Introduction to package dynmdl

Rob van Harrevelt

2017-10-19

Contents

1 Introduction **1**

2 The mod file **2**

3 The DynMdl class **2**

4 Cloning a DynMdl object **3**

5 Computing the steady state **4**

6 The model period **4**

7 Solving the model **5**

1 Introduction

In this tutorial I show how a simple model with rational expectations can be solved with package `dynmdl`. The example model is a version of the ISLM model with rational expectations. This model contains the following variables:

Endogenous variables		Exogenous variables	
Y	national income	G	government spending
Y^D	dispensible income	Ms	money supply
T	tax		
C	consumption		
I	investments		
M^D	money demand		

The model equations are given by

$$
\begin{aligned}
Y_t &= C_t + I_t + G_t \\
Y_t^D &= Y_t - T_t \\
T_t &= t_0 + t_1 Y_t \\
C_t &= c_0 + c_1 Y_{t-1}^D + c_2 Y_t^D + c_3 Y_{t+1}^D + c_4 r_t + c_5 r_t \\
I_t &= i_0 + i_1 Y_{t-1} + i_2 Y_t + i_3 Y_{t+1} + i_4 r_t + i_5 r_t^2 \\
M_t^d &= m_0 + m_1 Y_t + m_2 r_t + m_3 r_t^2 + m_3 r_t^2 \\
M_t^d &= M_t^s
\end{aligned}
$$

2 The mod file

`dynmdl` uses the same mod file as Dynare. However, only the `var`, `varexo`, `parameters`, `model` and `initval` blocks are used. The other blocks in the mod file are ignored.

The mod file for the ISLM has the following contents:

```
%Declaring variables
var y yd t c i md r;        % endogenous variables
varexo g ms;                % exogenous variables

%Setting parameter values
parameters c0 c1 c2 c3 c4 c5;
parameters i0 i1 i2 i3 i4 i5;
parameters m0 m1 m2 m3;
parameters t0 t1;

c0 = 100; c1 = 0.28; c2 = 0.32; c3 = 0.10; c4 = -20; c5 = 1;
i0 = 100; i1 = 0.12; i2 = 0.08; i3 = 0.04; i4 = -40; i5 = -1.5;
m0 = 80; m1 = 0.23; m2 = -35; m3 = -1.5;
t0 = -15; t1 = 0.22;

model;
y = c + i + g;
yd = y - t;
t = t0 + t1 * y;
c = c0 + c1 * yd(-1) + c2 * yd + c3 * yd(+1) + c4 * r + c5 * r^2;
i = i0 + i1 * y(-1) + i2 * y + i3 * y(+1) + i4 * r + i5 * r^2;
md = m0 + m1 * y + m2 * r + m3 * r^2;
md = ms;
end;

initval;
g  = 240; ms = 230; r  = 3.5; y  = 980; c  = 500; t  = 100;
md = ms; yd = y - t; i  = y - c - g;
end;
```

3 The `DynMdl` class

The function `dyn_mdl` creates a `DynMdl` object from the mod file.

```
> mdl <- dyn_mdl("islm.mod")
```

```
Starting Dynare (version 4.6-unstable).
Starting preprocessing of the model file ...
Found 7 equation(s).
Evaluating expressions...done
Computing static model derivatives:
 - order 1
Computing dynamic model derivatives:
 - order 1
 - order 2
Preprocessing completed.
```

```
> mdl
```

```
DynMdl object
Number of endogenous variables:                          7
Number of exogenous variables:                           2
Maximum endogenous lead:                                 1
Maximum endogenous lag:                                  1
Maximum exogenous lead:                                  0
Maximum exogenous lag:                                   0
Number of nonzeros dyn. jac:                             24
```

A `DynMdl` 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 simular way as classes in object oriented languages such as Java, C++ or Python.

For example, the method `get_params()` can be used to obtain the parameters of the model

```
> mdl$get_param()
```

```
    c0      c1      c2      c3      c4      c5      i0      i1      i2      i3
100.00    0.28    0.32    0.10  -20.00    1.00  100.00    0.12    0.08    0.04
    i4      i5      m0      m1      m2      m3      t0      t1
-40.00   -1.50   80.00    0.23  -35.00   -1.50  -15.00    0.22
```

Methods starting with `get_`, are ofter called a “getter” methods. There are also corresponding “setter” methods. For example

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

```
 m0
100
```

Input for function `set_params()` is a named numerical vector.

