4"
```

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

$tsrange
[1] 1921    1 1941    1

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

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

$eqSimExp
expression(cn[2, ] = cn_ADDFACTOR[2, ] + cn_a1 * 1 + cn_a2 *
    p[2, ] + cn_a3 * (p[1, ]) + cn_a4 * (w1[2, ] + w2[2, ]))
```

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 an important 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" section).

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

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

```
R> kleinModel$incidence_matrix

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

R> kleinModel$vpre

NULL

R> kleinModel$vsim

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

R> kleinModel$vfeed

[1] "y"
```

```
R> kleinModel$vpost

[1] "k"
```

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

Once the model has been parsed, users needs 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 OK
```

Since time series and other data (e.g. regressor coefficients, error coefficients, constant adjustments, targets, instruments, etc...) are stored into 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 into it, i.e. endogenous and exogenous time series, estimated coefficients, constant adjustments, simulation options, simulated time series, calculated instruments, targets, etc.

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*;

Restrictions procedure derives from the theory of Lagrange Multipliers, while the Cochrane-Orcutt method allows to account 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 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 into 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 example

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 conditional equation evaluations

   COMMENT> Consumption with autocorrelation on errors
   BEHAVIORAL> cn
   TSRANGE 1925 1 1941 1
   EQ> cn =  a1 + a2*p + a3*TSLAG(p,1) + a4*(w1+w2)
   COEFF> a1 a2 a3 a4
```

```
   ERROR> AUTO(2)

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

   COMMENT> Demand for Labor with PDL
   BEHAVIORAL> w1
   TSRANGE 1925 1 1941 1
   EQ> w1 = c1 + c2*(y+t-w2) + c3*TSLAG(y+t-w2,1) + c4*time
   COEFF> c1 c2 c3 c4
   PDL> c3 1 2

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

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

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

   END"

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

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

R> advancedKleinModel <- LOAD_MODEL_DATA(advancedKleinModel,kleinModelData)

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

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

Estimate the Model advancedKlein1.txt:
the number of behavioral equations to be estimated is 3.
The total number of coefficients is 13.

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

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

Convergence was reached in  9  /  20  iterations.


cn                  =   19.01352
                        T-stat. 13.1876     ***

                    +   0.3442816   p
                        T-stat. 3.841051    **

                    +   0.03443117  TSLAG(p,1)
                        T-stat. 0.4280928

                    +   0.6993905   (w1+w2)
                        T-stat. 15.30744    ***

ERROR:  AUTO(2)

AUTOREGRESSIVE PARAMETERS:
Rho             Std. Error      T-stat.
 0.05743131      0.3324101       0.1727725
 0.007785936     0.2647013       0.02941404


STATs:
R-Squared                       : 0.985263
Adjusted R-Squared              : 0.979595
Durbin-Watson Statistic         : 1.966609
Sum of squares of residuals     : 9.273455
Standard Error of Regression    : 0.8445961
Log of the Likelihood Function : -20.14564
F-statistic                     : 173.8271
F-probability                   : 1.977107e-11
Akaike's IC                     : 54.29129
Schwarz's IC                    : 60.90236
Mean of Dependent Variable      : 55.71765
```

```
Number of Observations         : 19
Number of Degrees of Freedom   : 13
Current Sample (year-period)   : 1925-1 / 1941-1


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



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

BEHAVIORAL EQUATION: i
Estimation Technique: OLS

i                   =   2.868104
                        T-stat. 0.3265098

                    +   0.5787626   p
                        T-stat. 4.456542    ***

                    +   0.4212374   TSLAG(p,1)
                        T-stat. 3.243579    **

                    -   0.09160307  TSLAG(k,1)
                        T-stat. -2.11748

RESTRICTIONS:
b2+b3=1

RESTRICTIONS F-TEST:
F-value            : 8.194478
F-prob(1,15)       : 0.0118602


STATs:
R-Squared                      : 0.8928283
Adjusted R-Squared             : 0.8794319
Durbin-Watson Statistic        : 1.173106
Sum of squares of residuals    : 26.76483
Standard Error of Regression   : 1.293368
Log of the Likelihood Function : -30.215
F-statistic                    : 66.64659
F-probability                  : 1.740364e-08
Akaike's IC                    : 68.43001
Schwarz's IC                   : 72.20776
Mean of Dependent Variable     : 1.310526
Number of Observations         : 19
Number of Degrees of Freedom   : 16
```

```
Current Sample (year-period)   : 1923-1 / 1941-1


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



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

BEHAVIORAL EQUATION: w1
Estimation Technique: OLS

w1                  =   1.103637
                        T-stat. 0.7290738

                    +   0.4358984   (y+t-w2)
                        T-stat. 11.35698    ***

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

                    +   0.1363549   time
                        T-stat. 3.398964    **

PDL:
c3 1 2

Distributed Lag Coefficient: c3
Lag     Coeff.          Std. Error     T-stat.
0        0.1212886       0.06620502     1.832015
1        0.0354339       0.04657983     0.7607135
SUM      0.1567225       0.04163457


STATs:
R-Squared                       : 0.9891508
Adjusted R-Squared              : 0.9855344
Durbin-Watson Statistic         : 2.219659
Sum of squares of residuals     : 6.3545
Standard Error of Regression    : 0.7276962
Log of the Likelihood Function : -15.75753
F-statistic                     : 273.5171
F-probability                   : 1.130929e-11
Akaike's IC                     : 43.51506
Schwarz's IC                    : 48.51434
Mean of Dependent Variable      : 37.69412
Number of Observations          : 17
Number of Degrees of Freedom    : 12
```

```
Current Sample (year-period)   : 1925-1 / 1941-1


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


...ESTIMATE OK
```

