Intro to Perturbation Approximation and Dynare

Frantisek Masek

Sapienza University of Rome

November 2021

Introduction

Content

- What does it mean to solve a model?
  - Distinction of the state and control variables. Policy function and its approximation. Outline of function approximation.

- Local approximation. How does it work aka what is inside the black box (i.e. Dynare)?
  - Intro to logic behind the perturbation. Concepts of the Taylor approximation and the Implicit Function Theorem.

- Let's set it up in Dynare.
  - Step-by-step description.

- Some quirks and handy extensions.
  - If someone seriously wants to use Dynare.

Original sources

- Let's be honest from the very beginning
  - There is not many innovative thoughts of mine in these slides
  - More or less I just retell awesome presentations and notes from Wouter den Haan, Petr Sedlacek, and Eric Sims in a way I understand the topic

Solving a model

Solving a model

- We have derived a system of non-linear equations. What next?
  - We do not know exact shapes of the resulting functions (indeed, analytical closed-form solutions are ultra-rare). It is necessary to approximate them.
  - Put it differently, we have to solve for so-called policy functions.
- It might be possibly quite a tricky problem.
  - Remember about the general name of this sort of models.
  - Dynamic Stochastic General Equilibrium

What does it mean to solve a model?

We have to find policy rules; i.e., relationships defining how choice variables depend on state variables.

Solving a model

- What are control and state variables? And what are the $\boxed{\text{policy functions?}}$
  - Let me bring up some example. Consider the simplest version of RBC model with the exogenous process for technology and without labour.
  - The model boils down to the following system of non-linear equations.

$$\boxed{C_t{}^{-\eta} = \beta\, \mathbb{E}_t\, C_{t+1}{}^{-\eta} \alpha Z_{t+1} k_{t+1}{}^{\alpha-1}}$$

$$C_t + k_{t+1} = Z_t k_t{}^{\alpha}$$

$$Z_t = (1-\rho) Z + \rho Z_{t-1} + \sigma \epsilon_t,$$

where σ controls the degree of uncertainty.

Examples

Sort out this simple model. Denoting the state variables s, we have: $s_t = [k_t, Z_t]$. When control variables are denoted c, we get: $c_t = [c_t]$.

Solving a model

- Have a look at the system of equations above. What do we know?
  - We surely know Z_t. We also know the initial given value of k. However, we need to choose C_t -> the policy function! Once we obtain C_t, k_{t+1} comes out straight from the budget constraint equation.
- We need to solve out (approximate) for C_t.

Function approximation

Function approximation

- Function approximation, econometrics, and machine learning
  - What do we see if we focus on underlying logic instead of buzzwords?
  - There is actually a lot in common

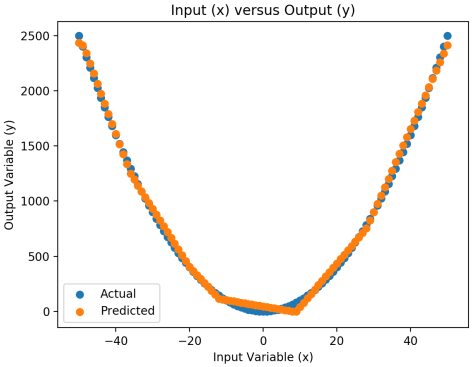


Figure 1: A neural network fitting function $y = x^2$

Function approximation

- Imagine you have a function $y = f(x)$.
  - You need to work with this function, but either you do not know how the function looks like (even though you may have some clue) or you know it, but it is overly complex and unbearable to work with.
- You may know finite set of derivatives of the function at some specific point or possibly finite set of function values.

Function approximation

- With one of the aforementioned info we can approximate our function of interest.
  - Think about polynomials.

$$P_k(x) = \alpha_0 + \alpha_1 * x + \alpha_2 * x^2 + ... + \alpha_k * x^k$$

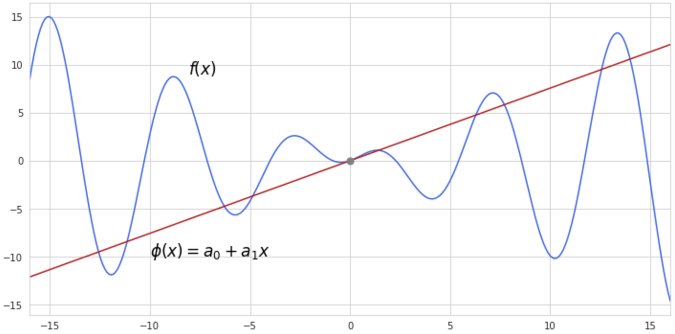


Figure 2: Polynomial approximation of the first degree (linear)

Function approximation

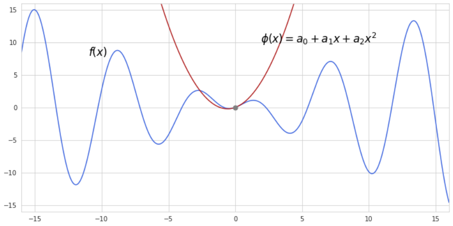


Figure 3: Polynomial approximation of the second degree (quadratic)

