DYNARE COURSE
Amsterdam University

Dynare macro–language and dynare++ k-order approximation

Michel Juillard

October 23, 2008

Dynare macro–language

Motivation

- The **Dynare language** (used in MOD files) is well suited for describing economic models
- However, it lacks some useful features, such as:
  - a loop mechanism for automatically repeating similar blocks of equations (such as in multi-country models)
  - an operator for indexed sums or products inside equations
  - a mechanism for splitting large MOD-files in smaller modular files
  - the possibility of conditionally including some equations or some runtime commands
- The **Dynare Macro-language** was specifically designed to address these issues
- Being flexible and fairly general, it can also be helpful in other situations

Design of the macro-language

- The Dynare Macro-language provides a new set of **macro-commands** which can be inserted inside MOD-files
- Language features include:
  - file inclusion
  - loops
  - conditional inclusion (if/then/else structures)
  - expression substitution
- The macro-processor transforms a MOD file with macro-commands into a MOD file without macro-commands (doing text expansions/inclusions) and then feeds it to the Dynare parser
- The key point to understand is that the macro-processor only does **text substitution** (like the C preprocessor or the PHP language)

Macro Directives

- Directives begin with an at-sign followed by a pound sign (@#) and occupy exactly one line
- However, a directive can be continued on next line by adding two anti-slashes (`\\`) at the end of the line to be continued
- A directive produces no output, but serves to give instructions to the macro processor

Inclusion directive

This directive simply includes the content of another file at the place where it is inserted.

Syntax

`@#include "`*filename*`"`

Example

```
@#include "modelcomponent.mod"
```

Note that it is possible to include a file from an included file (nested includes).

Variables

- The macro processor maintains its own list of variables (distinct of model variables and of Matlab variables)
- Variables can be of four types:
  - integer
  - character string (declared between *double* quotes)
  - array of integers
  - array of strings
- No boolean type:
  - false is represented by integer zero
  - true is any non-null integer

Macro-expressions (1/2)

It is possible to construct macro-expressions, using standard operators.

Operators on integers

- arithmetic operators: `+,-,*,/`
- comparison operators: `<,>,<=,>=,==,!=`
- logical operators: `&&,||,!`
- integer ranges: `1:4` is equivalent to integer array `[1,2,3,4]`

Operators on character strings

- comparison operators: `==,!=`
- concatenation: `+`
- extraction of substrings: if `s` is a string, then one can write `s[3]` or `s[4:6]`

Macro-expressions (2/2)

Operators on arrays

- dereferencing: if v is an array, then `v[2]` is its 2nd element
- concatenation: +
- difference `-`: returns the first operand from which the elements of the second operand have been removed
- extraction of sub-arrays: *e.g.* `v[4:6]`

Macro-expressions can be used at two places:

- inside macro directives, directly
- in the body of the MOD-file, between an at-sign and curly braces (like `@{expr}`): the macro processor will substitute the expression with its value

Define directive

The value of a macro-variable can be defined with the `@#define` directive.

Syntax

`@#define` *variable_name* `=` *expression*

Examples

```
@#define x = 5
@#define y = "foo"
@#define v = [ 1, 2, 4 ]
@#define w = [ "foo", "bar" ]
@#define z = 3+v[2]
```

Expression substitution

Dummy example

Before macro-processing

```
@#define x = [ "B", "C" ]
@#define i = 2

model;
  A = @{x[i]};
end;
```

After macro-processing

```
model;
  A = C;
end;
```

Loop directive

Syntax

`@#for` *variable_name* `in` *array_expr*
loop_body
`@#endfor`

Example: before macro-processing

```
model;
@#for country in [ "home", "foreign" ]
  GDP_@{country} = K_@{country}^a * L_@{country}^(1-a)
@#endfor
end;
```

Example: after macro-processing

```
model;
  GDP_home = K_home^a * L_home^(1-a);
  GDP_foreign = K_foreign^a * L_foreign^(1-a);
end;
```

Conditional inclusion directive

Syntax 1

```
@#if integer_expr
    body included if expr != 0
@#endif
```

Syntax 2

```
@#if integer_expr
    body included if expr != 0
@#else
    body included if expr == 0
@#endif
```

Example: alternative monetary policy rules

```
@#define linear_mon_pol = ...
...
model;
@#if linear_mon_pol
  i = w*i(-1) + (1-w)*i_ss + w2*(pie-piestar)
@#else
  i = i(-1)^w * i_ss^(1-w) * (pie/piestar)^w2
@#endif
...
end;
```

