Introduction to Dynare

Advanced Macroeconomics Tutorials

Matteo Coronese

April 22, 2021

Scuola Superiore Sant'Anna

What is Dynare?

DSGE models can be rather painful (or just impossible) to solve analytically, as they are typically involve a set of highly non-linear equations.

Dynare is a set of *Matlab* functions which solves, simulates and estimates these models for us.

In a nutshell:

- Solve a model: find the steady state values.
- Simulate a model: obtain time series for each variable or shock the model and compute Impulse Response Functions (IRFs).
- Estimate a model: feed it with real-world data and obtain estimates for model parameters.

Getting Started

First, tell *Matlab* where to find *Dynare*.[1]

- First option:
  - In Windows: addpath c:/dynare/4.x.y/matlab
  - In Linux/GNU: addpath /usr/lib/dynare/matlab
  - In MacOS: addpath /Applications/Dynare/4.x.y/matlab

  MATLAB will **not remember** this setting next time you run it, and you will have to do it again.
- Second option: Select the "Set Path" entry in the "File" menu, then click on "Add Folder...", and select the matlab subdirectory of 'your Dynare installation. Apply the settings by clicking on "Save". MATLAB will **remember** this setting next time you run it.

Reference

For every doubt, refer to the comprehensive on-line Manual.

[1] Current version to date is 4.6.1

How to Run Dynare

To work with *Dynare*, you typically write your code in a `.mod` file. You can write your script as a regular *Matlab* `.m` file, just remember to save it as a `.mod`.

To run your script, just type in your Matlab command line

```
dynare yourscript.mod
```

You will visualize results directly in the *Matlab* command window. Objects created (e.g. simulated variables, theoretical moments etc) are directly available in your *Matlab* workspace.

Clearly, you can create a separate `.m` file where you invoke different *Dynare* scripts, to reproduce a more complex work-flow.

The `.mod` file has a very simple structure. Let's go through it with a very simple example.

A Simple RBC Model

Consider the following optimization problem:

$$\begin{aligned} \max_{c_t} \quad & E_0 \sum_{t=0}^{\infty} \beta^t \frac{c_t^{1-\sigma} - 1}{1-\sigma} \\ \text{s.t.} \quad & k_{t+1} = a_t k_t^{\alpha} - c_t + (1-\delta)k_t \end{aligned} \tag{1}$$

with the representative consumer optimizing is life-time utility. The meaning of the law of motion of capital is simply $k_{t+1} = i_t + (1-\delta)k_t$, that is: previous capital inherited, minus capital depreciation, plus investment. Thus

$$\begin{aligned} y_t &= a_t k_t^{\alpha} \\ i_t &= y_t - c_t \end{aligned} \tag{2}$$

Finally, TFP is assumed to follow a mean zero AR(1) in logarithms:

$$\ln a_t = \rho \ln a_{t-1} + \varepsilon_t \tag{3}$$

A Simple RBC Model II

F.O.C. from the maximization problem above yields

$$c_t^{-\sigma} = \beta E_t c_{t+1}^{-\sigma}(\alpha a_{t+1} k_{t+1}^{\alpha-1} + (1-\delta)) \tag{4}$$

We assume the transversality condition $\lim_{t\to\infty} \beta^t c_t^{-\sigma} k_{t+1} = 0$ holds true. Our model can thus be expressed as a system with 4 equations and 4 variables.

$$\begin{aligned}
c_t^{-\sigma} &= \beta E_t c_{t+1}^{-\sigma}(\alpha a_{t+1} k_{t+1}^{\alpha-1} + (1-\delta)) && \text{Euler Equation} \\
k_{t+1} &= a_t k_t^{\alpha} - c_t + (1-\delta)k_t && \text{Capital Accumulation} \\
y_t &= a_t k_t^{\alpha} && \text{Production Function} \\
\ln a_t &= \rho \ln a_{t-1} + \varepsilon_t && \text{Exogenous Process}
\end{aligned} \tag{5}$$

The .mod file - Declare Variables

We are now ready to write our .mod file. The first thing you need to do is to declare your endogenous variables, that is y_t,k_t,a_t and c_t. This is done through the command var:

```
var y, k, a, c;
```

Remember

Always put a semicolon ";" at the end of each line. Otherwise *Matlab* will throw an error.