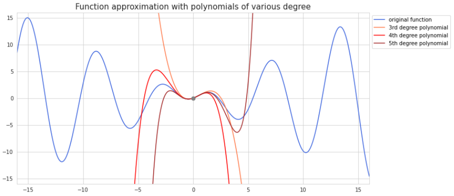


Figure 4: Polynomial approximation of various degrees

Function approximation

- With one of the aforementioned info we can approximate our function of interest.
  - Sometimes it is more suitable to use splines (linear interpolation is the most common example).

$$p_x = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)$$

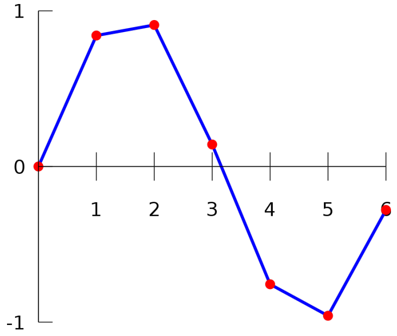


Figure 5: Linear interpolation

Function approximation

- Thus, we know that we can try to approximate unknown functions by polynomials or splines.
    - Formally, we have:

$$y = f(x) \approx \sum_{i=0}^{k} \alpha_i T_i(x),$$

where $T_i(x)$ is a basis function. In our example above, we use regular polynomials $T_i(x) = x^i$.

Function approximation

- Does that sound like kinda *Wingardium Leviosa* stuff to you?
  - It is a sort of magic, isn't it?

- Why did we need to come up with all this?
  - Remember that we need to work with $y = f(x)$, but we do not know its shape. In other words, y is infinite dimensional object.
  - However, once we use $\sum_{i=0}^{k} \alpha_i T_i(x)$ we have only $k + 1$ dimensional object.

Reducing dimensionality

By this procedure, we transferred an infinite dimensional problem into something that can be solved out.

Function approximation

Weierstrass Approximation Theorem

States that any continuous real function defined on a bounded closed interval of real numbers can be approximated uniformly as closely as desired by polynomials.

- A bit of math...
  - Suppose f is a continuous real-valued function defined on the real interval $[a, b]$. For every $\epsilon > 0$ and large enough k, there exists a polynomial $P(x)$ such that for all x in $[a, b]$, we have:

    $$|f(x) - P(x)| < \epsilon$$

  - Or equivalently, the supremum norm in case of $\mathbb{R}^2$

    $$||f - P|| < \epsilon,$$

    where ϵ is discretionary chosen metric.

Function approximation

- What do we need to do practically?
- Define the order of the approximation k
- Find the values of the coefficients α_i
- Adjust k if the approximation is not good enough

- Remember that we have info about x. These are our state variables.

Function approximation

- Find the values of the coefficients α_i
    - By using info that we know about the function.

- Local approximation
    - Knowing a finite set of derivatives (due to the Implicit function theorem), we can apply the Taylor expansion.

- Global approximation
    - Knowing a finite set of points (nodes) x_i and thus their function values $f(x_i)$, one can use projection methods.

Local approximation

- In this course, we are interested mostly in the local approximation.
- Let's dig into perturbation technique. About projection methods maybe next time (if someone is interested).

Perturbation

Perturbation

- What does this weird-looking word actually means?
- To profoundly grasp the idea behind local approximation using perturbation, you need to know two mathematical concepts you all might have encountered during your studies, haven't you?
  - Taylor series expansion
  - Implicit function theorem
- Once you understand them, it is quite simple process

Taylor series expansion

- Consider what Taylor's theorem says:
  - If a real-valued function $f(x)$ is k_{th} times differentiable at the point $a \in \mathbb{R}$, then there exists a function $h_1(x)$ such that:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^k(a)}{k!}(x-a)^k + h_k(x)(x-a)^k,$$

where f^k is k_{th} derivation of f and we assume $\lim_{x \to a} h_1(x) = 0$

  - Then if we denote:

$$P_k(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^k(a)}{k!}(x-a)^k,$$

we can write it down as follows:

$$f(x) = P_k(x) + h_k(x)(x-a)^k,$$

reshuffling it leads towards:

$$h_k(x)(x-a)^k = f(x) - P_k(x),$$

which is error of our approximation. Retrieve that $\lim_{x \to a} h_1(x) = 0$.

Taylor series expansion

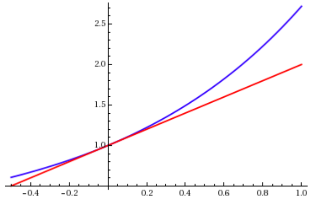


Figure 6: Graph of $f(x) = e^x$ and linear approximation $P_1(x) = 1 + x$ at a $= 0$

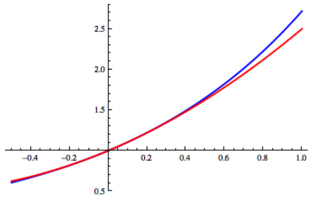


Figure 7: Graph of $f(x) = e^x$ and quadratic approximation $P_2(x) = 1 + x + \frac{x^2}{2}$ at a $= 0$

