# Tutorial on Dynare

Jay Rhee

## 1 Introduction to Dynare

In practice linear rational expectations models are solved numerically on the computer. Several Matlab programs are available freely on the web as many authors complement their papers with a program that implement their suggested solution method. Dynare, however, is a popular matlab-based program for solving models that allows the models to be written in a more natural form. Dynare is a suite of programs that solve linear rational expectations models.

### 1.1 Getting started

To run Dynare, one has to use Mathworks MATLAB.

- Download the latest Dynare Windows installer at www.dynare.org (which will be Dynare 4.3.x)
- To install Dynare, double-click on the Dynare installer and follow the instructions.
- Run MATLAB
- Configure MATLAB for Dynare (using MATLABs command window):
  - Type: addpath c:\dynare\4.3.x\matlab

  (You will have to do this every time you start MATLAB, as at least older versions of MATLAB will not remember the setting.)

Now MATLAB is prepared for the use of Dynare.

### 1.2 Running and Editing a Dynare *.mod-File

In order to give instructions to Dynare, the user has to write a model file whose filename extension has to be example.mod (instead of just .m). This file contains the description of the model and the computing tasks required by the user. Once the model file is written, Dynare is invoked using the dynare-command at the MATLAB prompt (with the filename of the .mod given as argument).

- Create a working directory that will hold your Dynare-files. For example, c:\dynare\work.
- After writing your example model file, save it to your working directory. (Note that dynare-files use the extension .mod instead of just .m)

- At the MATLAB command window type the following to change the working directory of MATLAB: cd c:\dynare\work
- Now type the following to run the example model file: dynare example (without any extension!)
- If you want to edit your .mod-file type the following: edit example.mod.

A Dynare model file contains a list of commands and of blocks. Each command and each element of a block is terminated by a semicolon (;). Blocks are terminated by end;. This terminations are mandatory in Dynare. To forget one or more of these terminations is one of the most common reasons for issuing error messages.

In the following, we discuss how to use dynare to solve the linear rational expectations model.

# 2 Solving/Simulating the RBC Model with Dynare - A Code Example

## 2.1 The linear model

Now consider the RBC model with labor supply. The linearized model is given by

$$y_t = \left(\frac{C^*}{Y^*}\right) c_t + \delta \left(\frac{K^*}{Y^*}\right) i_t$$

$$k_t = (1-\delta)k_{t-1} + \delta i_t$$

$$y_t = \alpha k_t + (1-\alpha) n_t + a_t$$

$$\eta n_t + \sigma c_t = y_t - n_t$$

$$c_t = \mathrm{E}_t c_{t+1} - \left(\frac{1}{\sigma}\right) \alpha \left(\frac{Y^*}{K^*}\right) (\mathrm{E}_t y_{t+1} - k_t)$$

$$a_t = \rho a_{t-1} + \varepsilon_t$$

where $c, y, k, n, i$ and $a$ denote consumption, output, capital, labor, investment, and technology, respectively. It is also assumed that $\varepsilon_t$ is $i.i.dN(0, sda^2)$. The steady-state values are give by

$$\frac{Y^*}{K^*} = \frac{1}{1-\alpha} * (\frac{1}{\beta} - 1 + \delta)$$

$$\frac{I^*}{Y^*} = \delta * (\frac{1}{\frac{Y^*}{K^*}})$$

$$\frac{C^*}{Y^*} = 1 - \frac{I^*}{Y^*}$$

Calibration of the model yields, $\alpha = 0.33, \delta = 0.012, \rho = 0.95$, the standard deviation of technology, sda = 0.007, $\beta = 0.987, \sigma = 2$ and $\eta = 2$.

## 2.2 Writing the example code: rbc.mod

### 2.2.1 Intro

In the .mod file, it is better to write the name of model. For example,

```
% Basic RBC Model
% rbc.mod
```

Matlab does not recognize any word or sentence typed after
.

### 2.2.2 Preamble

The preamble consists of the declarations to setup the endogenous variables, the exogenous variables as well as the parameters and assigns values to these parameters.