Then declare your exogenous variables, that is those variables which are not involved in the equilibrium: your shocks, in this case ε_t. This is done through the command varexo:

```
varexo e;
```

The .mod file - Declare Parameters

The next thing you need to declare is the set of model parameters, through the command parameters. In our case, the set of parameters is given by α, β, δ, ρ, σ and the standard deviation σ^a of the shock ε_t.

```
parameters  alpha beta delta rho sigma sigma_a;
```

Note

Unless specified, shocks are assumed to be normally distributed. In our case, $\varepsilon_t \sim N(0, \sigma_a^2)$.

After you declared your parameters, you can directly type their values below:

```
alpha = 0.33;
beta = 0.99;
delta = 0.025;
rho = 0.95;
sigma = 2;
sigma_a = 0.01;
```

The .mod file - Declare the Model

It's now time to declare the structure of the model, i.e. its equation.

To to that, start your model section by typing `model;`, then type your equations and close the model section through `end;`. Some timing conventions:

- Write you variables at current time (e.g. c_t) simply through their name (in this case `c`).
- If a variable appears lagged (e.g. c_{t-1}), write `c(-1)`.
- If a variable is leading (e.g. $E_t c_{t+1}$), write `c(+1)`.

Dynare requires that predetermined variables (i.e. state variables, like the capital stock) show up as dated $t-1$ in the time t equations and t in the $t+1$ equations. That's how you indicate to *Dynare* which are your state variables.

- In our case, just lag capital stock in each equation.

Put another way, for stock variables, the default in *Dynare* is to use a "stock at the end of the period" concept, instead of a "stock at the beginning of the period" convention.

Beside these conventions, shall we write our model in levels or in logs?

A Quick Note on Log-Linearization

The process of log-linearization allows us to express variables as percentage deviation from the steady state, which is quite handy, especially when we visualize IRFs.

Why logs? Differences in logs can be interpreted as percentage deviations.

Consider a generic variable x_t, with a steady state value $\bar{x}$. Taylor expanding its logarithm around $\bar{x}$ yields:

$$\ln x \approx \ln \bar{x} + \frac{1}{\bar{x}}(x - \bar{x})$$

$$\ln x - \ln \bar{x} \approx \frac{x - \bar{x}}{\bar{x}} \equiv \hat{x}$$

Let's try to log-linearize the production function $y_t = a_t k_t^{\alpha}$. First take logs on both sides

$$y_t = a_t k_t^{\alpha} \rightarrow$$

$$\ln y_t = \ln a_t k_t^{\alpha} \rightarrow$$

$$\ln y_t = \ln a_t + \alpha \ln k_t$$

Then Taylor-expand around steady state values $\bar{a}$,$\bar{y}$ and $\bar{k}$.

A Quick Note on Log-Linearization II

$$\ln \bar{y} + \frac{1}{\bar{y}}(y_t - \bar{y}) = \ln \bar{a} + \frac{1}{\bar{a}}(a_t - \bar{a}) + \alpha \ln \bar{k} + \alpha \frac{1}{\bar{k}}(k_t - \bar{k}) \rightarrow$$

$$\frac{y_t - \bar{y}}{\bar{y}} = \frac{a_t - \bar{a}}{\bar{a}} + \alpha \frac{k_t - \bar{k}}{\bar{k}} \rightarrow$$

$$\hat{y}_t = \hat{a}_t + \alpha \hat{k}_t$$

Log-linearize your model by hand can be painful.