Taylor series expansion

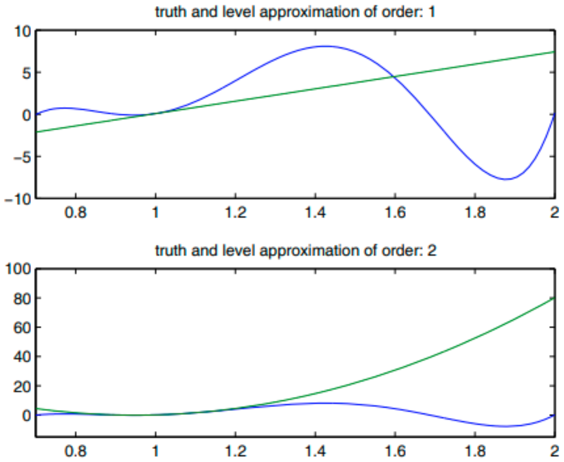


Figure 8: Example of perturbation from notes of Wouter Den Haan

Taylor series expansion

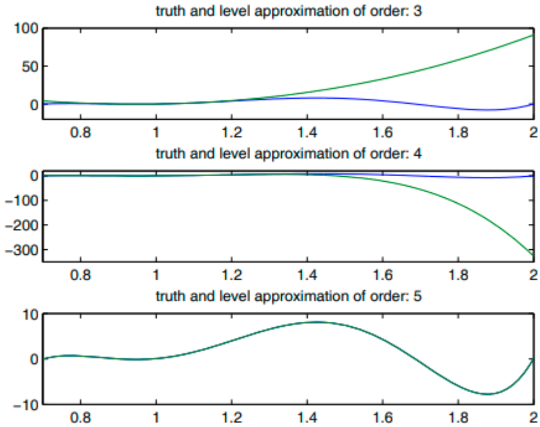


Figure 9: Example of perturbation from notes of Wouter Den Haan

Perturbation

- Let me come back to our simple model example once again
  - All of the model equations may be written in the following way:

$$\mathbb{E}_t\, F[c_{t+1}, c_t, k_{t+1}, Z_{t+1}, k_t, Z_t] = 0$$

  - remember about our notation of state variables $s_t = [k_t, Z_t]$ and control variables $c_t = [c_t]$. Then define general expression for the policy functions and law of motion of the state variables:

$$c_t = g(s_t, \sigma)$$

$$s_{t+1} = h(s_t, \sigma) + \sigma \epsilon_{t+1}$$

- Can you recognize which formula holds for the given equations of our simple model?

Perturbation

- Using the general notation, let's rewrite everything:

$$\mathbb{E}_t\, F[g(s_{t+1},\sigma), g(s_t,\sigma), s_{t+1}, s_t] = 0$$

- substituting out what we have defined:

$$\mathbb{E}_t\, F\left\{g[h(s_t,\sigma)+\sigma\epsilon_{t+1},\sigma], g(s_t,\sigma), h(s_t,\sigma)+\sigma\epsilon_{t+1}, s_t\right\} = 0$$

- As you might start to suspect, the toughest thing about perturbation is notation

Perturbation

- Thus, I always prefer to imagine everything at specific examples. In our case, we work with the Euler equation:

$$C_t^{-\eta} = \beta \, \mathbb{E}_t \, C_{t+1}^{-\eta} \alpha Z_{t+1} k_{t+1}^{\alpha-1}$$

- Let me follow the aforementioned cookbook:

$$C_t^{-\eta} - \beta \, \mathbb{E}_t \, C_{t+1}^{-\eta} \alpha Z_{t+1} k_{t+1}^{\alpha-1} = 0$$

$$g(k_t, Z_t, \sigma)^{-\eta} - \beta \, \mathbb{E}_t \, g[h(k_t, Z_t \sigma) + \sigma \epsilon_{t+1}, \sigma]^{-\eta} \alpha [(1-\rho)Z + \rho Z_t + \sigma \epsilon_{t+1}]$$
$$\left\{ [\ (1\text{ - }\rho)Z + \rho Z_{t-1} + \sigma \epsilon_t] k_t^{\alpha} - g(k_t, Z_t, \sigma) \right\}^{\alpha-1} = 0$$

- Okay, I have to concede that this is not necessarily clarifying at first glance. Anyway, try to have a closer look at it if you want to really understand

Perturbation

- Once we have the system written as:

$$\mathbb{E}_t\, F\left\{ g[h(s_t,\sigma)+\sigma\epsilon_{t+1},\sigma], g(s_t,\sigma), h(s_t,\sigma)+\sigma\epsilon_{t+1}, s_t \right\} = 0$$

- Let's perturbate it aka find the local approximation of g and f
  - Recall the Taylor expansion and approximate it around certain point $(\overline{s},\overline{\sigma})$. In terms of the general example, $a=(\overline{s},\overline{\sigma})$

Perturbation

- Let's perturbate the system aka find the local approximation of g and f

$$g(s,\sigma) \approx g(\overline{s},\overline{\sigma}) + g_s(\overline{s},\overline{\sigma})(s-\overline{s}) + g_\sigma(\overline{s},\overline{\sigma})(\sigma-\overline{\sigma})$$

$$+ 1/2 \left[\mathrm{g}_{ss}(\overline{s},\overline{\sigma})(s-\overline{s})^2 + g_{\sigma\sigma}(\overline{s},\overline{\sigma})(\sigma-\overline{\sigma})^2 + g_{s\sigma}(\overline{s},\overline{\sigma})(s-\overline{s})(\sigma-\overline{\sigma}) \right] + \ldots$$

$$h(s,\sigma) \approx h(\overline{s},\overline{\sigma}) + h_s(\overline{s},\overline{\sigma})(s-\overline{s}) + h_\sigma(\overline{s},\overline{\sigma})(\sigma-\overline{\sigma})$$

$$+ 1/2 \left[\mathrm{h}_{ss}(\overline{s},\overline{\sigma})(s-\overline{s})^2 + h_{\sigma\sigma}(\overline{s},\overline{\sigma})(\sigma-\overline{\sigma})^2 + h_{s\sigma}(\overline{s},\overline{\sigma})(s-\overline{s})(\sigma-\overline{\sigma}) \right] + \ldots$$

- There are just two question left that we need to figure out
  - What are $\overline{s}$ and $\overline{\sigma}$?
  - How to get the coefficients/derivatives?