Echo and error directives

- The echo directive will simply display a message on standard output
- The error directive will display the message and make Dynare stop (only makes sense inside a conditional inclusion directive)

Syntax

```
@#echo string_expr
@#error string_expr
```

Examples

```
@#echo "Information message."
@#error "Error message!"
```

Saving the macro-expanded MOD file

- For **debugging or learning** purposes, it is possible to save the output of the macro-processor
- This output is a valid MOD-file, obtained after processing the macro-commands of the original MOD-file
- Just add the `savemacro` option on the Dynare command line (after the name of your MOD-file)
- If MOD file is `filename.mod`, then the macro-expanded version will be saved in `filename-macroexp.mod`

Modularization

- The `@#include` directive can be used to split MOD-files into several modular components
- Example setup:
  - `modeldesc.mod`: contains variable declarations, model equations and shocks declarations
  - `simul.mod`: includes `modeldesc.mod`, calibrates parameters and runs stochastic simulations
  - `estim.mod`: includes `modeldesc.mod`, declares priors on parameters and runs bayesian estimation
  - Dynare can be called on `simul.mod` and `estim.mod` (but it makes no sense to run it on `modeldesc.mod`)

Indexed sums or products

Example: moving average

Before macro-processing

```
@#define window = 2

var x MA_x;
...
model;
...
MA_x = 1/@{2*window+1}*(
@#for i in -window:window
        +x(@{i})
@#endfor
       );
...
end;
```

After macro-processing

```
var x MA_x;
...
model;
...
MA_x = 1/5*(
        +x(-2)
        +x(-1)
        +x(0)
        +x(1)
        +x(2)
       );
...
end;
```

Multi-country models

MOD-file skeleton example

```
@#define countries = [ "US", "EU", "AS", "JP", "RC" ]
@#define nth_co = "US"

@#for co in countries
var Y_@{co} K_@{co} L_@{co} i_@{co} E_@{co} ...;
parameters a_@{co} ...;
varexo ...;
@#endfor

model;
@#for co in countries
 Y_@{co} = K_@{co}^a_@{co} * L_@{co}^(1-a_@{co});
...
@# if co != nth_co
 (1+i_@{co}) = (1+i_@{nth_co}) * E_@{co}(+1) / E_@{co}; // UIP relation
@# else
 E_@{co} = 1;
@# endif
@#endfor
end;
```

Endogeneizing parameters (1/3)

- When doing the steady-state calibration of the model, it may be useful to consider a parameter as an endogenous (and vice-versa)
- Example:

$$y = \left(\alpha^{\frac{1}{\xi}}\ell^{1-\frac{1}{\xi}} + (1-\alpha)^{\frac{1}{\xi}}k^{1-\frac{1}{\xi}}\right)^{\frac{\xi}{\xi-1}}$$

$$lab\_rat = \frac{w\ell}{py}$$

- During simulation or estimation, the share parameter α is a parameter, and *lab_rat* is an endogenous variable
- But for steady-state calibration, we may want to impose an economically relevant value for *lab_rat*, and deduce the implied value for α
  $\Rightarrow$ during calibration, α is endogenous and *lab_rat* is a parameter

Endogeneizing parameters (2/3)

- Create `modeqs.mod` with variable declarations and model equations
- For declaration of `alpha` and `lab_rat`:

```
@#if steady
 var alpha;
 parameter lab_rat;
@#else
 parameter alpha;
 var lab_rat;
@#endif
```

- Create `steady.mod`:
  - begins with `@#define steady = 1`
  - then with `@#include "modeqs.mod"`
  - initializes parameters (including `lab_rat`, excluding `alpha`)
  - computes steady state (using hints for endogenous, including `alpha`)
  - saves values of parameters and endogenous at steady-state to a file

Endogeneizing parameters (3/3)

- Create `simul.mod`:
  - begins with `@#define steady = 0`
  - then with `@#include "modeqs.mod"`
  - loads values of parameters and endogenous at steady-state from file
  - computes simulations
- *Note:* functions for saving and loading parameters and endogenous are not yet in Dynare distribution (they should be soon, ask me if you're interested)

Possible future developments

- Find a nicer syntax for indexed sums/products
- Implement other control structures: `elsif`, `switch/case`, `while/until` loops
- Implement macro-functions (or templates), with a syntax like:
  `@define QUADRATIC_COST(x, x_ss, phi) = phi/2*(x/x_ss-1`

