Introduction to Package `isismdl`

Rob van Harrevelt

2020-11-10

Contents

1 Introduction

This introduction shows how package `isismdl` can be used to solve a simple example model, a dynamical version of the Keynesian IS-LM model. This vignette will not cover all possibilities of package `isismdl`, but introduces the main functionality. Consult the *Reference Manual* for the full documentation.

Endogenous variables		Exogenous variables	
Y	national income	G	government spending
Y^d	dispensable income	M^s	money supply
C	consumption		
I	investments		
T	tax		
M^d	money demand		
r	interest rate		

Table 1: Variables of the IS-LM example model

Table 1 present all variables of the example model. The model equations are given by

$$C_t = \gamma_0 + \gamma_1(0.9Y_t^d + 0.1Y_{t-1}^d) - \gamma_2 r_t + \gamma_3 r_t^2 + \epsilon_{Ct} \tag{1}$$
$$I_t = \iota_0 + \iota_1(0.2Y_t + 0.8Y_{t-1}) - \iota_2 r_{t-1} + \iota_3 r_{t-1}^2 + \epsilon_{It} \tag{2}$$
$$M_t^d = \mu_0 + \mu_1 Y_t - \mu_2 r_t + \mu_3 r_t^2 + \epsilon_{M^d t} \tag{3}$$
$$T_t = \tau_0 + \tau_1 Y_t + \epsilon_{Tt} \tag{4}$$
$$Y_t^d = Y_t - T_t \tag{5}$$
$$Y_t = C_t + I_t + G_t \tag{6}$$
$$M_t^d = M_t^s, \tag{7}$$

where t is the period, for example a quarter for a quarterly model, and γ_i, ι_i, μ_i, τ_i are parameters. Equations 1-4 are *behavioural equations* and contain a residual ϵ_{it} (where i stands for C, I, M^d or T). Equations 4-7 are *identity equations* and have no residuals. Identity equations are typically definition or bookkeeping equations.

2 The model file

The model file for package `isismdl` should be defined on an external ASCII file. A detailed description of the model syntax is provided in the vignette *"Model Syntax Reference for Package isismdl"*.

The equations in the model file should be normalised, which implies that there must be one equation of the form $y_i = \ldots$ for each endogenous variable y_i. The system of equations 1-7 is not normalised: there are two equations for M^d and no equation for r. Therefore equation 7 has to be rewritten and transformed to an equation for r:

$$\begin{aligned} M^d &= M^s \\ 0 &= M^s - M^d \\ 0 &= \frac{M^s - M^d}{M^s} \\ r &= r + \frac{M^s - M^d}{M^s} \qquad \text{add } r \text{ at both sides} \end{aligned} \tag{8}$$

The effect of Equation 8 is that the value of the interest rate r is determined by the equilibrium between money demand (M^d) and supply (M^s).

The model file for this example model should include the equations 1 - 6 and equation 8. A model file for the IS-LM model is included in the example models in directory `models` of the package directory. To copy this file to your working directory, use

```
> mdl_file <- system.file("models", "islm.mdl", package = "isismdl")
> file.copy(mdl_file, "islm.mdl")
```

The model file `islm.mdl` has the following contents:

```
param c0 100 c1 0.7  c2 20 c3 0.5;
param i0 100 i1 0.2  i2 40 i3 1.5;
param m0  75 m1 0.23 m2 35 m3 1.5;
param t0 -25 t1 0.22;

? behavourial equations
frml c = c0 + c1 * (0.9 * yd + 0.1 * yd[-1]) - c2 * r + c3 * r**2;
frml i = i0 + i1 * (0.2 * y + 0.8 * y[-1]) - i2 * r[-1] + i3 * r[-1]**2;
frml md = m0 + m1 * y - m2 * r + m3 * r**2;
frml t = t0 + t1 * y;

? identity equations
ident yd = y - t;
ident y = c + i + g;
ident r = r + (ms - md) / ms;
```

The model definition contains a number of parameters, four behavioural equation (equations starting with the `frml` keyword) and three identity equations (equations starting with a `ident` keyword). As explained in Section 1, the behavioural equations have additive residuals. These residuals are implicit and not explicitly indicated in the model equations defined in the model file. Behavioural equations are also called *frml equations*[1] , and the residuals are also referred to as *constant adjustments*. The `frml` equations can be fixed (see Section 7) and play an important role in the fit procedure (see Section 8).