Perturbation

- What are $\overline{s}$ and $\overline{\sigma}$?
  - We have to approximate around some certain point. Thus, we need some point that we know
  - Suitable candidate is the non-stochastic steady state
  - In such a case, we have $s_t = \overline{s}$ and $\sigma = 0$
  - Which gives us $\overline{c} = g(\overline{s}, 0)$ and $\overline{s} = h(\overline{s}, 0)$

Perturbation

- Hence, if we for the sake of simplicity consider only the first order perturbation (aka log-linearization), we have:

$$g(s,\sigma) \approx g(\overline{s},0) + g_s(\overline{s},0)(s-\overline{s}) + g_\sigma(\overline{s},0)(\sigma)$$

$$h(s,\sigma) \approx h(\overline{s},0) + h_s(\overline{s},0)(s-\overline{s}) + h_\sigma(\overline{s},0)(\sigma),$$

  - where we know that $\overline{c} = g(\overline{s},0)$ and $\overline{s} = h(\overline{s},0)$
- Now only one thing remains to handle out - the derivatives of the policy function

Perturbation

- How to get the derivatives of the policy function?
  - It is time for the second introduced concept - the Implicit function theorem

Implicit function theorem

- Imagine you work with the following expression:

$$x^2 + y^2 = 5^2$$

  - How to find out what is the point A?

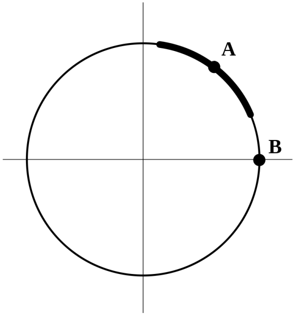


Figure 10: Example of implicit circle

Implicit function theorem

- If we had usual expression $y = f(x)$, we know what to do. Just take a derivative $\frac{\partial f(x)}{\partial x}$
- However, in our situation we do not have such a defined relationship
    - Notice that for each x, you can have two values of y that satisfy the equality and vice versa
    - That's to say we work with the implicit relationship
- To get the value of A, we have to proceed a slightly differently

Implicit function theorem

- To get the value of A, we have to proceed a slightly differently
  - Basically, we need to differentiate the whole expression and reshuffle it

$$2xdx + 2ydy = 0$$

$$\frac{dy}{dx} = -\frac{x}{y}$$

- This is the general formula for the implicit function derivation
  - For a more profound look, check out this great video

Here we go

- Now we know everything needed to perform the perturbation approximation

Perturbation technique

We use polynomial approximation around a certain point in form of the Taylor expansion and gain the derivatives using the Implicit function theorem.

Okay, it's not everything

- Actually there is slightly more to do...
- Once we get a linearized expression of our model (i.e. linear stochastic difference system of equations), we need to reshuffle its state-space representation in a way that enables us to show all of the control and the state variables only as linear functions of previous values of the endogenous state variables and exogenous shocks
  - To put it more elegantly. All variables of our system (the endogenous state variable, the forward-looking control variables, and the static control variables) are expressed as linear functions of the backward-looking, predetermined, state variables

Solution strategies

- There are different method that help us to do this.
  - By far the most famous one is an eigenvalue-eigenvector decomposition of Blanchard and Kahn (1980). This method allows us to study the determinacy properties of our model. The resulting condition is that the number of eigenvalues outside the unit circle (i.e. the unstable eigenvectors) has to be equal to the number of forward-looking variables to have a unique solution of a model
  - Sims (2002) in his approach studies a situation when it is not clear to distinguish between control and state variables. He also works with expectation errors rather than with expectation operators
  - blueKlein (2000) generalizes the approach of Blanchard and Kahn (1980) even for the case of non-invertible A (i.e. when a system is singular). While Blanchard and Kahn (1980) use a Jordan decomposition in its canonical form, Klein (2000) works with a Schur decomposition.
  - Differently, Uhlig (1999) uses the method of undetermined coefficients

Solution strategies

- Anyway, this presentation is not devoted to the solution methods. Right now we just need to know two things. Firstly, at the end of the day all of the strategies have to generate the same results. Second, it is not a complicated, yet super tedious algebra exercise to derive the aforementioned solutions (okay we might not need to know this)
- Dynare uses Klein (2000) to print out the policy rules! (this is the second thing we may find of our interest)

Dynare

How to handle Dynare

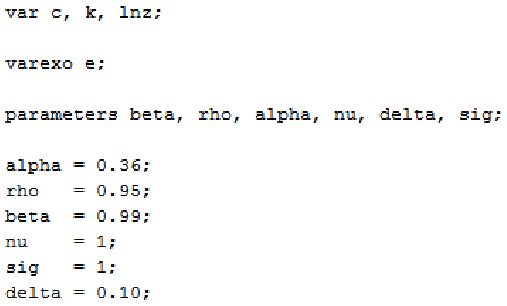
- First of all, you have to download and install Dynare
  - Visit the webpage of Dynare and download it
  - Go to your Matlab and set path to a folder called *"matlab"* that you find in the one you chose during the installation
  - Usually, you just type *"C:\dynare \4.6.4 \matlab"*
  - From now on, you should be able to use it
  - If you get lost, just follow this video