3.4. 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, and 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;
- *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 the 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 another exogenous time series - the "constant adjustment" - in the equation of the selected endogenous variables;

In details, the generic model suitable for simulation in **bimets** can be written as:

$$
\begin{aligned}
y_1 &= f_1(\bar{x}, \bar{y}) \\
&\dots \\
y_n &= f_n(\bar{x}, \bar{y})
\end{aligned}
$$

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 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 you will need to:

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 results of the simulation in a given time period depends on the results of the simulation 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 results of the simulation in a given time period does not depend on the results of the simulation 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 single time period through the use of historical values for each variable on the right-hand side of their equation, for both lagged and current periods. This kind of simulation is very helpful for debugging and checking 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 the model-equation and the data is carried out by means of a procedure called "Residual Check".

It consists in the following steps:

1. add another exogenous variable - the constant adjustment - to every equation of the model, both behavioral and technical identity (i.e. by using the `ConstantAdjustment` argument of the `SIMULATE()` function);
2. fill in with the estimated residuals all the constant adjustments for the behavioral equations;
3. fill in with zeroes the constant adjustments for the technical identities;
4. perform a simulation of the model with the `simType='RESCHECK'` option;
5. compute the difference between the historical and the simulated values for all the endogenous variables;
6. check whether all the differences assessed in step 5 are zero in the whole time range;

If a perfect tracking of the history is obtained then the equations have been written coherently with the data, otherwise a simulated equation not tracking the historical values is an unambiguous symptom of data inconsistent with the model definition.

Aside from the residual check, the add-factors constitute an important tool to significantly improve the accuracy of forecasts made through an econometric model. Considering the following model:

$$y_1 = f_1(\bar{x}, \bar{y}) + z_1$$
$$...$$
$$y_n = f_n(\bar{x}, \bar{y}) + z_n$$

the add-factors $\bar{z} = [z_1, ..., z_n]$ can be interpreted as estimates of the future values of the disturbance terms or, alternatively, as adjustments of the intercepts in each equation. These add-factors round out the forecasts, by summarizing the effects of all the systematic factors not included in the model. One choice for the computation of the add-factors is given by

past estimation residuals and past forecast errors or by an average of these errors. This consideration suggests an easy way of computing the add-factors:

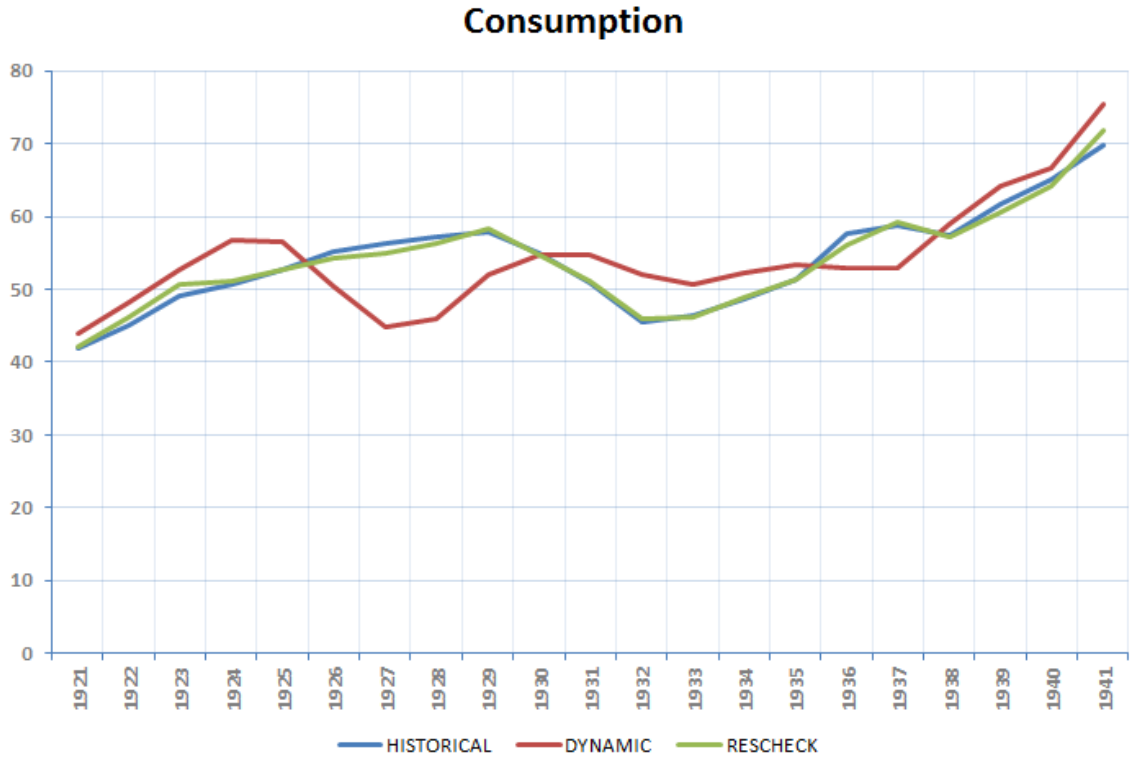
1. add the constant adjustments to every equation of the model, both behavioral and technical identity;
2. fill in with zeroes all the constant adjustments;
3. solve the model, with the `simType='RESCHECK'` option, in a time interval including some periods beyond the estimation sample;
4. compute the difference between the historical and the simulated values for each endogenous variables;
5. average, or process in a suitable way, the difference arising from point 4 in the time periods beyond the estimation sample to compute the constant value to be used as an add-factor in the following forecasting exercises;

Back to Kelin's model example, let's forecast the GNP (i.e. the "`y`" endogenous variable) up to 1943:

```
R> #FORECAST GNP in 1942 and 1943
R> #we need to extend exogenous variables in 1942 and 1943
R> kleinModel$modelData <- within(kleinModel$modelData,{
                        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> #simulate model
R> kleinModel <- SIMULATE(kleinModel
                      ,simType='FORECAST'
                      ,TSRANGE=c(1940,1,1943,1)
                      ,simConvergence=0.00001
                      ,simIterLimit=100
                      )

Simulation:     25.00%
Simulation:     50.00%
Simulation:     75.00%
Simulation:    100.00%
...SIMULATE OK

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

      DATE, PER, kleinModel$simulation$y

      1940, 1   ,  74.57806
```

```
1941, 1   ,   94.01525
1942, 1   ,   133.9687
1943, 1   ,   199.9133
```