For identity equations, the left hand side is determined exactly by the right hand side expression. The `ident` keyword for the identity equations is optional. If an equation starts without the `ident` or `frml` keyword, the equation is an identity equation.

3 Creating an `IsisMdl` object

The function `isis_mdl` parses the model file and creates an `IsisMdl` object

```
> mdl <- isis_mdl("islm.mdl")
```

```
Isis Model Compiler 3.00
Compiling model ...
       7 equations processed
Ordering equations ...
Checking redundant feedback variables
       0 redundant feedback variables detected
Generating feedback variable ordering
Writing MIF file ...
Writing cross-reference file ...
End compilation
Reading mif file ...
Model with     7 equations read
Checking Model-code ...
Model is ok ...
Feedback ordering generated ...
```

[1] "frml" is short for "formula".

This function generates the file `islm.mrf`, the so called model reference file (mrf file), a text file with information about the structure of the model. If the compiler detects an error in the model file, an additional file `islm.err` is created. This file contains a list of errors.

The `mrf` file for our ISLM looks like this:

```
ISIS model compilation : islm 2020-11-10  09:22:11

Model reference table

Each variable name is followed by its
maximum lag (with minus sign) and maximum lead in the model,
its type:  E(xogenous), B(ehavioral), I(dentity).
Exogenous variables listed first.

Parameters are listed separately.
Each parameter name is followed by its length.

Equations are listed in solution order,
followed by a list of feedback variables.

*** Statistics ***

      9 variables of which
             2 exogenous
             4 behavioral
             3 identity

      3 total number of lags and leads with
             1 maximum lag
             0 maximum lead
             0 variables with leads

     14 parameters
            14 total length of parameter values

      7 equations of which
             0 in prologue
             7 in simultaneous block
             0 in epilogue

      2 feedback variables
                4 (****%) structural non zero's in jacobian


Statistics of feedback ordering
               7 words of memory used
             2   minimum feedback chain (excl. fb equations)
             5   maximum feedback chain (excl. fb equations)
             3.5 average feedback chain (excl. fb equations)
             2   average newton steps (rounded up)


*** Variables ***
```

```
*** Exogenous ***

g       0   0  E
ms      0   0  E

*** Endogenous ***

c       0   0  B
i       0   0  B
md      0   0  B
r      -1   0  I
t       0   0  B
y      -1   0  I
yd     -1   0  I

*** Parameters ***

c0      1
c1      1
c2      1
c3      1
i0      1
i1      1
i2      1
i3      1
m0      1
m1      1
m2      1
m3      1
t0      1
t1      1

*** Equations (in solution order) ***

    0 Prologue equations

    7 Simultaneous equations

i    md   t    yd   c    y    r

    0 Epilogue equations


*** Feedback variables ***

    2 Feedback variables

r    y


*** Additional information for feedback variables
    Name of feedback variable followed by how it was chosen: fixed, diagonal, heuristic,
    and the length of the feedback cycle (excluding feedback variables)
```

```
r    diagonal          2
y    diagonal          5
```

The mrf file contains technical information about the model. For example, it shows which variables are exogenous or endogenous, and which variables are feedback variables. A brief explanation of these concepts:

- Exogenous variables are variables that only occur at the right hand side of the equation.
- Endogenous variables are variables on the left hand side of the equations.
- A feedback variable is a variable that directly or indirectly depends on itself. In order to solve the model, initial guess values for the feedback variables have to be specified.

4 IsisMdl objects

An `IsisMdl` object is an R6 class object. R6 classes behave quite differently than the more familiar S3 and S4 classes. For example, for R6 classes, methods are part of the object itself and not of generic functions. R6 classes behave in a similar way as classes in object oriented languages such as Java, C++ or Python.

For example, the method `get_param()` can be used to obtain the values of the model parameters. To obtain the values of parameters `m0` and `c1`, use

```
> mdl$get_param(names = c("m0", "c1"))

$m0
[1] 75

$c1
[1] 0.7
```

Methods starting with `get_`, are often called "getter" methods. There are usually corresponding "setter" methods. For example

```
> mdl$set_param(c(m0 = 100))
> mdl$get_param("m0")

$m0
[1] 100
```

Input for method `set_param()` is a named vector.