How to handle Dynare

- Dynare file needs to be saved as *yourmodel.mod*
- The structure of the file is following
  - endogenous variables
    - be careful about timing because predetermined variables (in our case capital) has to be written as $t-1$ if it is t in a model and as t if it appears with $t+1$
  - exogenous variables
  - parameters and their calibration
  - model equations (can be both non-linear and linear)
  - initial values for computing the steady state
    - alternatively, you can create a steady state file to solve for the steady state by your own
  - specification of the shock(s) variance
  - solution method (1st order or higher perturbation)
    - we can print out many things like IRFs, moments, simulations, etc

How to code up a simple model

- Firstly, stick with our simple RBC model

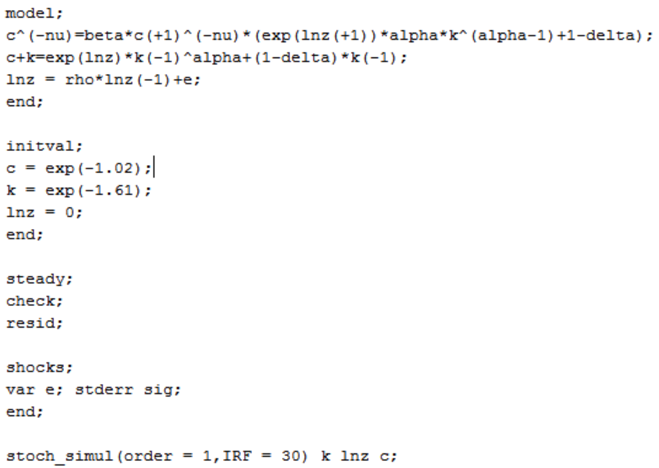


```
var c, k, lnz;

varexo e;

parameters beta, rho, alpha, nu, delta, sig;

alpha = 0.36;
rho   = 0.95;
beta  = 0.99;
nu    = 1;
sig   = 1;
delta = 0.10;
```

Figure 11: Dynare code of a simple RBC model without labour

How to code up a simple model

- Firstly, stick with our simple RBC model

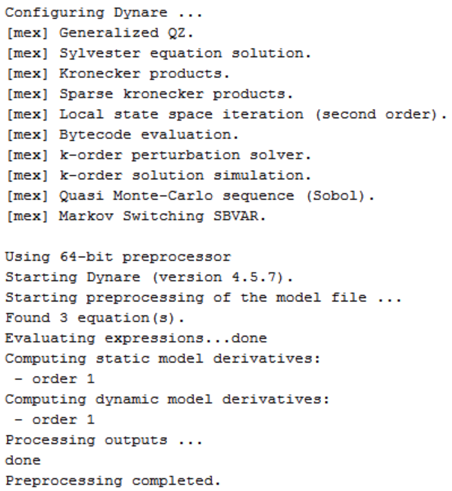


```
model;
c^(-nu)=beta*c(+1)^(-nu)*(exp(lnz(+1))*alpha*k^(alpha-1)+1-delta);
c+k=exp(lnz)*k(-1)^alpha+(1-delta)*k(-1);
lnz = rho*lnz(-1)+e;
end;

initval;
c = exp(-1.02);
k = exp(-1.61);
lnz = 0;
end;

steady;
check;
resid;

shocks;
var e; stderr sig;
end;

stoch_simul(order = 1,IRF = 30) k lnz c;
```

Figure 12: Dynare code of a simple RBC model without labour

Dynare output

- Firstly, stick with our simple RBC model

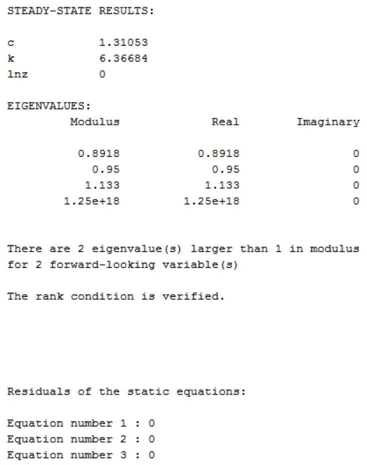


```
Configuring Dynare ...
[mex] Generalized QZ.
[mex] Sylvester equation solution.
[mex] Kronecker products.
[mex] Sparse kronecker products.
[mex] Local state space iteration (second order).
[mex] Bytecode evaluation.
[mex] k-order perturbation solver.
[mex] k-order solution simulation.
[mex] Quasi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

Using 64-bit preprocessor
Starting Dynare (version 4.5.7).
Starting preprocessing of the model file ...
Found 3 equation(s).
Evaluating expressions...done
Computing static model derivatives:
 - order 1
Computing dynamic model derivatives:
 - order 1
Processing outputs ...
done
Preprocessing completed.
```