1. First of all the endogenous variables of the model like output (y), capital (k), investment (i) etc. must be specified:

   ```
   var y c k i n a;
   ```

2. After the endogenous variables the exogenous variables (i.e. the shock $\varepsilon_t$) of the model must be specified:

   ```
   varexo e;
   ```

3. Now follows the list of parameters:

   ```
   parameters alpha delta rho sigma beta eta sda;

   parameters Y\_K C\_Y I\_Y;
   ```

4. The assignment of parameter values

   ```
   alpha = 0.66;
   delta = 0.012;
   rho = 0.95;
   sda = 0.007;
   ```

```
beta = 0.987;
sigma = 2;
eta = 2;

Y\_K = (1/1-alpha)*(1/beta -1 +delta);
I\_Y = delta*(1/Y\_K);
C\_Y = 1 - I\_Y;
```

### 2.2.3 Declaration of the Model

This step starts with the instruction

```
model(linear);
```

or just with model; which declares the model as being non-linear.

Now all equilibrium conditions of the model are written exactly the way you would write them on a piece of paper. However, there are very simple rules to follow concerning variables:

- for variables decided in $t$ (like $x_t$) write $x$
- for variables decided in $t1$ (like $x_{t1}$) write $x(-1)$
- for variables decided in $t+1$ (like $x_{t+1}$) write $x(+1)$

```
model(linear);
y - n = eta*n + sigma*c;
c = c(+1) - (1/sigma)*(alpha)*Y\_K*(y(+1) - k);
C\_Y*c + I\_Y*i = y;
y = alpha*k(-1) + (1-alpha)*n + a;
delta*i = k-(1-delta)*k(-1);
a = rho*a(-1)+e;
end;
```

Note that there must be as many equations as there are endogenous variables in the model, except when computing the unconstrained optimal policy with ramsey_policy or discretionary_policy. The declaration (or the block) of the model ends with end;

### 2.2.4 Solving the Model

A typical experiment is to study the effects of a temporary shock after which the system goes back to the original equilibrium (if the model is stable. . . ). A temporary shock is a temporary change of one or several exogenous variables of the model. Temporary shocks are specified within the next block between the commands shocks; and end;:

Another typical experiment in a deterministic context is to study the transition of one equilibrium position to another, it is equivalent to analyze the consequences of a permanent shock and this in done in Dynare through the proper use of initval and endval.

To get Dynare to compute the steady state values just add:

```
steady;
```

To get informations about the eigenvalues and, thus, about the stability of the model (i.e. if its specification satisfies the Blanchard-Kahn condition) just add:

```
check;
```

The model is then solved and simulated using the stoch_simul-command for 20 periods

```
stoch_simul((hp_filter = 1600,irf=20,,order = 1)
```

which solves a stochastic (i.e. rational expectations) model, using QZ techniques. More precisely, stoch_simul; computes the decision and transition functions for the model around the steady-state. Using them, it computes impulse response functions as the difference between the trajectory of a variable following a shock at the beginning of period 1 and its steady state value as well as various descriptive statistics (moments, variance decomposition, correlation and autocorrelation coefficients). By default, the

- coefficients of the approximated decision rules as well as
- moments of the variables are reported and
- impulse response functions for each exogenous shock are plotted

Before calculating the second moments, the Hodrick-Prescott Filter(short: HP-Filter) to remove a smooth trend from data is sometimes applied. For Quarterly data, it is normally set as $\lambda = 1600$ Then, we can add option (hp_filter = 1600) to stoch_simul. Since we only deal with first order approximation in this model, the order=1 must be specified in stoch_simul. Note that while running the dynare code it stores the values of the endogenous variables of each period in corresponding vectors. For example, since all of our endogenous variables response to the shock $e$ dynare names them $c_e, y_e, i_e$ and so on (see figure 1). This is very convenient because the default dynare plots arent very pretty so that we have the opportunity to plot the impulse-response-functions the way we want. We can write the name of an endogenous variable. This permits to limit output to those variables listed here. For example,

```
stoch_simul((hp_filter = 1600,irf=20,,order = 1) y c
```

Then output is produced only for $y$ and $c$.

