2.1. Dynare Basics

Occasionally Binding Constraints in DSGE Models

Jonathan Swarbrick[1]
Bank of Canada

Bank of Canada – CMFE-Carleton Virtual Series
Advanced Topics in Macroeconomic Modelling

January 2021

[1]The views expressed are those of the authors and should not be interpreted as reflecting the views of the Bank of Canada.

Some Dynare Basics

Throughout the rest of the course, we will use dynare

You can install dynare from https://www.dynare.org/download/

- Dynare will take Taylor approximations and solve the models
- Modeller only needs to enter the non-linear model conditions
- The model is coded up in a **FILENAME.mod** file, and solved from matlab command window:

```
dynare FILENAME
```

- There is comprehensive documentation (https://www.dynare.org/manual/) and an active forum (https://forum.dynare.org/)

Model file structure: variables & parameters

The model file is split into sections:

1. Endogenous variables declared after `var`, exogenous variables after `varexo` and parameters after `parameters`:

```
var c h b z;

varexo epsz;

parameters betta R delta rho chi sigma;
```

2. The parameters are then given values:

```
betta = 0.99;
R = 1/betta;
delta = 0.01;
rho = 0.9;
chi = 0.5;
sigma = 0.01;
```

Model file structure: the model block

3. Model conditions are given between `model;` and `end;`
   - For N endogenous variables, there should be N equations.

```
model;

c = ( exp( z ) + R * b(-1) -  b ) / ( 1 + chi );
h = 1 - chi * c / exp(z);
1/c = 1/c(+1) - delta * b;
z = rho * z(-1) + sigma * epsz;

end;
```

Note on timings

- Predetermined variables in period t must be dated $t-1$, e.g., in RBC model, capital:

$$K_t = I_t + (1-\delta)K_{t-1} \tag{1}$$

  K_{t-1} is what is used in production. In the example model, borrowing is decided at the end-of-period and so debt taken into period t is b_{t-1}.
- the expectation operator is placed around each line $\mathbb{E}_t\left[\cdot\right]$

Model file structure: stochastic simulation

4. The steady state must be either given analytically between `steady_state_model;` and `end;` or numerically with an external function.
   In this case, we can solve exactly:

```
steady_state_model;
z = 0;
b = 0;
c = 1 / ( 1 + chi );
h = c;
end;
```

5. We define the variance of shocks

```
shocks;
    var epsz = 1;
end;
```

6. Choose solution method and simulation options, e.g.,:

```
stoch_simul( order = 1, irf = 80, periods = 10000 );
```

   This will tell dynare to compute a 1st-order approximation, compute impulse response functions over 80 periods, and simulated time-series over 10,000 periods.

Dynare Macroprocessor

Dynare has a set of ‘macro’ commands to write more efficient code:

- **Model local variables**: I use these *all the time*.

  You can create additional variables that are not endogenous model variables.

  - Do not include after `var`, write a definition in model block preceded with `#`, e.g.:

    ```
    # c = ( exp( z ) + R * b(-1) -  b ) / ( 1 + chi );
    h = 1 - chi * c / exp(z);
    ```

    which is exactly equivalent to

    ```
    h = 1 - chi * (  ( exp( z ) + R * b(-1) -  b ) / ( 1 + chi ) ) / exp(z);
    ```

  - I usually do it for log transformations (i.e., so log(c) can be my endogenous variable):

    ```
    # c exp( log_c );
    ```

    where `log_c` is defined as a variable instead of c.
  - Very useful for transformations and minimizing number of model variables
- **Others**: loops, conditionals etc, see https://www.dynare.org/manual/the-model-file.html#macro-processing-language

1st-order perturbation

- Let the actions/controls be $y_t = \begin{bmatrix} b_t & c_t & h_t & z_t \end{bmatrix}'$,
- and the state $x_t = \begin{bmatrix} b_{t-1} & z_{t-1} & \varepsilon_t \end{bmatrix}'$. The linear solution (policy) is

$$y_t = \bar{y} + A\left(x_t - \bar{x}\right) \tag{2}$$

or

$$b_t = \bar{b} + a_{b,b}\left(b_{t-1} - \bar{b}\right) + a_{b,z}\left(z_{t-1} - \bar{z}\right) + a_{b,\varepsilon}\varepsilon_t \tag{3}$$

$$c_t = \bar{c} + a_{c,b}\left(b_{t-1} - \bar{b}\right) + a_{c,z}\left(z_{t-1} - \bar{z}\right) + a_{c,\varepsilon}\varepsilon_t \tag{4}$$