Figure 13: Dynare output of a simple RBC model without labour

Dynare output

- Firstly, stick with our simple RBC model

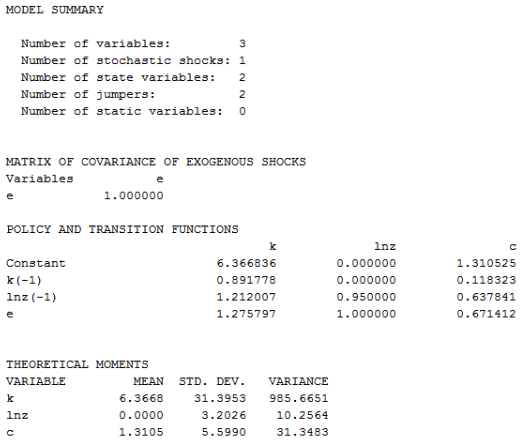


```
STEADY-STATE RESULTS:

c             1.31053
k             6.36684
lnz           0

EIGENVALUES:
         Modulus             Real        Imaginary

          0.8918           0.8918                0
            0.95             0.95                0
           1.133            1.133                0
        1.25e+18         1.25e+18                0


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

The rank condition is verified.




Residuals of the static equations:

Equation number 1 : 0
Equation number 2 : 0
Equation number 3 : 0
```

Figure 14: Dynare output of a simple RBC model without labour

Dynare output

- Firstly, stick with our simple RBC model

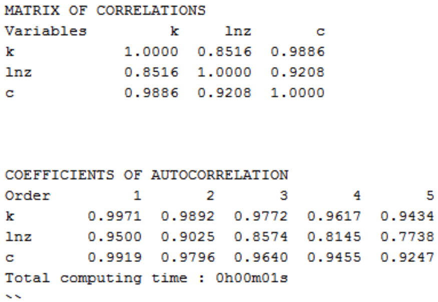


```
MODEL SUMMARY

  Number of variables:         3
  Number of stochastic shocks: 1
  Number of state variables:   2
  Number of jumpers:           2
  Number of static variables:  0


MATRIX OF COVARIANCE OF EXOGENOUS SHOCKS
Variables           e
e            1.000000

POLICY AND TRANSITION FUNCTIONS
                                   k          lnz            c
Constant                    6.366836     0.000000     1.310525
k(-1)                       0.891778     0.000000     0.118323
lnz(-1)                     1.212007     0.950000     0.637841
e                           1.275797     1.000000     0.671412


THEORETICAL MOMENTS
VARIABLE         MEAN  STD. DEV.   VARIANCE
k              6.3668    31.3953   985.6651
lnz            0.0000     3.2026    10.2564
c              1.3105     5.5990    31.3483
```

Figure 15: Dynare output of a simple RBC model without labour

Dynare output

- Firstly, stick with our simple RBC model



```
MATRIX OF CORRELATIONS
Variables          k     lnz       c
k             1.0000  0.8516  0.9886
lnz           0.8516  1.0000  0.9208
c             0.9886  0.9208  1.0000



COEFFICIENTS OF AUTOCORRELATION
Order           1       2       3       4       5
k          0.9971  0.9892  0.9772  0.9617  0.9434
lnz        0.9500  0.9025  0.8574  0.8145  0.7738
c          0.9919  0.9796  0.9640  0.9455  0.9247
Total computing time : 0h00m01s
>>
```

Figure 16: Dynare output of a simple RBC model without labour

Dynare output

- Firstly, stick with our simple RBC model

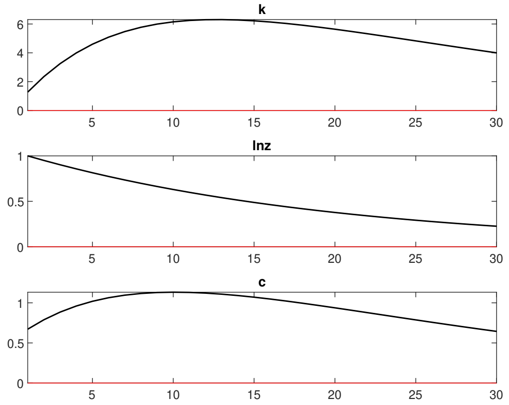
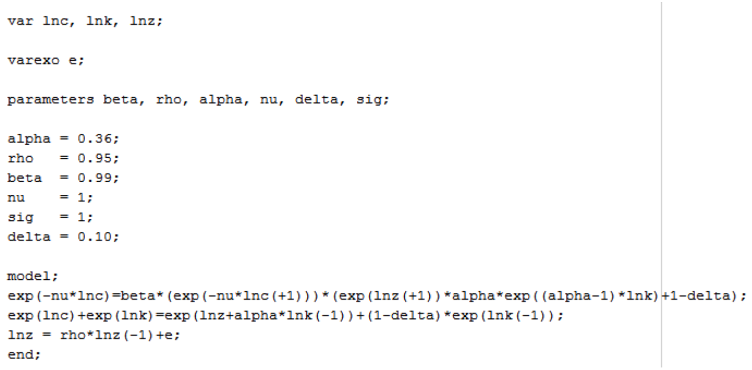


Figure 17: Dynare IRFs of a simple RBC model without labour

Dynare output

- What if we want Dynare to log-linearize the model for us?

