Perturbation and Dynare

Introduction to Dynare

Tools for Macroeconomists: The essentials

Petr Sedláček

Dynare

What is Dynare and why use it?

- (free) software for perturbation solutions and more
    - also estimation: ML, Bayesian
    - many options
- you MUST know what it is doing
- once you do, its a very useful tool

Where/how to get Dynare

- download at `www.dynare.org`
- install *and*
- in Matlab set path to `.../Dynare/Matlab`
- read the documentation

What does Dynare do?

Dynare implements a perturbation solution to your model

- model described by $\mathbb{E}_t[F(y_t, x_t)] = \mathbb{E}_t[F(g(x_t), x_t)] = 0$
  - where state variables are denoted by $x_t = [x_{1,t}, ..., x_{n,t}]$
  - and choice variables are denoted by $y_t = g(x_t)$
    - $g(.)$ denote policy rules
- Dynare approximates policy rules, $g(.)$
  - that satisfy first order conditions $\mathbb{E}_t[F(g(x_t), x_t)] = 0$

Result of approximation (e.g. 1st order perturbation)

$$y_t = g(x_t) \approx \bar{y} + (x_t - \bar{x})' a$$

- "bars" indicate steady states
- a coefficient of approximating (Taylor) polynomial

Dynare

NOTATION

Dynare's main file

- main file type is a `*.mod` file
- into this file you specify
  - variables of your model
  - parameters and their values
  - model equations (linearized or not)
  - initial values (ideally steady state)
  - solution method (1st or higher order)
    - many other options (IRFs, simulations, moments etc.)
    - you can also estimate models

Notation in Dynare

- variables known at the beginning of period
- are dated as $t-1$!
  - k_t: capital *choice* in period t
  - k_{t-1}: capital stock available in t

Policy rules

- Dynare produces perturbation approximation to policy rules
- for now consider linear approximations
- linear in what?!
  - Dynare doesn't know that "k" means capital
  - k could be
    - level of capital
    - log of capital
- its up to you to decide
- Dynare will produce policy rules for specified variables

Policy rules

- in neoclassical growth model
- Dynare generates following policy rules

$$k_t = \bar{k} + a_{kk}(k_{t-1} - \bar{k}) + a_{kz}(z_{t-1} - \bar{z}) + a_{k\epsilon}\epsilon_t$$

- i.e. it splits structural shocks into
  - past value and
  - innovation
  - i.e. if $z_t = 1 - \rho + \rho z_{t-1} + \epsilon_t$ then $a_{kz} = \rho a_{k\epsilon}$

Dynare

Blocks

Dynare blocks

A Dynare file has several blocks:

1. list of variables
2. list of exogenous shocks
3. list of model parameters and their values
4. model block (optimality conditions)
5. shock properties
6. initial values
7. solution (and other) commands

Definitions and parametrization

1. Specify variables
    - specified by typing "var" and then listing variables
2. Specify exogenous shocks
    - specified by typing "varexo" and then listing shocks
3. Specify parameters and their values
    - specified by typing "parameters" and then listing parameters
    - each parameter must then be assigned a value
        - either directly in Dynare file
        - or by loading it from outside Dynare file
        - the latter is more convenient for calibration

Model block

4. Model block contains equilibrium conditions
    - initialize block by typing "model;"
    - end it by typing "end;"
    - in between simply write your model equations

Specifics

- Dynare figures out there are expectations when you write $(+1)$
- e.g. the Euler equation:

```
c^(-gamma)=beta*c(+1)^(-gamma)*(alpha*Z(+1)k^(alpha-1)+1-delta)
```

Shock properties

5. Shock properties
    - initialize the block by typing “shocks;”
    - end it by typing “end;”
    - in between specify shock properties
        - e.g. “var e; stderr sigZ;”
        - can specify more, like correlations etc.