Now, *Dynare* **only linearizes the model for you** (i.e solves the model in the neighborhood of the steady state, where the approximation error is small enough). Ways to obtain log-linearization:

- As it is always true that $exp(x_t) = Z \Leftrightarrow ln(Z) = x_t$, you can write every variable x_t in your model as $\exp x_t$. *Dynare* will interpret x_t (the initial variables you declared) as the log of the variable of interest Z. Remember to change your initial values accordingly: if $x_0 = 4$, write $x_0 = \ln(4)$ (more on that later).
- Just write your model normally, and then add the `loglinear` option to the `stoch_simul()` command (more on that later).

A Quick Note on Log-Linearization III

Let's verify that rewriting our variables will do the trick.

$$
\begin{aligned}
y_t &= a_t k_t^{\alpha} \rightarrow \\
\exp(\ln y_t) &= \exp(\ln a_t)\exp(\ln k_t^{\alpha}) \rightarrow \\
\exp(\ln y_t) &= \exp(\ln a_t)\exp(\alpha \ln k_t) \rightarrow \\
\exp(\ln \bar{y}) + \frac{1}{\bar{y}}\exp(\ln \bar{y})(y_t - \bar{y}) &= \ldots \\
\cdots &= \exp(\ln \bar{a})\exp(\alpha \ln \bar{k}) + \exp(\ln \bar{a})\exp(\alpha \ln \bar{k})\frac{1}{\bar{a}}(a_t - \bar{a}) + \ldots \\
\ldots &+ \exp(\ln \bar{a})\exp(\alpha \ln \bar{k})\frac{1}{\bar{k}}\alpha(k_t - \bar{k}) \rightarrow \\
\exp(\ln \bar{y}_t)\left[1 + \frac{y_t - \bar{y}}{\bar{y}}\right] &= \exp(\ln \bar{a})\exp(\alpha \ln \bar{k})\left[1 + \frac{a_t - \bar{a}}{\bar{a}} + \alpha\frac{k_t - \bar{k}}{\bar{k}}\right] \rightarrow \\
\hat{y}_t &= \hat{a}_t + \alpha\hat{k}_t
\end{aligned}
$$

The .mod file - Declare the Model II

Remind that our equations are:

$$
\begin{aligned}
c_t^{-\sigma} &= \beta E_t c_{t+1}^{-\sigma}(\alpha a_{t+1} k_{t+1}^{\alpha-1} + (1-\delta)) && \text{Euler Equation} \\
k_{t+1} &= a_t k_t^{\alpha} - c_t + (1-\delta)k_t && \text{Capital Accumulation} \\
y_t &= a_t k_t^{\alpha} && \text{Production Function} \\
\ln a_t &= \rho \ln a_{t-1} + \varepsilon_t && \text{Exogenous Process}
\end{aligned}
$$

We thus declare the model as follows:

```
model;
exp(c)^(-sigma) = beta*(exp(c(+1))^(-sigma))*(alpha*exp(a(+1))*(exp(k))^(alpha-1) + (1-delta));
exp(k) = exp(a)*exp(k(-1))^(alpha) - exp(c) + (1-delta)*exp(k(-1));
exp(y) = exp(a)*exp(k(-1))^(alpha);
a = rho*a(-1) + e;
end;
```

Note that the last equation reports `a` without `exp()`, as it was already defined in logs.

The .mod file - The initval Block

Initial values are declared within the `initval` block, delimited by the commands `initval ;` and `end;`. It has two main purposes:

- In both stochastic and deterministic (i.e. with perfect foresight) simulations, it provides initial guess values for steady state computations.
- In deterministic simulations, provides guess values for non-linear solvers.

We here focus on stochastic simulations (i.e. simulations corresponding to a random draw of the shocks).

- If you do not provide an `initval` block, the solver will use default ones. Bad initial values might lead, in complicated models, to non-convergence.
- It is not necessary to declare 0 as initial value for exogenous stochastic variables, since it is the only possible value.
- You have to use your knowledge of the model to write good initial conditions.