$$h_t = \bar{h} + a_{h,b}\left(b_{t-1} - \bar{b}\right) + a_{h,z}\left(z_{t-1} - \bar{z}\right) + a_{h,\varepsilon}\varepsilon_t \tag{5}$$

$$z_t = \rho z_{t-1} + \sigma\varepsilon_t \tag{6}$$

More convenient linear notation

Note, this is equivalent to $y_t = \begin{bmatrix} b_t & c_t & h_t \end{bmatrix}'$, and $x_t = \begin{bmatrix} b_{t-1} & z_t \end{bmatrix}'$. and

$$b_t = \bar{b} + a_{b,b}\left(b_{t-1} - \bar{b}\right) + a_{b,z}\left(z_t - \bar{z}\right) \tag{7}$$

$$c_t = \bar{c} + a_{c,b}\left(b_{t-1} - \bar{b}\right) + a_{c,z}\left(z_t - \bar{z}\right) \tag{8}$$

$$h_t = \bar{h} + a_{h,b}\left(b_{t-1} - \bar{b}\right) + a_{h,z}\left(z_t - \bar{z}\right) \tag{9}$$

But dynare uses the former structure.

- At 1st order, the shock σ does not affect these coefficients
- At 2nd order, the shock σ *only* affects the constant

Dynare Output

Dynare reports several outputs to screen:

- Steady state, eigenvalues, model summary, shock covariance matrix
- Policy and transition functions
- Theoretical/simulated moments
- Cross-/auto-correlations

Policy and transition functions

POLICY AND TRANSITION FUNCTIONS

	c	h	b	mu	z
Constant	0.666667	0.666667	0	0	0
b(-1)	0.055864	-0.027932	0.926305	0	0
z(-1)	0.284978	0.157511	0.472533	0	0.900000
epsz	0.003166	0.001750	0.005250	0	0.010000

Policy and transition functions

POLICY AND TRANSITION FUNCTIONS

	c	h	b	mu	z
Constant	0.666667	0.666667	0	0	0
b(-1)	0.055864	-0.027932	0.926305	0	0
z(-1)	0.284978	0.157511	0.472533	0	0.900000
epsz	0.003166	0.001750	0.005250	0	0.010000

Corresponds to the policy function:

$$b_t = 0.926305\left(b_{t-1} - \bar{b}\right) + 0.472533\left(z_{t-1} - \bar{z}\right) + 0.005250\varepsilon_t \quad (10)$$

noting variables are on RHS are deviations, or

$$b_t = g\left(b_{t-1}, z_t\right) \approx 0.926305 b_{t-1} + 0.5250 z_t \quad (11)$$

Theoretical/simulated moments

- If `periods` is set > 0 so a time-series is computed, moments will be conditional on this simulation (ergodic)
- If `periods=0`, dynare will report theoretical moments
- At 1st order, solution is in form of linear state space model - exhibits analytical solution to moments
- Kim et al. (2008) show that at 2nd order, second moments based on linear terms are correct to 2nd order
  - That is, higher order moments the same for 1st- and 2nd-order perturbation
  - The first moments will differ

Dynare outputs

Dynare stores output in matlab structs in the workspace: **M_.** and **oo_.**

- **M_.** stores information about the model including solution settings, names of variables and parameters, parameter values, shock covariance structure
- **oo_.** stores solutions, e.g.: steady state, policy function coefficients, moments and simulations.
- **options_.** stores broad set of options

The policy function

$$b_t = \underbrace{\bar{b}}_{\texttt{oo\_.dr.ys(i)}} + \overbrace{0.926305}^{\texttt{oo\_.dr.ghx(i,j)}} \left(b_{t-1} - \bar{b}\right) + \underbrace{0.472533}_{\texttt{oo\_.dr.ghx(i,k)}} (z_{t-1} - \bar{z}) + \overbrace{0.005250}^{\texttt{oo\_.dr.ghu(i)}} \varepsilon_t \qquad (12)$$

where the indices correspond to:

- `i = oo_.dr.order_var(strmatch('b',M_.endo_names,'exact'))` to give b var index
- `j = find(oo_.dr.state_var==strmatch('b',M_.endo_names ,'exact'))` to give state var index b
- `k = find(oo_.dr.state_var==strmatch('z',M_.endo_names ,'exact'))` to give state var index z