In the following figure you will find the historical consumption, the dynamic simulated consumption and the RHS computation of the consumption equation, from 1921 to 1941:



3.5. The Optimal Reordering

In fact, the simulation process takes advantage of an appropriate ordering of the equations to increase the performances by iteratively solving only one subset of equations, while the others are solved straightforwardly[1].

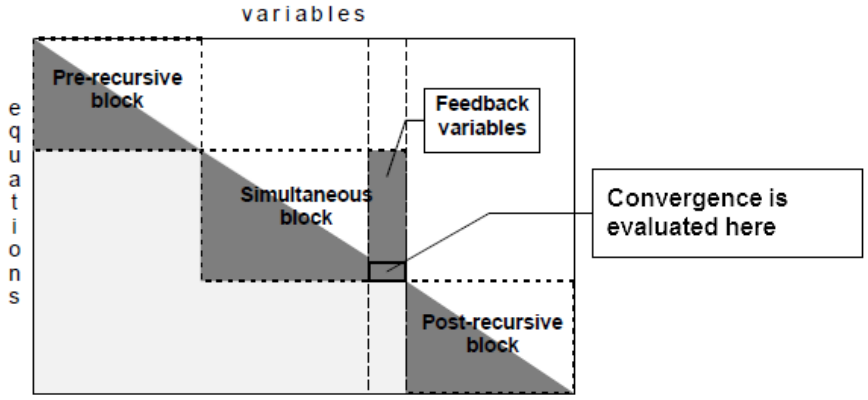
The `LOAD_MODEL()` function builds the incidence matrix of the model, then defines the proper equation reordering. The incidence matrix is built from the equations of the model; it is a square matrix in which each row and each column represents an endogenous variable. If the `(i,j)` element is equal 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.

[1] *"...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.

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 certain 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 right 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 dependecies in the incidence matrix: `cn <- w1 <- y <- cn`

For a subset of the equations, some 1's will occur in the upper triangle of the incidence matrix for all possible orderings. Such subset of equations is called *simultaneous*. In order to be able 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 contains both 0's and 1's. The 1's in the light gray areas refer to variables already assessed in previous blocks, therefore they are known terms within the block. The 1's in the dark gray areas refer to variables assessed within the block.



The final pattern of an incidence matrix after the equation reordering generally features three blocks:

1. a recursive block (the pre-recursive block);
2. a simultaneous block;
3. another recursive block (the post-recursive block);

As said, the pre-recursive and the post-recursive blocks are lower triangular. Therefore the corresponding equations are solvable with a cascade substitution with no iteration. Just the simultaneous equations set needs an iterative algorithm to be solved. It is important to say that the convergence criterion may also be applied to these variables only: when the feedback variables converge, the rest of the simultaneous variables also do.

bimets builds and analyzes the incidence matrix of the model, and then it orders the equations in pre-recursive, simultaneous and post-recursive blocks. The simultaneous block is then analyzed in order to find a minimal set of feedback variables. This last problem is known to be NP-complete[2].

The optimal reordering of the model equations is programmatically achieved through the use of an iterative algorithm applied to the incidence matrix that can produce 4 ordered lists of endogenous variables:

1. `vpre` is the ordered list containing the names of the endogenous pre-recursive variables to be sequentially computed (using their `EQ>` definition in the `MDL`) before the simulation iterative algorithm takes place;
2. `vsim` is the ordered list containing the names of the endogenous variables to be sequentially computed during each iteration of the simulation iterative algorithm;
3. `vfeed` is the list containing the names of the endogenous feedback variables;
4. `vpost` is the ordered list containing the names of the endogenous post-recursive variables to be sequentially computed once the simulation iterative algorithm has converged;