There are more getters and setters, for example for model variables
(`get_data()` and `set_data()`) and constant adjustments (`get_ca()` and `set_ca()`). All getters and setters are described in the *Reference Manual*.

4.1 Copying IsisMdl objects

Consider the following assignment:

```
> mdl2 <- mdl
```

Variables `mdl2` and `mdl` refer to the same object. If you modify `mdl2`, then also `mdl` is modified.

```
> mdl2$set_param(list(m0 = 75))
> mdl$get_param("m0")

$m0
[1] 75
```

The usual copy-on-modify semantics that are used for conventional R objects such as S3 or S4 classes do not apply to R6 classes.

If you want to create a copy of the model, use the `copy()` method:

```
> mdl2 <- mdl$copy()
> mdl2$set_param(list(m0 = -9999))
> mdl$get_param("m0")
```

```
$m0
[1] 75
```

4.2 Writing and reading `IsisMdl` objects

To write an `IsisMdl`object to a file, use for example

```
> mdl$write_mdl(file = "islm_mdl.ismdl")
```

This methods serialises the model equations, parameters, model data, solve options, etc. Also included are fix values (discussed in Section 7), fit targets and root mean square errors (discussed in Section 8).

You can read this model back with the command

```
> mdl2 <- read_mdl("islm_mdl.ismdl")
```

`mdl2` is now an identical copy of `mdl`.

Do not use the standard functions for writing objects to a file and to restore them, such as `save`, `saveRDS`, `load` and `readRDS`.

5 Solving the model

Suppose that we want to solve the model for the period from `2020Q1` to `2021Q2`. The first step is to set the model period, the period for which the model will be solved. For this we use method `set_period()`:

```
> mdl$set_period("2020Q1/2021Q2")
```

The argument of `set_period` is a character representing a period range using the syntax of package `regts`.

Let's print the model object:

```
> mdl
```

```
IsisMdl object
Model index:                                    1
Number of variables:                            9
Maximum lag:                                    1
Maximum lead:                                   0
Model period:                                   2020Q1/2021Q2
Model data period:                              2019Q4/2021Q2
```

The model data period is the model period extended with the lag period (and lead period for models with leads). You can retrieve the model period and model data period with the methods `get_period()` and `get_data_period`.

To solve the model, we need

- Values for the exogenous variables `g` and `ms` for the model period `2020Q1/2021Q2`
- Values for the lagged variables `r`, `y` and `yd` in the lag quarter `2019q4`

- Initial values for the feedback variables `r` and `y` in the model period. The solution does not depend on the initial values, but it is easier to find the solution of the model if you provide a reasonable guess for the initial values.

In typical applications, the input timeseries are read from an external file, for example a csv or Excel file. For this simple example, we will create a timeseries object by employing function `regts` of the `regts` package. For the exogenous variables `g` and `ms`, we assume an annual growth of 1.5% after `2019Q4`.

```
> # exogenous variables:
> g  <- regts(210 * cumprod(rep(1.015, 6)), start = "2020Q1")
> ms <- regts(200 * cumprod(rep(1.015, 6)), start = "2020Q1")
> # feedback variables (with lag):
> r <- regts(3.4, period = "2019Q4/2021Q2")
> y <- regts(980, period = "2019Q4/2021Q2")
> # lagged variable
> yd <- regts(790, start = "2019Q4")
> data <- cbind(g, ms, r, y, yd)
> data
```

```
              g       ms   r   y  yd
2019Q4       NA       NA 3.4 980 790
2020Q1 213.1500 203.0000 3.4 980  NA
2020Q2 216.3472 206.0450 3.4 980  NA
2020Q3 219.5925 209.1357 3.4 980  NA
2020Q4 222.8863 212.2727 3.4 980  NA
2021Q1 226.2296 215.4568 3.4 980  NA
2021Q2 229.6231 218.6887 3.4 980  NA
```

To transfer these variables to the model, use method `set_data()`:

```
> mdl$set_data(data)
```

Let's check the model data:

```
> mdl$get_data()
```

```
        c        g  i md       ms   r  t   y  yd
2019Q4 NA       NA NA NA       NA 3.4 NA 980 790
2020Q1 NA 213.1500 NA NA 203.0000 3.4 NA 980  NA
2020Q2 NA 216.3472 NA NA 206.0450 3.4 NA 980  NA
2020Q3 NA 219.5925 NA NA 209.1357 3.4 NA 980  NA
2020Q4 NA 222.8863 NA NA 212.2727 3.4 NA 980  NA
2021Q1 NA 226.2296 NA NA 215.4568 3.4 NA 980  NA
2021Q2 NA 229.6231 NA NA 218.6887 3.4 NA 980  NA
```