4 Cloning a `DynMdl` object

Consider the following assignment:

```
> mdl2 <- mdl
```

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

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

```
m0
75
```

The usual copy-on-modify semantics that are used for convential R objects, including S3 or S4 objects, do not apply to R6 classes.

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

```
> mdl2 <- mdl$clone()
> mdl2$set_param(c(m0 = -9999))
> mdl$get_param("m0")
```

```
m0
```

```
75
```

5 Computing the steady state

To compute the steady state of the model, use the method `solve_steady()`:

```
> mdl$solve_steady(control = list(trace = 1))
```

```
  Algorithm parameters
  --------------------
  Method: Newton  Global strategy: double dogleg (initial trust region = -2)
  Maximum stepsize = 1.79769e+308
  Scaling: fixed
  ftol = 1e-08 xtol = 1e-08 btol = 0.001 cndtol = 1e-12

  Iteration report
  ----------------
  Iter         Jac     Lambda      Eta     Dlt0     Dltn         Fnorm   Largest |f|
     0                                                    2.206061e+04  1.582500e+02
     1  N(6.4e-04) N             0.2717 360.2173 720.4346  6.143994e-02  2.242076e-01
     2  N(6.5e-04) N             0.7092   0.7251   1.4502  1.493885e-10  1.105563e-05
     3  N(6.5e-04) N             0.7092   0.0000   0.0001  1.979094e-26  1.136868e-13
```

Control parameter `trace` has been set to `TRUE` in order to get an iteration report. Method `solve_steady()` employs package `nleqslv` to solve the steady state problem.

The method `get_static_endos()` returns the computed static endogenous variables.

```
> mdl$get_static_endos()
```

```
          y          yd           t           c           i          md
1210.381827  959.097825  251.284002  718.831355  251.550472  230.000000
          r
   3.110669
```

The steady state computation requires

- Static values for the exogenous variables
- An initial guess for the static values of the endogenous variables

After compiling a model, these values are set to the corresponding values in the `initval` block of the mod file (or set to 0). You can change these values with `set_steady_exos()` or `set_steady_endos()`. Input for these function is a named numerical vector. For example:

```
> mdl$set_static_exos(c(g = 240))
```

6 The model period

The model period is the period for which the model will be solved. Function `set_period()` can be used to set the model period:

```
> mdl$set_period("2017Q1/2018Q2")
```

The model period should be distinguished from the data period, the period for which data is needed to solve the model. The data period includes the lag and lead period. To get the data period, use for example

```
> mdl$get_data_period()
```

```
[1] "2016Q4/2018Q3"
```

Because the example model has a maximum lag and lead of 1, the data period starts one period before the model period and ends one quarter after the end of the model period.

Method `set_period()` also initialises the endogenous and exogenous model variables for the full data period. They are initialised with the static endogenous and exogenous variables, respectively. The method `get_endo_data()` returns the values of the endogenous variables as a `regts` object.

```
> mdl$get_endo_data()
```

```
              y       yd       t        c        i  md        r
2016Q4 1210.382 959.0978 251.284 718.8314 251.5505 230 3.110669
2017Q1 1210.382 959.0978 251.284 718.8314 251.5505 230 3.110669
2017Q2 1210.382 959.0978 251.284 718.8314 251.5505 230 3.110669
2017Q3 1210.382 959.0978 251.284 718.8314 251.5505 230 3.110669
2017Q4 1210.382 959.0978 251.284 718.8314 251.5505 230 3.110669
2018Q1 1210.382 959.0978 251.284 718.8314 251.5505 230 3.110669
2018Q2 1210.382 959.0978 251.284 718.8314 251.5505 230 3.110669
2018Q3 1210.382 959.0978 251.284 718.8314 251.5505 230 3.110669
```

Similarly, method `get_exo_data()` can be uses to retrieve the values of the exogenous variables