Impulse response functions

- At 1st order, IRFs are independent of the current state, and size of shock doesnt matter
  - Double size of shock $\Longrightarrow$ double size of impact
  - Generalized (unconditional) IRF $=$ (conditional) IRF

Impulse response functions

- At 1st order, IRFs are independent of the current state, and size of shock doesnt matter
  - Double size of shock $\implies$ double size of impact
  - Generalized (unconditional) IRF = (conditional) IRF
- At 2nd order, the current state-of-the-world and shock sign/size matters
  - Conditional IRFs will differ from unconditional
  - If you set all shocks to zero except the one IRF shock this will given an ‘MIT’ shock
  - ‘Conditional’ IRF/forecast should set the initial conditions but still integrate future uncertainty via Monte Carlo
  - ‘Unconditional’ IRF starts from unconditional mean (in practice the erogodic mean)

Simulating Impulse response functions

Compute GIRFs as follows:

1. For replication i, draw a sequence of model shocks for periods $t = 1 : d + T$
2. Simulate the model twice to give time-series y_1 and y_2. In the second simulation, include the IRF shock of interest in period $t = d + 1$.
3. Take the difference $y = y_2 - y_1$, and remove first d periods.
4. Repeat for $i = 1 : R$ and take the average.

Simulating Impulse response functions

Compute GIRFs as follows:

1. For replication i, draw a sequence of model shocks for periods $t = 1 : d + T$
2. Simulate the model twice to give time-series y_1 and y_2. In the second simulation, include the IRF shock of interest in period $t = d + 1$.
3. Take the difference $y = y_2 - y_1$, and remove first d periods.
4. Repeat for $i = 1 : R$ and take the average.

Dynare already computes GIRFs by default with $R = 50$ (replications) and $d = 100$ (drop)

- To change, use:

```
stoch_simul( order=2 , irf=100 , replic=400 , drop=200 );
```

Which sets $T = 100$, $R = 400$ and $d = 200$.

Uncertainty bands

If you code the GIRF simulation yourself, you can save all replications

- These will allow you to construct forecast uncertainty bands
- Note, these are different from bands stemming from parameter uncertainty

Pruning

At 1st order, simulations/IRFs are shock-size invariant. At 2nd order, the size matters

Sometimes simulations are explosive

Two not entirely satisfying options:

1. Reduce shock size
2. Pruning

Usually pruning is preferred. To use in dynare, invoke option `pruning`:

```
stoch_simul( order=3 , irf=100 , periods=0 , pruning );
```

- Uses algorithm of Kim et al. (2008) at 2nd order (Andreasen et al. (2018) at 3rd order)
- Split higher-order approximation into linear and non-linear parts
- Use linear approximation in higher order terms, e.g. for 2nd-order:

$$x_t(1) = g_x x_{t-1}(1) + g_u u_t \tag{13}$$

$$x_t(2) = g_x x_{t-1}(2) + g_u u_t + g_{xx} \left(x_{t-1}(1)\right)^2 + g_{xu} x_{t-1}(1) u_t^2 + g_{uu} u_t^2 \tag{14}$$

Using dynare in loops

You don't need a full dynare call every time you want to solve/simulate

Once you have solved once, you have **M_.**, **oo_.** and **options_.**

To, e.g., update parameters in a loop and re-simulate:

- Set up vector `rho_vec = [0.6 0.7 0.8];`,
- Draw shocks for all simulations `shock_mat=randn(M_.exo_nbr,1000)`
- Loop over iterations `ii=1:length(rho_vec)`:
  1. Update parameter in **M_.**:

     ```
     set_param_value('rho',rho_vec(ii));
     ```

  2. Resolve steady state, resolve model:

     ```
     [dr,~,M,~,oo] = resol(0,M_,options_,oo_);
     ```

  3. Resimulate:

     ```
     y_ = simult_(oo_.steady_state,dr,shock_mat,options_.order);
     ```

References I

Andreasen, M. M., Fernández-Villaverde, J. & Rubio-Ramírez, J. F. (2018), 'The Pruned State-Space System for Non-Linear DSGE Models: Theory and Empirical Applications', *Review of Economic Studies* **85**(1), 1–49.

Kim, J., Kim, S., Schaumburg, E. & Sims, C. A. (2008), 'Calculating and using second-order accurate solutions of discrete time dynamic equilibrium models', *Journal of Economic Dynamics and Control* **32**(11), 3397 – 3414.
URL: *http://www.sciencedirect.com/science/article/pii/S0165188908000316*