Model variables for which we have not yet specified values (`c`, `i`, and `t`) all have value `NA`. Values for these variables are computed when the model is solved. We are now ready to solve the model:

```
> mdl$solve()
```

```
Model Solve Options
Solution period               2020Q1/2021Q2
Simulation mode               dynamic
Feedback starting values      current period
Maximum iterations per period 50
Relaxation minimum            0.500E-01
           maximum            1.00
           shrinkage          0.500
```

```
Criteria stepback             1.30
          matrix              0.900
Maximum updates Newton matrix per period                 10
Maximum number of line searches with old Jacobian        5
Criterion for line search decisions etc.                 geometric

   1 steps for Newton matrix at iteration   0 (1/condscal =  1.43E-01)
Convergence for 2020Q1 in    4 iterations
Convergence for 2020Q2 in    5 iterations
Convergence for 2020Q3 in    4 iterations
Convergence for 2020Q4 in    4 iterations
Convergence for 2021Q1 in    4 iterations
Convergence for 2021Q2 in    4 iterations
Total number of iterations   26
Solve model used     0.00 CPU secs
```

The output of `solve` starts with an overview of the solve options. Subsequently, the number of iterations per each period is printed. The solve options can be specified with method `set_solve_options` (see the *Reference Manual* for more information).

The method `get_data()` can be used to retrieve the solution.

```
> mdl$get_data()
```

```
              c        g        i       md       ms        r        t         y
2019Q4       NA       NA       NA       NA       NA 3.400000       NA  980.0000
2020Q1 595.2722 213.1500 177.5801 203.0000 203.0000 3.284697 191.9205  986.0023
2020Q2 602.9867 216.3472 182.6351 206.0450 206.0450 3.309682 195.4332 1001.9690
2020Q3 610.2563 219.5925 184.9507 209.1357 209.1357 3.304113 198.2559 1014.7995
2020Q4 617.5631 222.8863 187.7053 212.2727 212.2727 3.301510 201.1941 1028.1548
2021Q1 624.9866 226.2296 190.4614 215.4568 215.4568 3.298568 204.1691 1041.6776
2021Q2 632.5206 229.6231 193.2628 218.6887 218.6887 3.295613 207.1894 1055.4065
             yd
2019Q4 790.0000
2020Q1 794.0818
2020Q2 806.5358
2020Q3 816.5436
2020Q4 826.9607
2021Q1 837.5085
2021Q2 848.2171
```

A detailed description of the numerical methods used to solve the model is provided in Chapters 17 to 20 in the Isis Gebruikershandleiding (Afdeling Informatietechnologie en Onderzoeksondersteuning 2015) (in Dutch, there is currently no English documentation available). Briefly, the equations are ordered and a preferably small set of feedback variables is determined (Don and Gallo 1987; Hasselman 2004). The condensed system of equations involving feedback variables only is solved using the Broyden method (Dennis and Schnabel 1996), a variant of the Newton method with an update method for the jacobian at each iteration.

6 Constant adjustments

As explained in Section 2, each behavioural or `frml` equation has a residual, also called constant adjustment. By default, all constant adjustments are zero. It is possible to set constant adjustments with `IsisMdl` method `set_ca` or `set_ca_values`. Here I illustrate the usage of method `set_ca_values`:

```
> mdl$set_ca_values(20, names = "c", period = "2020q1")
> mdl$get_ca(period = "2020q1/2020q2")
```

```
        c i md t
2020Q1 20 0  0 0
2020Q2  0 0  0 0
```

7 Fixing variables

The left hand side variables of `frml` equations can be fixed, which means that the left hand side variable is exogenous while the constant adjustment is endogenous. The constant adjustment is calculated as the difference between the left hand side variable and the right hand side expression. Equations can be fixed for the whole period but also for specific periods.