Dynare++ and k-order approximation

The general problem

Stochastic case:

$$E_t\left\{f(y_{t+1}, y_t, y_{t-1}, u_t)\right\} = 0$$

y : vector of endogenous variables

u : vector of exogenous stochastic shocks

$$u_t = \sigma\epsilon_t \qquad E(\epsilon_t) = 0$$

$$E\left([\epsilon_t]_{i_1}[\epsilon_t]_{i_2}\dots[\epsilon_t]_{i_k}\right) = \left[\Sigma^{(k)_\epsilon}\right]_{i_1 i_2 \dots i_k}$$

Then,

$$E\left([u_t]_{i_1}[u_t]_{i_2}\dots[u_t]_{i_k}\right) = \sigma^k\left[\Sigma^{(k)}\right]_{i_1 i_2 \dots i_k}$$

The solution function

$$y_t = g\left(y_{t-1}, u_t, \sigma\right)$$

and

$$F\left(y_{t-1}, u_t, u_{t+1}, \sigma\right) = f\left(g\left(g\left(y_{t-1}, u_t, \sigma\right), u_{t+1}, \sigma\right), g\left(y_{t-1}, u_t, \sigma\right), y_{t-1}, u_t\right)$$

or with $s_t = \begin{bmatrix} y_{t-1} \\ u_t \end{bmatrix}$

$$F\left(s_t, u_{t+1}, \sigma\right) = f\left(g\left(g\left(s_t, \sigma\right), u_{t+1}, \sigma\right), g\left(s_t, \sigma\right), s_t\right)$$

k–order approximation of the structural model

$$E_t\left\{F^i\left(s, u', \sigma\right)\right\} = F^i\left(\bar{s}, 0, 0\right) + E_t\left\{\sum_{j=1}^{p} \frac{1}{j!} \sum_{k=0}^{j} \sum_{m=k}^{j} \left[F^i_{s^k u'^{m-k} \sigma^{j-m}}\right]_{\alpha_1 \ldots \alpha_k \beta_1 \ldots \beta_{m-k}} \left[\hat{s}\right]^{\alpha_1} \ldots \left[\hat{s}\right]^{\alpha_k} \left[u'\right]^{\beta_1} \ldots \left[u'\right]^{\beta_{m-k}} \sigma^{j-m}\right\} = 0$$

where $s = s_t$, $\hat{s} = s_t - s_{t-1}$ and $u' = u_{t+1}$.
In tensor notation, the same index used first as subscript and then superscript of two tensors implies summation of the products:

$$\left[A\right]_{\alpha\beta} \left[B\right]^{\alpha} \left[C\right]^{\beta} = \sum_i \sum_j A_{ij} B_i C_j.$$

Reduction of conditional expectation

$$E_t\left\{F^i\left(s_t, u_{t+1}, \sigma\right)\right\} = F^i\left(\bar{s}, 0, 0\right) + \sum_{j=1}^{p} \frac{1}{j!} \sum_{k=0}^{j} \sum_{m=k}^{j} \left[F^i_{s^k u'^{m-k} \sigma^{j-m}}\right]_{\alpha_1 \ldots \alpha_k \beta_1 \ldots \beta_{m-k}} \left[\hat{s}\right]^{\alpha_1} \ldots \left[\hat{s}\right]^{\alpha_k} \sigma^{m-k} \left[\Sigma_\epsilon\right]^{\beta_1 \ldots \beta_{m-k}} \sigma^{j-m} = 0$$

Perturbation approach

For each $j = 1, \ldots, p$, find

$$\left[g_{s^j}\right]$$

such that

$$\left[F^i_{s^j}\right]_{\alpha_1 \ldots \alpha_j} = 0$$

and find

$$\left[g_{s^k \sigma^{j-k}}\right]$$

such that

$$\sum_{m=k}^{j} \left[F^i_{s^k} u'^{m-k} \sigma^{j-m}\right]_{\alpha_1 \ldots \alpha_k} \left[\Sigma_\epsilon\right]^{\beta_1 \ldots \beta_{m-k}} = 0$$