7 Solving the model

Suppose that we know the values of the endogenous variables Y and Y^D in the lag period `2016Q4`. The corresponding model variables can be set as follows:

```
> mdl$set_endo_values(1200, names = "y", period = "2016Q4")
> mdl$set_endo_values(1000, names = "yd", period = "2016Q4")
```

Further suppose that in `2017Q1` we know the intended government spending:

```
> mdl$set_exo_values(280, names = "g", period = "2017Q1")
```

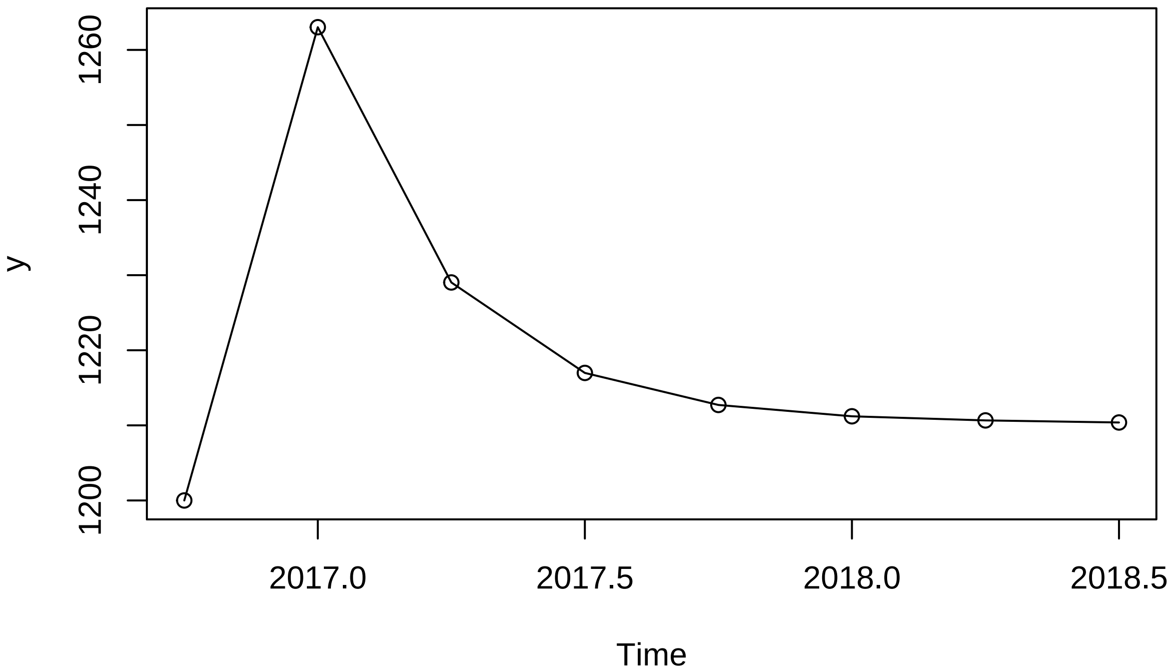
Now we can solve the model using the stacked time Newton method;

```
> mdl$solve(control = list(trace = TRUE))
```

```

Iteration report
----------------
 Iter          Jac        Largest |f|   Index largest |f|
    0                       4.000e+01                   1
    1     1.81e-02          1.113e-01                   5
    2     1.80e-02          4.956e-06                   6
    3     1.80e-02          2.274e-13                  15
Convergence after 3 iterations
```

```
> plot(mdl$get_endo_data(names = 'y'), type = "o")
```



The method `set_value()` is useful for simple experiments, but for more typical applications the exogenous shocks and values of the endogenous variables are stored in a csv file. For example, consider the following csv file

```
      , g,   y,    yd
2016Q4,    , 1200, 1000
2017Q1, 280, ,
```

To update the model workspace, first read the csv file and convert it to a `regts`

```
> df <- read.csv("input.csv")
> ts <- as.regts(df, time_column = 1)
> ts
```

```
         g    y   yd
2016Q4  NA 1200 1000
2017Q1 280   NA   NA
```

Then update the model data with

```
> mdl$set_data(ts, upd_mode = "updval")
```

In order to ignore the `NA` values in the input timeseries, the update mode `"updval"` is used.