For example, suppose that we have some observations for `c` and `t`:

```
> c  <- regts(c(600, 610), period = "2020q1/2020q2")
> t <- regts(200, start = "2020q1")
> obs <- cbind(c, t)
> obs
```

```
         c   t
2020Q1 600 200
2020Q2 610  NA
```

We can fix `c` and `i` at these given values:

```
> mdl$set_fix(obs)
> mdl$solve(options = list(report = "none"))  # solve but suppress output
> mdl$get_data(names = c("c", "t"), period = 2020)
```

```
              c        t
2020Q1 600.0000 200.0000
2020Q2 610.0000 196.9097
2020Q3 610.3983 198.1100
2020Q4 617.5490 201.2084
```

`c` is fixed at the specified value for `2020Q1` and `2020Q2`, but endogenous for other periods. `t` is only fixed at `2020Q1`: if a fix value is `NA`, then the corresponding variable is not fixed at the corresponding period. Note that if we specify a year for argument `period`, `get_data` returns the data for each quarter in that year.

The constant adjustments for `c` and `t` are now nonzero:

```
> mdl$get_ca(period = 2020)
```

```
              c i md        t
2020Q1 7.469207 0  0 6.996042
2020Q2 4.965774 0  0 0.000000
2020Q3 0.000000 0  0 0.000000
2020Q4 0.000000 0  0 0.000000
```

Method `clear_fix()` removes all fix values:

```
> mdl$clear_fix()
```

Now all `frml` variables are free (i.e. endogenous), and the constant adjustments are exogenous. However, `clear_fix` does *not* set all constant adjustments to zero. All constant adjustments retain their values.

To remove only specific fix values, use for example:

```
> mdl$set_fix_values(NA, "t")  # remove all fix values for t
```

Table 2 presents an overview of the methods related to fixing. Full documentation with examples is provided in the *Reference Manual*.

`set_fix`, `set_fix_values` and `get_fix`	set and retrieve fix values. See also Section 10.2.
`clear_fix`	remove all fix values and unfix all frml variabkles
`fix_variables`	fix some variables for a specified period at the current model data values

Table 2: Overview of methods related to fixing.

8 The fit procedure

8.1 Introduction

Suppose that we have some observations for `y`. Since `y` is determined by an identity equation, we cannot fix `y`. To force the model forecast to reproduce the observed values for `y`, we can use the so called "fit procedure". The basic idea of the fit procedure is as follows. Since `y` depends indirectly on the constant adjustments of `c`, `i`, `t` and `md`, we can try to find the values of these constant adjustments so that the model reproduces the observed values of `y`.

The name "fit procedure" may be a bit confusing. Usually, the term "fitting" refers to the process of finding the parameters of a mathematical function that best describe a series of data points. The number of observations is always larger than the number of parameters to be fitted. However, in the fit procedure, we have more degrees of freedom than observations. For the example discussed above, we have one observation for `y` for each period, but four degrees of freedom (the four constant adjustments for that period). In Dutch, the fit procedure is called "waarnemingenprocedure", for which the literal English translation is "observation procedure". This would be a better name for "fit procedure", but the usage of the term "fit procedure" is already well established.

The method of the fit procedure is described in Ref. (Sandee, Don, and Berg 1984). The method is based on the maximum likelihood principle: find the values of the constant adjustments for which the probability has a maximum subject to the constraint that the model reproduces the observations. The constant adjustments (residuals), ϵ_{it} in Equations 1 - 4, are assumed to be distributed according to a normal distribution with mean zero and standard deviation σ_i. This implies that the fit procedure minimises $\sum_i (\epsilon_{it}/\sigma_i)^2$ subject to the constraint that the model reproduces the observations.

8.2 Numerical implementation

The numerical implementation of the fit procedure is described in CPB Memorandum Hasselman (2003), which is included as vignette *"A Newton algorithm for solving an underdetermined system of equations"* in this package. Here a brief summary is presented that will be sufficient to understand the output of the fit procedure discussed below.

First, define $w \in \mathbb{R}^m$ as the vector of observations at a specific period (in this section the period index t is omitted), and the $u \in \mathbb{R}^n$ as the vector of residuals scaled with the standard deviation (i.e. $u_i = \epsilon_i/\sigma_i$). The u_is are also called *fit instruments*. The number of observations is less than or equal to the number of fit instruments ($m \leq n$). The fit procedure can be formulated mathematically as

$$\min_{u} \quad u^T u \tag{9}$$

$$\text{subject to} \quad w = h(u) \tag{10}$$

The vector valued function $h : \mathbb{R}^n \to \mathbb{R}^m$ represents the implicit reduced form equations relating the targeted endogenous variables to the residuals. The function is evaluated by solving the model for a given set of fit instruments.