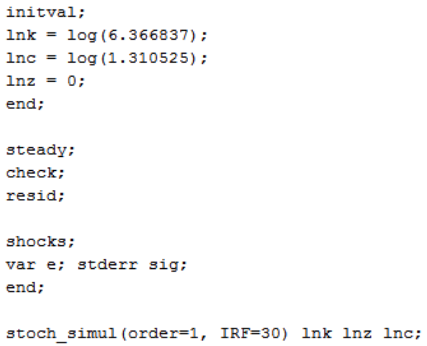


```
var lnc, lnk, lnz;

varexo e;

parameters beta, rho, alpha, nu, delta, sig;

alpha = 0.36;
rho   = 0.95;
beta  = 0.99;
nu    = 1;
sig   = 1;
delta = 0.10;

model;
exp(-nu*lnc)=beta*(exp(-nu*lnc(+1)))*(exp(lnz(+1))*alpha*exp((alpha-1)*lnk)+1-delta);
exp(lnc)+exp(lnk)=exp(lnz+alpha*lnk(-1))+(1-delta)*exp(lnk(-1));
lnz = rho*lnz(-1)+e;
end;
```

Figure 18: Dynare IRFs of a simple log-linearized RBC model

Dynare output

- What if we want Dynare to log-linearize the model for us?



```
initval;
lnk = log(6.366837);
lnc = log(1.310525);
lnz = 0;
end;

steady;
check;
resid;

shocks;
var e; stderr sig;
end;

stoch_simul(order=1, IRF=30) lnk lnz lnc;
```

Figure 19: Dynare IRFs of a simple log-linearized RBC model

Dynare output

- What if we want Dynare to log-linearize the model for us?

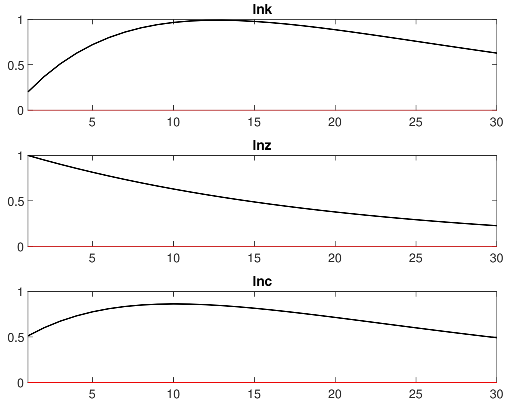
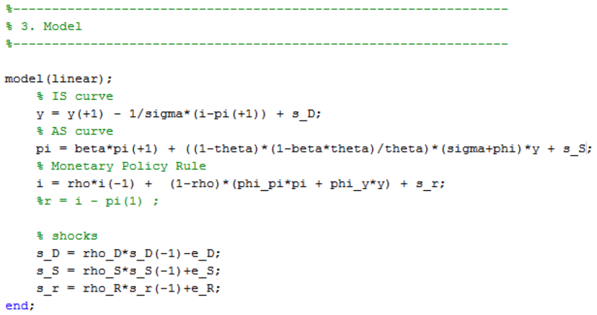


Figure 20: Dynare IRFs of a simple log-linearized RBC model

Dynare output

- If we have already log-linearized our model by hand, we can use *"model(linear)"* so that we do not need to compute the steady state anymore (always zeros) and this can speed some things up
  - Note that log-linearization is equivalent to the first order perturbation approximation after taking logs



```
%----------------------------------------------------------------
% 3. Model
%----------------------------------------------------------------

model(linear);
    % IS curve
    y = y(+1) - 1/sigma*(i-pi(+1)) + s_D;
    % AS curve
    pi = beta*pi(+1) + ((1-theta)*(1-beta*theta)/theta)*(sigma+phi)*y + s_S;
    % Monetary Policy Rule
    i = rho*i(-1) +  (1-rho)*(phi_pi*pi + phi_y*y) + s_r;
    %r = i - pi(1) ;

    % shocks
    s_D = rho_D*s_D(-1)-e_D;
    s_S = rho_S*s_S(-1)+e_S;
    s_r = rho_R*s_r(-1)+e_R;
end;
```

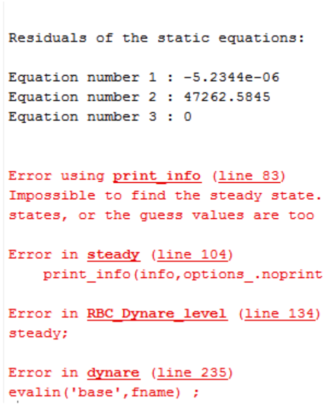
Figure 21: Using linear model

Handy tips

- Use *"resid"*. It helps you out in case of mistakes
  - You should have all equations zeros in the steady state. If not, there is a mistake
- It might be better to define the parameter values in a Matlab file rather than in Dynare
  - If you need to loop over different values of parameters
- Good to know how to draw the policy functions
  - Sometimes you can use Dynare just as a tool within larger Matlab file
    - In this case, use command *"noclearall"*. Otherwise, Dynare will automatically erase all previous output
- Different ways of coping with the steady state
  - If you can, rather derive it by hand