Faà di Bruno formula

Most of what follows depends on the kth order partial derivatives of the composite of two functions, given the partial derivatives of each function.
If $F(r) = f(z(r))$,

$$\left[F^i_{r^j}\right]_{\alpha_1\ldots\alpha_j} = \sum_{l=1}^{j} \left[f^i_{z^l}\right]_{\beta_1\ldots\beta_l} \sum_{c\in\mathcal{M}_{l,j}} \prod_{m=1}^{l} \left[z_{r^{|c_m|}}\right]^{\beta_m}_{\alpha(c_m)}$$

where $\mathcal{M}_{l,j}$ is the set of all partitions of the set of j indices with l classes, $|.|$ is the cardinality of a set, c_m is m-th class of partition c, and $\alpha(c_m)$ is a sequence of α's indexed by c_m. Note that $\mathcal{M}_{1,j} = \{\{1,\ldots,j\}\}$ and $\mathcal{M}_{j,j} = \{\{1\},\{2\},\ldots,\{j\}\}$.
Example for the third partial derivatives

$$\left[F^i_{r^3}\right]_{\alpha_1\alpha_2\alpha 3} = \left[f^i_z\right]_{\beta_1} \left[z_{r^3}\right]^{\beta_1}_{\alpha_1\alpha_2\alpha_3} + \left[f^i_{z^2}\right]_{\beta_1\beta_2} \left(\left[z_r\right]^{\beta_1}_{\alpha_1} \left[z_{r^2}\right]^{\beta_2}_{\alpha_2\alpha_3} + \left[z_r\right]^{\beta_1}_{\alpha_2} \left[z_{r^2}\right]^{\beta_2}_{\alpha_1\alpha_3} + \left[z_r\right]^{\beta_1}_{\alpha_3} \left[z_{r^2}\right]^{\beta_2}_{\alpha_1\alpha_2}\right) + \left[f^i_{z^3}\right]_{\beta_1\beta_2\beta_3} \left[z_r\right]^{\beta_1}_{\alpha_1} \left[z_r\right]^{\beta_2}_{\alpha_2} \left[z_r\right]^{\beta_3}_{\alpha_3}$$

A linear equation that is hard to solve efficiently

$$AX + BXC \otimes C \otimes \ldots \otimes C = D$$

dynare++ uses an efficient algorithm proposed by Ondra Kamenik

Recovering g_{y^j}

Applying twice the Faá di Bruno formula to

$$F\left(y_{t-1}, u_t, u_{t+1}, \sigma\right) = f\left(g\left(g\left(y_{t-1}, u_t, \sigma\right), u_{t+1}, \sigma\right), g\left(y_{t-1}, u_t, \sigma\right), y_{t-1}, u_t\right)$$

and grouping together all terms not involving g_{y^j}, one gets

$$\left[f_{y^+}\right]_\beta \left[g_y\right]^\beta_\gamma \left[g_{y^j}\right]^\gamma_{\alpha_1\ldots\alpha_j} + \left[f_{y^+}\right]_\beta \left[g_{y^j}\right]^\beta_{\gamma_1\ldots\gamma_j} \prod_{m=1}^{j} \left[g_y\right]^{\gamma_m}_{\alpha_m} + \left[f_{y^0}\right]_\beta \left[g_{y^j}\right]^\beta_{\alpha_1\ldots\alpha_j} + \left[B\right]_{\alpha_1\ldots\alpha_j} = 0$$

or, in matrix form

$$f_{y^+} g_y g_{y^j} + f_{y^+} g_{y^j} \left(g_y \otimes g_y \otimes \ldots \otimes g_y\right) + f_{y^0} g_{y^j} + B = 0$$

where the derivatives are unrolled along the rows of matrix g_{y^j}

An asset pricing model

Urban Jermann (1998) "Asset pricing in production economies" *Journal of Monetary Economics*, *41*, 257–275.

- real business cycle model
- consumption habits
- investment adjustment costs
- compares return on several securities
- log–linearizes RBC model + log normal formulas for asset pricing

Simple asset pricing formulas

In a production economy:
Euler equation:

$$Uc_t = \mathcal{E}_t \{\beta Uc_{t+1} r_{t+1}\}$$

Rate of return on firm's capital:

$$r_t = A_t k_{t-1}^{\alpha} n_t^{1-\alpha}$$

Risk free rate:

$$rf_t = \left[\mathcal{E}_t \left\{\beta \frac{Uc_{t+1}}{Uc_t}\right\}\right]^{-1}$$