A Newton-like method is used to solve Equations 9 - 10. This method involves the fit jacobian $D_{ij} = \partial h_i / \partial u_j$, which is calculated numerically.

The fit procedure for $m = n$ is often referred to as "square fit procedure", since the fit jacobian is a square matrix in this case. For the square fit procedure, the results are independent of the standard deviations σ_i, since there is a unique solution of Equation 10. However, because of the numerical implementation, σ_i should have the same scale as the corresponding variable. So simply setting all σ_is to 1 is not always a good idea.

8.3 Example

Suppose that we have some observations for variables `c` and `r` that we want to use as fit targets for the fit procedure

```
> y <- regts(c(1000, 1005), period = "2020q1/2020q2")
> r <- regts(3.5, start = "2020q1")
> fit_targets <- cbind(y, r)
> fit_targets
```

```
          y   r
2020Q1 1000 3.5
2020Q2 1005  NA
```

We register them as fit targets with function `set_fit`

```
> mdl$set_fit(fit_targets)
```

Method `set_fit` works similarly as `set_fix`: if a fit target is `NA`, then the corresponding variable is not a fit target.

Before solving the model, the standard deviations of the residuals, also called root mean square errors (rms errors), have to be specified. We assume that the rms values are given[2].

```
> rms_errors <- c(c = 5.0, i = 21, md = 2, t = 2)
> mdl$set_rms(rms_errors)
```

Function `set_rms` expects a named numeric vector as argument. The default values for the rms errors are zero. If the rms error of a constant adjustment is zero, then the constant adjustment is not used as a fit instrument and retains the original value when the model is solved.

We are now ready to solve with the fit procedure:

```
> mdl$solve()
```

```
Model Solve Options
Solution period               2020Q1/2021Q2
Simulation mode               dynamic
```

[2]The rms values can be estimated from the model if for some historical period the values of the frml variables are known. Solve the model while fixing the frml variables. The rms errors can now be estimated from the calculated constant adjustments.

```
Feedback starting values      current period
Maximum iterations per period 50
Relaxation minimum            0.500E-01
           maximum            1.00
           shrinkage          0.500
Criteria stepback             1.30
          matrix              0.900
Maximum updates Newton matrix per period                10
Maximum number of line searches with old Jacobian       5
Criterion for line search decisions etc.                geometric


     4 CAs used by Fit:
     c  i  md t
Convergence for 2020Q1 in    0 iterations
 Fiter     Icond       Delwmx       Delsmx Deltyp        Ratio Type Name
     0               0.04861              w                    Rel  r
     1D 1.02E-01      0.00038      0.05591 w           0.00788 Rel  r
     2D 1.02E-01  2.22083E-07      0.00038 s           0.00684 Rel  r
     3D 1.02E-01  1.26303E-10  2.21957E-07 s           0.00058 Rel  r
Fit convergence in 2020Q1       after    3 iterations and    3 jacobians
Total number of iterations for accurate calculation fit Jacobian     37
Average number of iterations per column    3

   1 steps for Newton matrix at iteration   0 (1/condscal =  1.41E-01)
Convergence for 2020Q2 in    4 iterations
 Fiter     Icond       Delwmx       Delsmx Deltyp        Ratio Type Name
     0               0.00053              w                    Rel  y
     1D 1.00E+00  1.40972E-06      0.02800 s                    Rel  i
     2D 1.00E+00  3.11730E-10  4.84362E-06 s           0.00017 Rel  i
     3D 1.00E+00  3.11730E-10  1.07990E-09 s           0.00022 Rel  i
Fit convergence in 2020Q2       after    3 iterations and    3 jacobians
Total number of iterations for accurate calculation fit Jacobian     36
Average number of iterations per column    3

Convergence for 2020Q3 in    4 iterations
Convergence for 2020Q4 in    3 iterations
Convergence for 2021Q1 in    3 iterations
Convergence for 2021Q2 in    2 iterations
Total number of iterations  105
Solve model used     0.00 CPU secs
```

The output includes an iteration report of the fit procedure for each period with fit targets. The iteration report start with the line that starts with `Fiter`. A brief description of the columns in the iteration report is presented in table 3.