6. Initial values
    - initialize block by typing “initval;”
    - end it by typing “end;”
    - inbetween list the initial values of all variables
        - ideally give Dynare the steady state
        - often difficult to compute, so supply it yourself

Solution

7. Give Dynare the green light to solve the model
   - "`stoch_simul(options)`"
   - options include
     - order of perturbation: e.g. "order=1" for linear
     - length of IRFs: e.g. IRF=20
     - many, many more

To actually run Dynare type `dynare filename.mod`

Other useful features

- "resid" command shows equation errors
    - it plugs initial values into model equations
    - they should all be zero in steady state
    - useful for finding out typos

Dynare

Example code

RBC model in Dynare

```
// neoclassical growth model solution and simulation

var c, k, y, z;
varexo e;
parameters alpha, beta, delta, nu, rhoz, sigz, kss, css;

load params;

set_param_value('alpha'     ,par.alpha);    // returns to scale parameter
set_param_value('beta'      ,par.beta);     // discount factor
set_param_value('delta'     ,par.delta);    // depreciation rate
set_param_value('nu'        ,par.nu);       // relative risk aversion coefficient
set_param_value('rhoz'      ,par.rhoz);     // autocorrelation of productivity shock
set_param_value('sigz'      ,par.sigz);     // standard deviation of productivity shock

set_param_value('kss'       ,par.k);        // steady state capital
set_param_value('css'       ,par.c);        // steady state consumption

model;
c^(-nu)  = beta*c(+1)^(-nu)*(alpha*z(+1)*k^(alpha-1) + 1 - delta);
c + k    = y + k(-1)*(1-delta);
y        = z*k(-1)^alpha;
z        = 1 - rhoz + rhoz*z(-1) + e;
end;
```

RBC model in Dynare

```
initval;
k   = kss;
c   = css;
y   = kss^alpha;
z   = 1;
end;

shocks;
var e; stderr sigz;
end;

resid;
steady;

stoch_simul(order=1,nomoments, irf=0, periods = 5000);
```

Dynare's output (coefficients of policy rules)

- remember the quirks of Dynare!

POLICY AND TRANSITION FUNCTIONS

	c	k	y	z
Constant	2.754327	37.989254	3.704059	1.000000
k(-1)	0.044825	0.965276	0.035101	0
z(-1)	0.798702	2.720154	3.518856	0.950000
e	0.840739	2.863320	3.704059	1.000000

Now let's increase the size of shocks (from $\sigma = 0.01$ to $\sigma = 0.1$)

- what happens to the solution?

POLICY AND TRANSITION FUNCTIONS

	c	k	y	z
Constant	2.754327	37.989254	3.704059	1.000000
k(-1)	0.044825	0.965276	0.035101	0
z(-1)	0.798702	2.720154	3.518856	0.950000
e	0.840739	2.863320	3.704059	1.000000

What about a log-linear approximation?

```
var c, k, y, z;
varexo e;
parameters alpha, beta, delta, nu, rhoz, sigz, kss, css;

load params;

set_param_value('alpha'     ,par.alpha);   // returns to scale parameter
set_param_value('beta'      ,par.beta);    // discount factor
set_param_value('delta'     ,par.delta);   // depreciation rate
set_param_value('nu'        ,par.nu);      // relative risk aversion coefficient
set_param_value('rhoz'      ,par.rhoz);    // autocorrelation of productivity shock
set_param_value('sigz'      ,par.sigz);    // standard deviation of productivity shock

set_param_value('kss'       ,par.k);       // steady state capital
set_param_value('css'       ,par.c);       // steady state consumption

model;
c^(-nu) = beta*c(+1)^(-nu)*(alpha*z(+1)*k^(alpha-1) + 1 - delta);
c + k   = y + k(-1)*(1-delta);
y       = z*k(-1)^alpha;
z       = 1 - rhoz + rhoz*z(-1) + e;
end;
```