## 2.3 rbc.mod

The example below provides the code. Then the code is written in text file and saved and named as rbc.mod. You would run it by entering on the matlab command line `dynare rbc.`

```
\\ Basic RBC Model
\\rbc.mod

close all;
\\----------------------------------------------------------------
\\ \\ Preamble
\\----------------------------------------------------------------
\\ Defining variables
var y c k i n a;

\\ Define exogenous shock
varexo e;

\\Define parameters
parameters alpha delta rho sigma beta eta sda;

parameters Y\_K C\_Y I\_Y;

\\Calibration

\\Technology
alpha = 0.66;
delta = 0.012;
rho = 0.95;
sda = 0.007;

\\Preferences
beta = 0.987;
sigma = 2;
eta = 2;

\\ internal parameters
Y\_K = (1/1-alpha)*(1/beta -1 +delta);
I\_Y = delta*(1/Y\_K);
C\_Y = 1 - I\_Y;
```

```
\\----------------------------------------------------------------
\\ Model
model(linear);
y - n = eta*n + sigma*c;
c = c(+1) - (1/sigma)*(alpha)*Y\_K*(y(+1) - k);
C\_Y*c + I\_Y*i = y;
y = alpha*k(-1) + (1-alpha)*n + a;
delta*i = k-(1-delta)*k(-1);
a = rho*a(-1)+e;
end;

\\----------------------------------------------------------------
\\Computation
\\Setting initial values
initval;
y = 0;
k = 0;
c = 0;
i = 0;
n = 0;
a = 0;
e = 0;
end;

\\Setting the variance of shocks
shocks;
var e = sda^2;
end;

\\Computing steady state
steady;

check;

\\Computing solution and do simulation
stoch_simul(hp_filter = 1600, rif=20,order = 1);
```

## 2.4 The Output

At first, Dynare starts to run the code and to generate results.

```
Configuring Dynare ...
[mex] Generalized QZ.
[mex] Sylvester equation solution.
[mex] Kronecker products.
[mex] Sparse kronecker products.
[mex] Local state space iteration (second order).
[mex] Bytecode evaluation.
[mex] k-order perturbation solver.
[mex] k-order solution simulation.
[mex] Quasi Monte-Carlo sequence (Sobol).
[mex] Markov Switching SBVAR.


Starting Dynare (version 4.4.3).
Starting preprocessing of the model file ...
Found 6 equation(s).
Evaluating expressions...done
Computing static model derivatives:
 - order 1
Computing dynamic model derivatives:
 - order 1
Processing outputs ...done
Preprocessing completed.
Starting MATLAB/Octave computing.
```

Due to the steady;-command the steady-states of the endogenous variables are computed. Since the shock in a stable model is considered as zero, all values are equal to zero.

```
STEADY-STATE RESULTS:


y   0
c   0
k   0
i   0
n   0
a   0
```

Since we have used check; the eigenvalues of the system and a confirmation of the Blanchard-Kahn condition is displayed. Note that the eigenvalues are given in ascending order.

```
EIGENVALUES:
         Modulus             Real        Imaginary

            0.95             0.95                0
          0.9917           0.9917                0
           1.002            1.002                0
       1.695e+16       -1.695e+16                0


There are 2 eigenvalue(s) larger than 1 in modulus
for 2 forward-looking variable(s)

The rank condition is verified.
```