Expected risk premium:

$$\mathcal{E}_t \{r_{t+1}\} - rf_t = \mathcal{E}_t \{r_{t+1}\} - \left[\mathcal{E}_t \left\{\beta \frac{Uc_{t+1}}{Uc_t}\right\}\right]^{-1}$$

Firms

The representative firm maximizes its value:

$$\mathcal{E}_t \sum_{t+k}^{\infty} \beta^k \frac{\mu_{t+k}}{\mu_t} D_t$$

with

$$\begin{aligned}
Y_t &= A_t K_{t-1}^{\alpha} (X_t N_t)^{1-\alpha} \\
D_t &= Y_t - W_t N t - I_t \\
K_t &= (1-\delta) K_{t-1} + \left(\frac{a_1}{1-\xi}\left(\frac{I_t}{K_{t-1}}\right)^{1-\frac{1}{x}} + a_2\right) K_{t-1} \\
\log A_t &= \rho \log A_{t-1} + e_t \\
X_t &= (1+g) X_{t-1}
\end{aligned}$$

Households

The representative households maximizes current value of future utility:

$$\mathcal{E}_t \sum_{k=0}^{\infty} \beta^k \frac{(C_t - \chi C_{t-1})^{1-\tau}}{1-\tau}$$

subject to the following budget constraint:

$$W_t N_t + D_t = C_t$$

and with $N_t = 1$. Good market equilibrium imposes

$$Y_t = C_t + I_t$$

Interest rate

Risk free interest rate:

$$r_f = \frac{1}{\mathcal{E}_t \left\{\beta g^{-\tau} \frac{\mu_{t+1}}{\mu_t}\right\}}$$

where μ_t is the utility of a marginal unit of consumption in period t.

$$\mu_t = (c_t - \chi c_{t-1}/g)^{-\tau} - \chi\beta (g c_{t+1} - \chi c_t)^{-\tau}$$

Rate of return

Rate of return of firms

$$r_t = \mathcal{E}_t\left\{a_1\left(\frac{gi_t}{k_{t-1}}\right)^{-\frac{1}{\xi}}\left(\alpha z_{t+1}g^{1-\alpha}k_t^{\alpha-1} + \frac{1-\delta+\frac{a_1}{1-\frac{1}{\xi}}\left(\frac{gi_{t+1}}{k_t}\right)^{1-\frac{1}{\xi}}+a_2}{a_1\left(\frac{gi_{t+1}}{k_t}\right)^{-\frac{1}{\xi}}} - \frac{gi_{t+1}}{k_t}\right)\right\}$$

jermann98.mod

```
//-------------------------------------------------------------------
// 1. Variable declaration
//-------------------------------------------------------------------

var c, d, erp1, i, k, m1, r1, rf1, w, y, z, mu;
varexo ez;
```

suite

```
  //-------------------------------------------------------------------
// 2. Parameter declaration and calibration
//-------------------------------------------------------------------

parameters alf, chihab, xi, delt, tau, g, rho, a1, a2, betstar, bet;

alf        = 0.36;    // capital share in production function
chihab     = 0.819;   // habit formation parameter
xi         = 1/4.3;   // capital adjustment cost parameter
delt       = 0.025;   // quarterly deprecition rate
g          = 1.005;   //quarterly growth rate (note zero growth =>g=1)
tau        = 5;       // curvature parameter with respect to c
rho        = 0.95;    // AR(1) parameter for technology shock

a1         = (g-1+delt)^(1/xi);
a2         = (g-1+delt)-(((g-1+delt)^(1/xi))/(1-(1/xi)))*
             ((g-1+delt)^(1-(1/xi)));
betstar    = g/1.011138;
bet        = betstar/(g^(1-tau));
```

suite

```
//-------------------------------------------------------------------
// 3. Model declaration
//-------------------------------------------------------------------

model;
g*k  = (1-delt)*k(-1) + ((a1/(1-1/xi))*(g*i/k(-1))^(1-1/xi)+a2)*k(-1);
d    = y - w - i;
w    = (1-alf)*y;
y    = z*g^(-alf)*k(-1)^alf;
c    = w + d;
mu   = (c-chihab*c(-1)/g)^(-tau)-chihab*bet*(c(+1)*g-chihab*c)^(-tau);
mu   = (betstar/g)*mu(+1)*(a1*(g*i/k(-1))^(-1/xi))*(alf*z(+1)*g^(1-alf)*
       (k^(alf-1))+((1-delt+(a1/(1-1/xi))*(g*i(+1)/k)^(1-1/xi)+a2))/
       (a1*(g*i(+1)/k)^(-1/xi))-g*i(+1)/k);
log(z) = rho*log(z(-1)) + ez;
```