What about a log-linear approximation?

```
var c, k, y, z;
varexo e;
parameters alpha, beta, delta, nu, rhoz, sigz, kss, css;

load params;

set_param_value('alpha'     ,par.alpha);   // returns to scale parameter
set_param_value('beta'      ,par.beta);    // discount factor
set_param_value('delta'     ,par.delta);   // depreciation rate
set_param_value('nu'        ,par.nu);      // relative risk aversion coefficient
set_param_value('rhoz'      ,par.rhoz);    // autocorrelation of productivity shock
set_param_value('sigz'      ,par.sigz);    // standard deviation of productivity shock

set_param_value('kss'       ,par.k);       // steady state capital
set_param_value('css'       ,par.c);       // steady state consumption

model;
exp(c)^(-nu) = beta* exp(c(+1))^(-nu)*(alpha* exp(z(+1))* exp(k)^(alpha-1) + 1 - delta);
exp(c) + exp(k)   = exp(y) + exp(k(-1))*(1-delta);
exp(y)       = exp(z)* exp(k(-1))^alpha;
exp(z)       = 1 - rhoz + rhoz* exp(z(-1)) + e;
end;
```

What about a log-linear approximation?

Dynare's output (coefficients of policy rules): linear

POLICY AND TRANSITION FUNCTIONS

	c	k	y	z
Constant	2.754327	37.989254	3.704059	1.000000
k(-1)	0.044825	0.965276	0.035101	0
z(-1)	0.798702	2.720154	3.518856	0.950000
e	0.840739	2.863320	3.704059	1.000000

Dynare's output (coefficients of policy rules): log-linear

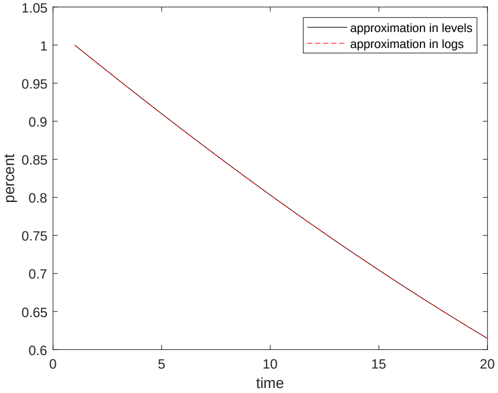
POLICY AND TRANSITION FUNCTIONS

	c	k	y	z
Constant	1.013173	3.637303	1.309429	0
k(-1)	0.618247	0.965276	0.360000	0
z(-1)	0.289981	0.071603	0.950000	0.950000
e	0.305243	0.075372	1.000000	1.000000

What about a log-linear approximation?

Does it matter for the dynamics?

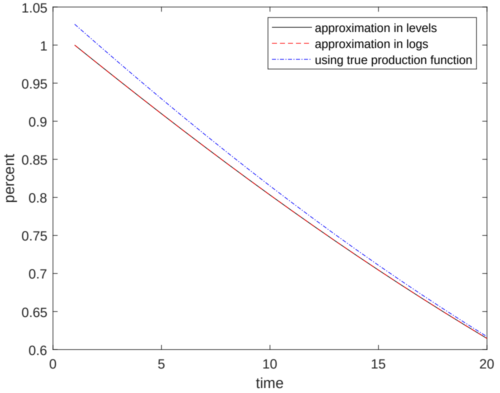
Output impulse response



What about a log-linear approximation?

Does it matter for the dynamics?

Output impulse response



Dynare

Tips and tricks

Incorporating Dynare into a broader code

Often very useful to have the `.mod` file as a part of bigger code

- e.g. when calibrating a model, conducting your own simulation, IRFs etc
- to make this efficient, it requires a few tricks

Tips/tricks

- keeping variables in memory
- loading, instead of setting, parameter values
- saving solution in a separate file
- the idea of homotopy

Keeping variables in memory

As a default, Dynare clears all variables from memory

- to over-ride this, include `noclearall` after your Dynare command
- e.g. `dynare neoclassModel.mod noclearall`

Setting parameter values

In the "parameter block" of the `.mod` file, you need to specify all parameter values