The results of the fit procedure are:

```
> mdl$get_data(names = c("y", "r"), period = 2020)
```

```
             y        r
2020Q1 1000.000 3.500000
2020Q2 1005.000 3.336329
2020Q3 1014.562 3.301934
2020Q4 1028.178 3.301724
```

Column	Description
`Fiter`	Iteration counter. Suffixed with `D` if a new fit jacobian is calculated at the beginning of the iteration.
`Icond`	Inverse condition of the fit jacobian. A very small number (smaller than about 10^{-7}) indicates that the jacobian is nearly singular.
`Delwmx`	Maximum relative deviation of observation w_i from the corresponding endogenous variables y_i, defined as $\mid w_i - y_i \mid / \max(\mid w_i \mid, 1)$.
`Delsmx`	Maximum relative step size, defined as $\mid y_i - y_i^{\text{prev}} \mid / \max(\mid y_i^{\text{prev}} \mid, 1)$, where y_i^{prev} is the value of the endogenous variable for the previous iteration
`Deltyp`	Type of the difference (`Delwmx` or `Delsmx`) for which the next three columns show further information: `w` for `Delwmx` and `s` for `Delsmx`. Convergence of the fit procedure is achieved if both `Delwmx` or `Delsmx` are smaller than the convergence threshold, which is by default 100 times the square root of the machine precision ($\approx 1.5 \times 10^{-6}$) and can be set by specifying fit option `cvgabs`. `Deltyp` is `w` if `Delwmx` is larger than the convergence threshold and otherwise `s`.
`Ratio`	The ratio of the current and previous value of `Delwmx` or `Delsmx`, depending on `Deltyp`.
`Type`	Specifies whether `Delwmx` or `Delsmx`, depending on `Deltyp`, is an absolute or relative difference.
`Name`	The name of the variable with the largest discrepancy between the observation and value, or the name of the variable with the largest (relative) step size, depending on `Deltyp`.

Table 3: Description of the columns of the iteration report of the fit procedure

```
> mdl$get_ca(period = 2020)
```

```
                c        i           md           t
2020Q1 0.5349534 9.566257  2.12500001 -0.05363441
2020Q2 0.2822417 4.977867 -0.03011695 -0.02845191
2020Q3 0.0000000 0.000000  0.00000000  0.00000000
2020Q4 0.0000000 0.000000  0.00000000  0.00000000
```

In the previous example, the two fit targets were the left hand side variables of identity equations, but frml variables can be used as fit targets as well. For example

```
> fit_targets <- cbind(c, t, y, r)
> mdl$set_fit(fit_targets)
```

It is also possible to combine the fit procedure with fixing:

```
> mdl$set_fit_values(NA) # remove all fit targets
> fit_targets <- cbind(y, r)
> fix_data <- cbind(c, t)
> mdl$set_fit(fit_targets)
> mdl$set_fix(fix_data)
```

In general, it is preferable to also use the fit procedure for the observations of frml variables, because the solution of the fit procedure has a higher probability, assuming that we have good estimates of the standard deviations.

As shown in the example above, the expression `mdl$set_fit_values(NA)` can be used to remove all fit targets. This can also be achieved with method `clear_fit`. However, `clear_fit` not only removes the fit targets, but also sets all rms errors to zero.

```
> mdl$clear_fix() # remove all fit targets and set all rms errors to 0
```

8.4 Options for the fit procedure

There are many options that can be specified to fine-tune the fit procedure, for example to fix numerical problems or to get more detailed information about the iteration process or the fit jacobian. If the fit jacobian is (nearly) singular, it can be useful to turn on an SVD (Singular Value Documentation) analysis of the jacobian. Consult the documentation of method `set_fit_options` in the *Reference Manual* for more information. In interactive R-sessions, you can also get the documentation by typing

```
> ?set_fit_options
```

8.5 Overview of methods for to the fit procedure

Table 4 presents an overview of the methods related to the fit procedure. Full documentation with examples is provided in the *Reference Manual*.

`set_fit`, `set_fit_values` and `get_fit`	set and retrieve fit targets. See also Section 10.2.
`set_rms` and `get_rms`	set and get rms errors
`clear_fit`	remove all fit targets and set all rms errors to 0.
`set_fit_options` and `get_fit_options`	set and get the options for the fit procedure.