If equations are reordered, the previous conceptual scheme is modified as follow:

- initialize the solution for the current simulation period;
- compute the pre-recursive equations (i.e. the equation of the endogenous variables in the `vpre` ordered list);
- iteratively compute the system of simultaneous equations (i.e. the equation of the endogenous variables in the `vsim` ordered list); for each iteration update the values of the current endogenous variables and verify that the convergence criterion is satisfied on the feedback variables or that the maximum number of iterations has been reached;
- compute the post-recursive equations (i.e. the equation of the endogenous variables in the `vpost` ordered list);
- save the solutions;

Given $x_j, j = 1..m$ the exogenous variables and $y_{i,k}, i = 1..n$ the value of the i-endogenous variable in the simultaneous block at the iteration k, with i the position of the equation in a reordered model, the modified Gauss-Seidel method simply takes for the approximation of

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

the endogenous variable $y_{i,k}$ the solution of the following:

$y_{i,k} = f_i(x_1, ..., x_m, y_{1,k}, ..., y_{i-1,k}, y_{i,k-1}, ..., y_{n,k-1})$

As said, the convergence is then tested at the end of each iteration on the feedback variables.

Newton'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-Siebel method is small or negative. Moreover the Gauss-Siebel 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.

The simulation of a non-trivial model, if computed by using the same data but on different hardware, software or numerical libraries, produces numerical differences. Therefore a convergence criterion smaller than $10^{-7}\%$ frequently leads to a local solution.

See *Numerical methods for simulation and optimal control of large-scale macroeconomic models - Gabay, Nepomiastchy, Rachidi, Ravelli - 1980* for further information.

Below is an example of advanced simulation:

```
R> #STATIC SIMULATION EXAMPLE WITH EXOGENIZATION AND CONSTANT ADJUSTMENTS
R>
R> #define exogenization list
R> #'cn' exogenized in 1923-1925
R> #'i' exogenized in the 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
                      ,simType='STATIC'
                      ,TSRANGE=c(1923,1,1941,1)
                      ,simConvergence=0.00001
                      ,simIterLimit=100
                      ,Exogenize=exogenizeList
                      ,ConstantAdjustment=constantAdjList
                      )
```

3.6. Multipliers Analysis

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

The `MULTMATRIX()` procedure is articulated as follows:

1. simultaneous simulations are done;
2. the first simulation establishes the base line solution (without shocks);
3. the other simulations are done with shocks applied to each of the `INSTRUMENT` one at a time for every period in `TSRANGE`;
4. each simulation follows the defaults described in the "Simulation" section, but has to be `STATIC` for the IMPACT multipliers and `DYNAMIC` for INTERIM multipliers;
5. given the `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 variable 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 variable 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 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 non-Wage Spending "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')
                           )

Multipliter 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 non-Wage Spending "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 of 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'))

Multipliter Matrix:     50.00%
Multipliter 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
```

3.7. Renormalization

The renormalization[3] of econometric models consists of solving the model while interchanging the role of one or more endogenous variables with an equal number of exogenous variables.

The **bimets** `RENORM()` procedure determines the values for the `INSTRUMENT` exogenous variables which allow the objective `TARGET` endogenous variables to be achieved, with respect to the constraints given by the model `MDL` definition.

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 renormalization 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 related to the provided endogenous variable will be used as the instrument exogenous variable. This procedure is particularly suited for the automatic computation of the add-factors needed to fine tune the model into a baseline path and to improve the forecasting accuracy.

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

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

Back to our Klein's model example, we can perform the renormalization 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 renormalization by using the "w2" (Wage Bill of the Government Sector) and the "g" (Government non-Wage Spending) 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 into the model:

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

     DATE, PER, 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 non-Wage Spending) 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, PER, cn             , y

       1940, 1   ,  66.01116      ,  77.01772
       1941, 1   ,  78.02538      ,  98.04121
```

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

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