- either you set them directly, e.g. `beta=0.99`
- or you can load parameter values

Loading parameter values

- In a "standard" Matlab program, you can set all your parameter values

```
%% 1. Parametrization
%==========================================================================

par.beta    = 0.99;         % discount factor
par.alpha   = 0.36;         % returns to scale in production
par.delta   = 0.025;        % depreciation rate
par.rhoz    = 0.95;         % autocorrelation of productivity shock
par.sigz    = 0.01;         % standard deviation of productivity shock
par.nu      = 1;            % relative risk aversion coefficient (1=log utility)
```

SETTING PARAMETER VALUES

In the “parameter block” of the `.mod` file, you need to specify all parameter values

- either you set them directly, e.g. `beta=0.99`
- or you can load parameter values

Loading parameter values

- in a “standard” Matlab program, you can set all your parameter values
- then, save all the parameter values as e.g. `save params par`
- in your `.mod` file, load those parameters as `load params`
- finally, set parameters to loaded values using
  `set_param_value('alpha', par.alpha);`

```
set_param_value('alpha'     ,par.alpha);   // returns to scale parameter
set_param_value('beta'      ,par.beta);    // discount factor
set_param_value('delta'     ,par.delta);   // depreciation rate
set_param_value('nu'        ,par.nu);      // relative risk aversion coefficient
set_param_value('rhoz'      ,par.rhoz);    // autocorrelation of productivity shock
set_param_value('sigz'      ,par.sigz);    // standard deviation of productivity shock
```

WOUTER'S `dynarerocks` FILE

All Dynare output is saved in `oo_`

- e.g. IRFs of capital to a productivity shock are in `oo_.irfs.k_e`
- decision rule coefficients are in `oo_.dr.ghx`, in a particular order

Wouter's `disp_dr.m` function

- includes a command that saves decision rules in a matrix in the format you see on screen

POLICY AND TRANSITION FUNCTIONS

	c	k	y	z
Constant	2.754327	37.989254	3.704059	1.000000
k(-1)	0.044825	0.965276	0.035101	0
z(-1)	0.798702	2.720154	3.518856	0.950000
e	0.840739	2.863320	3.704059	1.000000

Wouter's `dynarerocks` file

All Dynare output is saved in `oo_`

- e.g. IRFs of capital to a productivity shock are in `oo_.irfs.k_e`
- decision rule coefficients are in `oo_.dr.ghx`, in a particular order

Wouter's `disp_dr.m` function

- includes a command that saves decision rules in a matrix in the format you see on screen
- matrix is conveniently called `dynarerocks.mat`
- i.e. in order to load decision rules, simply type `load dynarerocks`

Loops and homotopy

All the above is super-useful when calibrating a model

- often, you can solve a model under "some" parametrization
- but getting to your preferred parametrization is harder
  - you might not know what it is
  - you might not have good initial values (steady state)
- in both of the above cases, it is useful to use the homotopy idea
  - move slowly from what you know to where you want to be

Loops and homotopy

Example: suppose you want to solve $[F(x; \alpha_1)] = 0$

- and suppose you know the solution to $[F(x; \alpha_0)] = 0$
- using the solution from $[F(x; \alpha_0)] = 0$ as an initial guess for $[F(x; \alpha_1)] = 0$ may not work!

Instead, solve a sequence of "intermediate" cases $[F(x; \omega\alpha_0 + (1 - \omega)\alpha_1)] = 0$

- where $\omega \in [0, 1]$
- allows for a transition between what you know (α_0) to where you want to be (α_1)
  1. solve model for $\omega_0 = 1$, save x
  2. use x from 1 as initial conditions for case where $\omega_1 < \omega_0$ and save x again
  3. repeat 2 until $\omega_J = 0$

Taking stock

Taking stock

Dynare

- incredibly useful software for perturbation solutions of DSGE models
- can solve, estimate (ML, Bayesian), simulate, produce IRFs etc.
- read documentation for specific syntax


That's all Folks!