The model summary just counts the various variable types of the model:

```
MODEL SUMMARY

  Number of variables:         6
  Number of stochastic shocks: 1
  Number of state variables:   2
  Number of jumpers:           2
  Number of static variables:  2


MATRIX OF COVARIANCE OF EXOGENOUS SHOCKS

Variables       e
e            0.000049

POLICY AND TRANSITION FUNCTIONS
              y             c         k          i          n          a
k(-1)    0.288623     0.236947    0.991715   0.309572  -0.061757   0
a(-1)    1.204413     0.032625    0.020153   1.679448   0.379721   0.950000
e        1.267803     0.034342    0.021214   1.767840   0.399706   1.000000

THEORETICAL MOMENTS (HP filter, lambda = 1600)

VARIABLE      MEAN    STD. DEV.   VARIANCE
y              0.0000     0.0115     0.0001
```

| | | | |
|---|---|---|---|
| c | 0.0000 | 0.0003 | 0.0000 |
| k | 0.0000 | 0.0007 | 0.0000 |
| i | 0.0000 | 0.0161 | 0.0003 |
| n | 0.0000 | 0.0037 | 0.0000 |
| a | 0.0000 | 0.0091 | 0.0001 |

MATRIX OF CORRELATIONS (HP filter, lambda = 1600)

| Variables | y | c | k | i | n | a |
|---|---|---|---|---|---|---|
| y | 1.0000 | 0.8831 | 0.1860 | 1.0000 | 0.9996 | 0.9999 |
| c | 0.8831 | 1.0000 | 0.6253 | 0.8812 | 0.8692 | 0.8749 |
| k | 0.1860 | 0.6253 | 1.0000 | 0.1821 | 0.1576 | 0.1691 |
| i | 1.0000 | 0.8812 | 0.1821 | 1.0000 | 0.9997 | 0.9999 |
| n | 0.9996 | 0.8692 | 0.1576 | 0.9997 | 1.0000 | 0.9999 |
| a | 0.9999 | 0.8749 | 0.1691 | 0.9999 | 0.9999 | 1.0000 |

COEFFICIENTS OF AUTOCORRELATION (HP filter, lambda = 1600)

| Order | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| y | 0.7136 | 0.4717 | 0.2717 | 0.1105 | -0.0158 |
| c | 0.7787 | 0.5755 | 0.3934 | 0.2343 | 0.0987 |
| k | 0.9608 | 0.8657 | 0.7336 | 0.5804 | 0.4185 |
| i | 0.7135 | 0.4715 | 0.2715 | 0.1103 | -0.0159 |
| n | 0.7131 | 0.4709 | 0.2708 | 0.1096 | -0.0166 |
| a | 0.7133 | 0.4711 | 0.2711 | 0.1098 | -0.0163 |