suite

```
m1   = (betstar/g)*mu(+1)/mu;
rf1  = 1/m1;
r1   = (a1*(g*i/k(-1))^(-1/xi))*(alf*z(+1)*g^(1-alf)*(k^(alf-1))+
       (1-delt+(a1/(1-1/xi))*(g*i(+1)/k)^(1-1/xi)+a2)/
       (a1*(g*i(+1)/k)^(-1/xi))-g*i(+1)/k);
erp1 = r1 - rf1;

end;
```

continued

```
initval;
m1     = betstar/g;
rf1    = (1/m1);
r1     = (1/m1);
erp1   = r1-rf1;
z      = 1;
k      = ((((g/betstar)-(1-delt))/(alf*g^(1-alf)))^(1/(alf-1));
y      = (g^(1-alf))*k^alf;
w      = (1-alf)*y;
i      = (1-(1/g)*(1-delt))*k;
d      = y - w - i;
c      = w + d;
mu     = ((c-(chihab*c/g))^(-tau))-chihab*bet*((c*g-chihab*c)^(-tau));
ez     = 0;
end;

vcov = [0.0001];
```

Running dynare++

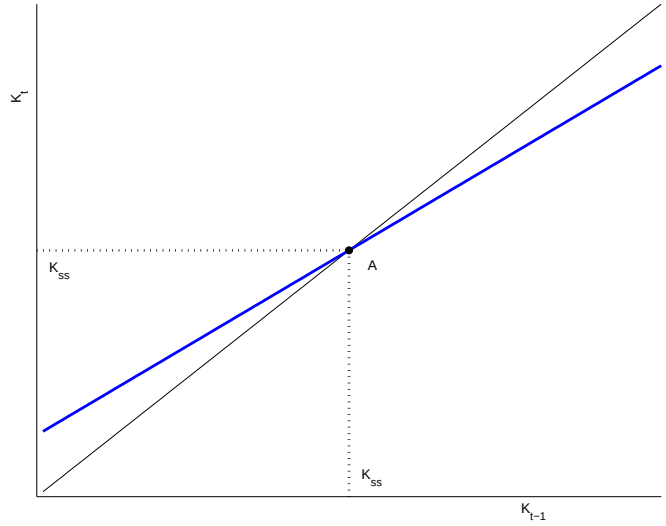
```
dynare++ --order=2 jermann98.mod
```

Three different concepts

1. (deterministic) steady state
2. stochastic steady state
3. unconditional expectation

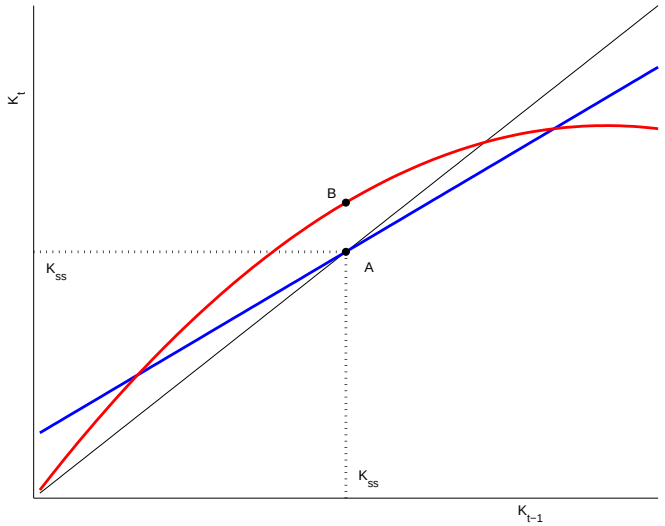
Deterministic steady state

A linearized decision rule cuts the main diagonal at the deterministic steady state (K_{ss}).