The .mod file - Declare Shocks

- Through the command steady, we get the (deterministic) steady state.
- In a stochastic framework, we study the effects of random draws of the shocks. In *Dynare*, these random values follow a normal distribution with zero mean, but it belongs to the user to specify the variability of these shocks.
- Non-zero elements of variance-covariance matrix of shocks are entered in the shocks block (variances of single shocks and, if any, covariances among them).

```
initval;
k = log(29);
y = log(3);
a = 0;
c = log(2.5);
end;

steady;

shocks;
var e = sigma_a^2;
end;
```

The .mod file - Stochastic Simulation

Finally, trough the command stoch_simul(), we can finally obtain our stochastic simulation. Default output:

- Steady-state values of endogenous variables.
- Model summary.
- Covariance matrix of shocks.
- Policy and transition functions.
- Theoretical first and second moments.
- Theoretical correlation matrix.
- Theoretical autocovariances up to order 5.
- Impulse responses.

The .mod file - Stochastic Simulation II

Some options of the stoch_simul():

- order: order of Taylor approximation (1, 2, 3).
- k_order_solver: use a C++ solver (more complicated, check manual).
- ar = sets the number of autocorrelations to be computed (default=5)
- irf= sets the number of responses to be computed; if equal to 0 suppresses plotting of impulse response functions.
- relative_irf: computes normalized impulse responses (w.r.t. shock size).
- periods: species the number of periods to use in simulations. Periods triggers the computation of moments and correlation using simulated data rather than the population solution.
- drop = Integer: Number of points dropped at the beginning of simulation before computing the summary statistics (default = 100).
- loglin = Consider the model in logs.

```
stoch_simul(order = 1, irf = 40);
```

Making Sense of Output

Firs, *Dynare* prints steady state values (required through steady), the variance-covariance matrix of shocks (specified in the shocks block) and a model summary, counting the various "types" of variable detected in the model (states, jumpers, and static):

- Dynare counts things which show up in the equilibrium conditions just dated t as static variables (in our case y).
- Things which show up with a $t-1$ as states. In our case, k and a.
- Things which show up with a $t+1$ as jumpers. In our case, a and c.
- They do not necessarily add up to the number of variables in the model. Something could show up as both a state and a jumper.
- In our case, a appears lagged (as a state) and with a "+1" in the Euler equation (jumper).

Making Sense of Output II

Then *Dynare* prints policy and transition functions. How to interpret them? Remember again our equations (FOCs + equilibrium conditions):

$$
\begin{aligned}
c_t^{-\sigma} &= \beta E_t c_{t+1}^{-\sigma}\left(\alpha a_{t+1} k_{t+1}^{\alpha-1} + (1-\delta)\right) && \text{Euler Equation} \\
k_{t+1} &= a_t k_t^{\alpha} - c_t + (1-\delta)k_t && \text{Capital Accumulation} \\
y_t &= a_t k_t^{\alpha} && \text{Production Function} \\
\ln a_t &= \rho \ln a_{t-1} + \varepsilon_t && \text{Exogenous Process}
\end{aligned}
$$

In a matrix notation, our system can be defined as

$$
E_t\{f(\mathbf{y}_{t+1}, \mathbf{y}_t, \mathbf{y}_{t-1}, \mathbf{u}_t, \theta)\} = \mathbf{0} \tag{6}
$$

where

$$
\begin{aligned}
\mathbf{y}_t &= \text{Vector of endogenous variables} \\
\mathbf{u}_t &= \text{Vector of exogenous shocks} \\
\theta &= \text{Vector of parameters}
\end{aligned}
$$

Making Sense of Output III

A linearized version of the model in Equation 6 can thus be written as

$$\Pi_1 \mathbf{y}_{t+1} = \Pi_2 \mathbf{y}_t + \Pi_3 \mathbf{y}_{t-1} + \Gamma_1 \mathbf{u}_t \tag{7}$$

This way to express the model is the so-called state-space form.