Total computing time : 0h00m03s

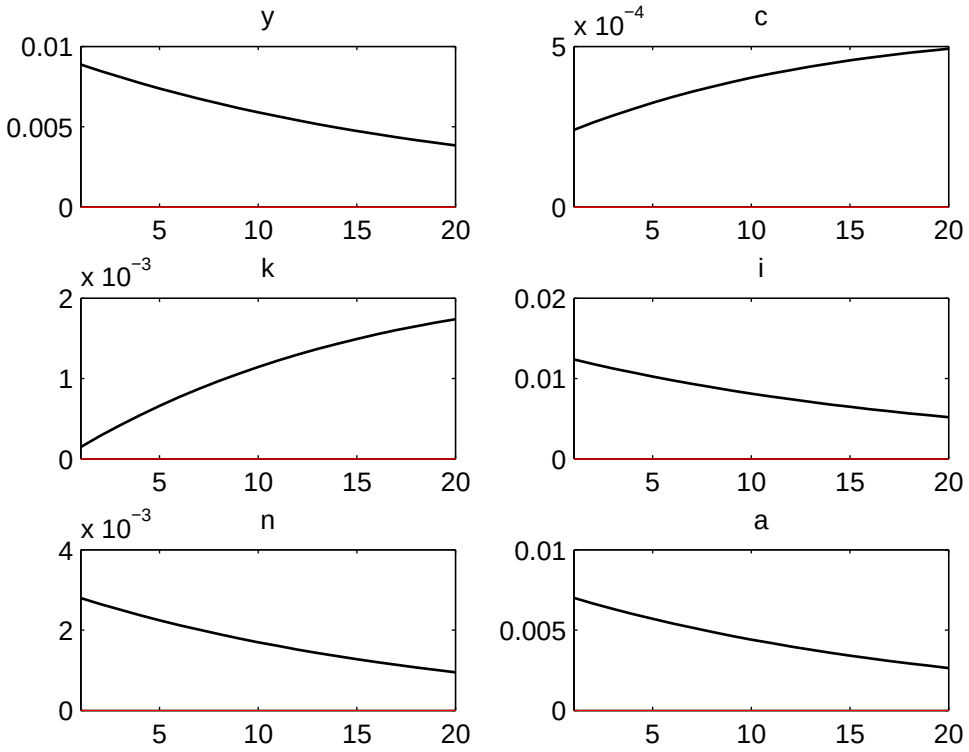


Figure 1: Impulse respnses

# 3 Using Dynare to solve optimal policy numerically

Dynare has tools to compute optimal policies for various types of objectives. You can either solve for optimal policy under commitment with "ramsey policy", for optimal policy under discretion with "discretionary policy".

The computation of optimal policy under commitment is controlled by main command "ramsey_policy" and auxiliary commands "planner_objective", and "eliminate_lagrange_multipliers". The planner objective specifies the objective of the policy planner, ramsey policy controls the computation of Ramsey policy, and eliminate_lagrange_multipliers" eliminates the Lagrange multipliers from the solution.

The computation of optimal policy under discretion is controlled by main command discretionary_policy".

## 3.1 Declare the policy objective

To compute optimal policy, the objective function must be declared. This can be done with the planner_objective command. You need to give the one-period objective, not the discounted lifetime objective.

```
planner objective MODEL EXPRESSION;
```

MODEL EXPRESSION is the one-period objective, not the discounted lifetime objective. The discount factor is given by the planner_discount option of ramsey_policy command and

discretionary_policy. For example, if the period objective function for central bank is

$$\frac{1}{2}\left(y_t^2 + \alpha\pi_t^2\right)$$

then, following command will specify the objective function

```
planner_objective 1/2 y^2 + alpha/2 inf^2;
```

With ramsey_policy, you are not limited to quadratic objectives: you can give any arbitrary nonlinear expression. With discretionary_policy, the objective function must be quadratic.

```
planner_objective MODEL EXPRESSION;
```

## 3.2 Optimal policy under commitment

The following command computes the first order approximation of the policy that maximizes the policy maker objective function submitted to the constraints provided by the equilibrium path of the economy. The planner objective must be declared with the planner_objective command.

```
ramsey_policy (OPTIONS. . . ) [VARIABLE_NAME. . . ];
```

This command accepts all options of stoch_simul. However, we sometimes declares the discount factor of the central planner (Default: 1.0)

```
planner\_discount = EXPRESSION
```

VARIABLE NAME name of an endogenous variable. This permits to limit output to those variables listed here. Note that only first order approximation is available (i.e. order=1 must be specified).

The name of policy instrument can be specified under the following command:

```
instruments = ([VARIABLE NAME, . . . ])
```

Naming policy instruments is not necessary to find the solution. It is however useful, when providing a analytical, recursive solution for the steady state under optimal policy. Note that in that case, the term instrument is used somewhat abusively. For example,

```
ramsey_policy(planner_discount=0.99,instruments=(i),order=1);
```

### 3.2.1 output

Command ramsey_ policy generates the same output as stoch_simul. In addition, it provides two approximate evaluation of the current value of the objective under optimal policy. These evaluations are computed on the basis of a second order approximation of the objective function and of the first order conditions. The first evaluation is computed while setting the initial value of the Lagrange multipliers to 0. The second evaluation, on the basis of setting them to their steady state value.

## 3.3 Optimal policy under discretion

```
discretionary_policy (OPTIONS. . . ) [VARIABLE_NAME. . . ];
```

This command computes an approximation of the optimal policy under discretion. The algorithm implemented is essentially an LQ solver, and is described by Dennis (2007). You should ensure that your model is linear and your objective is quadratic. Also, you should set the linear option of the model block. This command accepts the same options than ramsey_policy. However, the name of policy instrument must be specified under the discretionary rule.