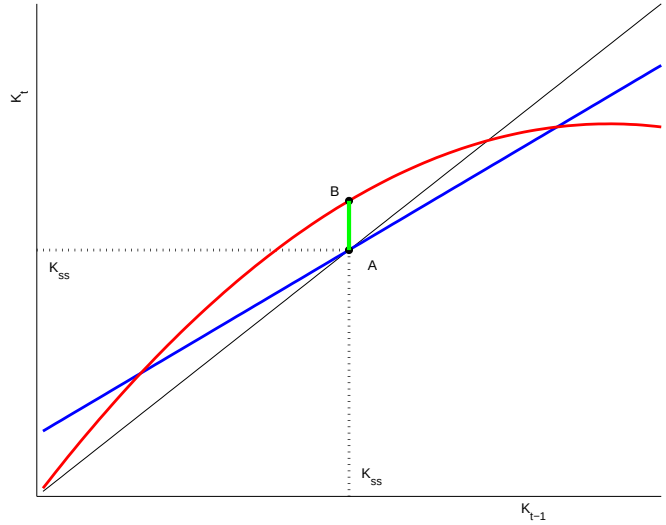


Nonlinear decision rule

In general, the decision is shifted at the deterministic steady state: agents don't decide to stay at the deterministic steady state.

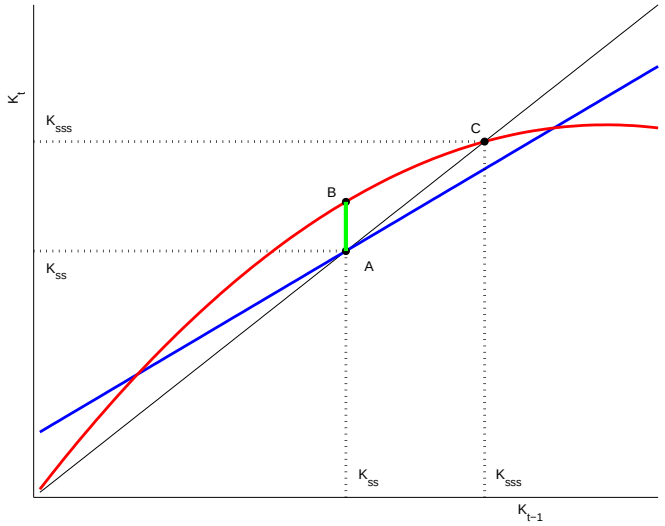


Nonlinear decision rule



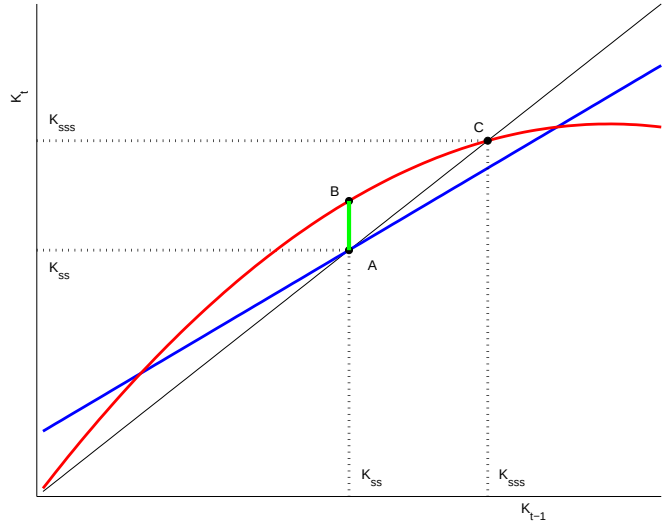
Stochastic steady state

The stochastic steady state, K_{sss}, describe the point where agents decide to stay in absence of shocks this period, but taking into account the distibution of shocks in the future.



Unconditional expectation

Because of Jensen inequality, the unconditional expectation, $E(K)$, is somewhere below the quadratic decision rule, but not on it. In absence of shocks, agents don't decide to go to the unconditional expectation.



Further issues

- Impulse response functions depend of state at time of shocks and history of future shocks.
- For large shocks higher order approximation simulation may explode
  - pruning algorithm (Sims)
  - truncate normal distribution (Judd)

State dependent risk premium

- Compute expected risk premium at different state points
- Compute capital stocks between deterministic steady state plus/minus two standard error (according to first order approximation)
- Warning: this generates unlikely state points as previous consumption remains at steady state value!

premium.m

```
v_k = dyn_vcov(8,8);
k_steadystate = dyn_steady_states(8);
kk = k_steadystate-2*sqrt(v_k):0.5:...
                    k_steadystate+2*sqrt(v_k);
nk = size(kk,2);
for i=1:nk
  k = kk(i);
  start = dyn_steady_states;
  start(8) = k;
  temp = dynare_simul('jermann98',0,start);
  results(i) = temp(2);
end
figure;
plot(kk,results);
```