Table 4: Overview of methods related to the fit procedure.

9 Deactivating equations

It is possible to completely deactivate an equation. For example, suppose we no longer wish to calculate the interest rate by the model. We can use:

```
> mdl$set_eq_status("inactive", names = "r")
```

Now `r` is an exogenous variable for the whole solution period. If a frml equation is deactivated, then the constant adjustment is also exogenous and retains the original value.

After (de)activating an equation, it is often desirable, although not necessary, to reorder the model equations and determine a new feedback set. This is possible with method `order`:

```
> mdl$order()

Summary ordering information
           0 prologue equations
           5 simultaneous equations
           2 epilogue equations
No feedback ordering determined
```

10 Modifying model timeseries and parameters

10.1 Model data

There are three methods that can be used to modify model variables: `set_data`, that we already encountered in Section 5, `set_values`, and `change_data`. A number of examples is presented below.

set_data

Method `set_data` updates the model data with a timeseries object. If `data` is a timeseries object, we can use

```
> mdl$set_data(data)
```

`set_data` uses the column names of `data` to determine which model variables should be updated.

set_values

Method `set_values` is useful to set the value of a model variable for a specific period, without creating a timeseries object. Example:

```
> mdl$set_values(1000, names = "y", period = "2019q4")
```

Method `set_data` can also be used to give more than one variable the same value. Some examples:

```
>  # give all variables starting with m (ms and md) the value 200 for all periods:
> mdl$set_values(200, pattern = "^m")
> # make all model variables equal to 1:
> mdl$set_values(1)
```

change_data

Suppose we wish to increase exogenous variable `g` with 10% in the period `"2020Q1/2020Q2"`. This is easy with method `change_data`:

```
> mdl$change_data(fun = function(x) {x * 1.1}, names = "g", period = "2020Q1/2020Q2")
```

Method `change_data` applies a function to the model timeseries.

10.2 Constant adjustments, fix values and fit targets

Since constant adjustments, fix values and fit targets are timeseries just as the model variables, similar methods are available to modify these model timeseries. Table 5 presents an overview of the available methods. The methods in the same column work similarly. The fourth column gives the getter methods that can be used to retrieve the model timeseries. Full documentation with examples is provided in the *Reference Manual*.

set_data	set_values	change_data	get_data
set_ca	set_ca_values	change_ca	get_ca
set_fix	set_fix_values		get_fix
set_fit	set_fit_values		get_fit

Table 5: Overview of methods to modify and retrieve model timeseries

10.3 Parameters and rms errors

Parameters can be modified with methods `set_param` and `set_param_values`. In Section 4.1, we have already seen an example of `set_param`. Another example:

```
> mdl$set_param(c(m0 = 75, c0 = 120))
```

The argument of `set_param` is named vector.

Method `set_param_values` is useful to give more than one parameter the same value. For example, to make all parameters for the quadratic terms equal to zero, use

```
> mdl$set_param_values(0, pattern = "3$")
```

The rms errors, i.e. the standard deviation of the residuals used in the fit procedure (see Section 8), are modified by similar methods `set_rms` and `set_rms_values`. Table 6 presents an overview of the available methods. The methods in the same column work similarly. The third column gives the getter methods that can be used to retrieve the parameters or rms errors. Full documentation with examples is provided in the *Reference Manual*.

`set_param`	`set_param_values`	`get_param`
`set_rms`	`set_rms_values`	`get_rms`

Table 6: Overview of methods to modify and retrieve model parameters and rms errors

References

Afdeling Informatietechnologie en Onderzoeksondersteuning. 2015. "Isis Gebruikershandleiding."

Dennis, J. E., Jr., and R. B. Schnabel. 1996. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. SIAM.

Don, F.J. Henk, and G. M. Gallo. 1987. "Solving Large Sparse Systems of Equations in Econometric Models." *Journal of Forecasting*, 167–80.

Hasselman, B. H. 2003. "A Newton Algorithm for Solving an Underdetermined System of Equations." *CPB Memorandum*.

———. 2004. "Detecting and Removing Redundant Feedback Vertices." *CPB Memorandum*.

Sandee, J., F. J. H. Don, and P. J. C. M. van den Berg. 1984. "Adjustment of Projections to Recent Observations." *European Economic Review* 26: 153–66.