Solving a DSGE model means essentially to find the unknown function $g(\cdot)$, though which we are able to express the current values of your variables as a function of past values and current shocks:

$$y_t = g(\mathbf{y}_{t-1}, \mathbf{u}_t)$$

also know as policy function. If plugged into the original system, it obviously solves the model. The linearized stochastic solution found by *Dynare* is of the form

$$\mathbf{y}_t = \bar{\mathbf{y}} + \mathbf{A}(\mathbf{z}_{t-1} - \bar{\mathbf{z}}) + \mathbf{B}\mathbf{u}_t \tag{8}$$

What is $\mathbf{z}$? It is a subset of $\mathbf{y}$, containing only **state variables**. The idea is to express the evolution of all variables $\mathbf{y}$ in terms of past values of state variables only $\mathbf{z}$ (plus currents shocks). More on this during next lecture.

The policy and transition functions gives you the content of matrixes $\bar{\mathbf{y}}$, $\mathbf{A}$ and $\mathbf{B}$.

Making Sense of Output IV

Quite obviously, $\bar{\mathbf{y}}$ represents steady state values (the "constant" line in *Dynare*'s output.)

POLICY AND TRANSITION FUNCTIONS

	y	k	a	c
Constant	1.103709	3.344571	0	0.835782
k(-1)	0.330000	0.974256	0	0.440543
a(-1)	0.950000	0.072913	0.950000	0.345784
e	1.000000	0.076751	1.000000	0.363983

Note

- The interpretation of your coefficients varies if you write you model in a log-form or in levels. In the former case, $\mathbf{y}_t$ actually represents $\ln \mathbf{y}_t$.
- Row names should be read as difference with respect to steady state levels.
- For example, 0.33 (row k(–1), column y) is the effect of $(\ln k_{t-1} - \ln \bar{k})$ on $\ln y_t$.

Making Sense of Output V

Consider again the effect of capital k(−1) on output y, and look at Equation 8. Ignoring every other variable, we could write:

$$\ln y_t = \ln \bar{y} + 0.33 * (\ln k_{t-1} - \ln \bar{k})$$

What would show up if we run the level model? Since we know that, at a first linear approximation, $\ln(x_t) - \ln \bar{x} = \frac{x_t - \bar{x}}{\bar{x}}$, some back-of-the-envelope algebra yields

$$\begin{aligned} \ln y_t - \ln \bar{y} =& 0.33 * (\ln k_{t-1} - \ln \bar{k}) \\ \frac{y_t - \bar{y}}{\bar{y}} =& 0.33 * (\frac{k_{t-1} - \bar{k}}{\bar{k}}) \\ y_t =& \bar{y} + \left[0.33 * \frac{\bar{y}}{\bar{k}}\right] (k_{t-1} - \bar{k}) \end{aligned}$$

From the table above we know that $\ln \bar{y} = 1.10.379$ and $\ln \bar{k} = 3.344571$. Thus we should expect a value

$$0.33 * \frac{\bar{y}}{\bar{k}} = 0.33 * \frac{\exp(1.10379)}{\exp(3.344571)} = 0.0351$$

Check yourselves!

Making Sense of Output VI

Further outputs:

- Variable moments (theoretical if `periods` is not specified, simulated otherwise). They also depends on your model specification.
- Contemporaneous correlations among variables.
- Autocorrelations of each variable.

Dynare also plots the Impulse Response Functions.

- The Impulse Response Functions (IRFs) represent the trajectories of each variable when the model is in the steady state and is perturbed by one or multiple shocks. In our case, we only have one shock. The magnitude of the shock is equal to one standard deviation (in our case `sigma_a`).
- If the variance of an exogenous variable is set to zero, this variable will appear in the report on policy and transition functions, but isn't used in the computation of moments and of Impulse Response Functions. Setting a variance to zero is an easy way of removing an exogenous shock.

Where is the output stored?

You can find all your output in Matlab workspace, under different objects

- Impulse Response Functions: reaction of variable x to shock e is recorded in a the object x_e.
- Steady state values are recorded in the object oo.dr.ys.
- Policy function:
  - Coefficients of state variables (the **A** matrix in Equation 8) are stored the object oo.dr.ghx.
  - Coefficients of shock variables (the **B** matrix in Equation 8) are stored the object oo.dr.ghu.
- Simulated variables are recorded in objects with their names (e.g. time series for c_t as c).