Handy tips

- Use *"resid"*. It helps you out in case of mistakes

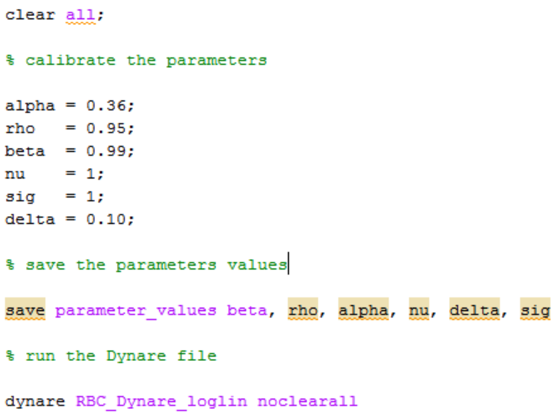


```
Residuals of the static equations:

Equation number 1 : -5.2344e-06
Equation number 2 : 47262.5845
Equation number 3 : 0


Error using print_info (line 83)
Impossible to find the steady state.
states, or the guess values are too

Error in steady (line 104)
    print_info(info,options_.noprint

Error in RBC_Dynare_level (line 134)
steady;

Error in dynare (line 235)
evalin('base',fname) ;
```

Figure 22: Using *"resid"* for detecting mistakes

Handy tips

- It might be better to define the parameter values in a Matlab file rather than in Dynare

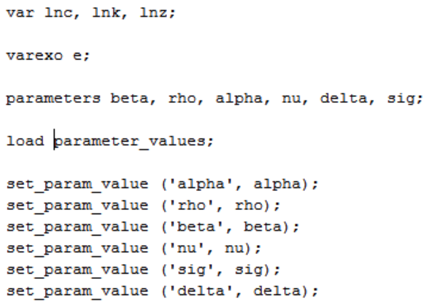


```
clear all;

% calibrate the parameters

alpha = 0.36;
rho   = 0.95;
beta  = 0.99;
nu    = 1;
sig   = 1;
delta = 0.10;

% save the parameters values

save parameter_values beta, rho, alpha, nu, delta, sig

% run the Dynare file

dynare RBC_Dynare_loglin noclearall
```

Figure 23: Calibrating the model in a separate Matlab file

Handy tips

- It might be better to define the parameter values in a Matlab file rather than in Dynare

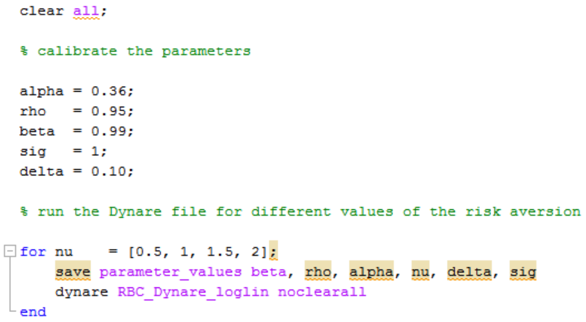


```
var lnc, lnk, lnz;

varexo e;

parameters beta, rho, alpha, nu, delta, sig;

load parameter_values;

set_param_value ('alpha', alpha);
set_param_value ('rho', rho);
set_param_value ('beta', beta);
set_param_value ('nu', nu);
set_param_value ('sig', sig);
set_param_value ('delta', delta);
```

Figure 24: Calibrating the model in a separate Matlab file

Handy tips

- It might be better to define the parameter values in a Matlab file rather than in Dynare
    - If you need to loop over different values of parameters



```
clear all;

% calibrate the parameters

alpha = 0.36;
rho   = 0.95;
beta  = 0.99;
sig   = 1;
delta = 0.10;

% run the Dynare file for different values of the risk aversion

for nu    = [0.5, 1, 1.5, 2];
    save parameter_values beta, rho, alpha, nu, delta, sig
    dynare RBC_Dynare_loglin noclearall
end
```

Figure 25: Looping across more values of a parameter

- You get four outputs, each for a specific value

Handy tips

- Good to know how to draw the policy functions
  - You have to know where Dynare stores outputs
  - Let's clarify what Dynare does in a state-space form
- You can rewrite your system of equations as follows:

$$S_t = AS_{t-1} + B\epsilon_t$$

$$C_t = \Phi S_t,$$

  - where S_t is $m \times 1$ vector of state variables, C_t is $n \times 1$ vector of control variables, and ϵ_t is $w \times 1$ vector of shocks. Thus, A is $m \times m$ matrix, B is $m \times w$ matrix, and Φ is $n \times m$ matrix of which we can think as the policy function

Handy tips

- Dynare substitutes out the law of motion for the vector of state variables. Hence, we have:

$$C_t = \Phi(AS_{t-1} + B\epsilon_t)$$

- To simplify a little the notation, let me to write $\Phi A = C$ $(n \times m)$ and $\Phi B = D$ $(n \times w)$. This allows us to write the whole system as:

$$C_t = CS_{t-1} + D\epsilon_t$$

$$S_t = AS_{t-1} + B\epsilon_t$$

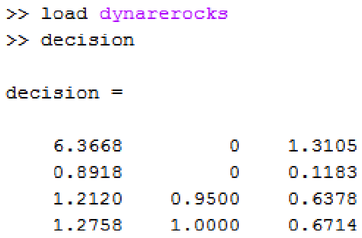
Handy tips

- From now on you should be well-equipped to draw data from Dynare once you need to do it. Everything important is hidden in *"oo_."*
  - Starting with simple thing, you can find the IRFs of the shock e on variable k under *"oo_.irfs.k_e"*. For this case, you even do not need to type it in complete, *"k_e"* should be enough to print it out
  - Coefficients of matrices C and A are stored under *"oo_.dr.ghx"* while the ones in D and B under *"oo_.dr.ghu"*
    - Unfortunately, the ordering is quite messy. The "DR" ordering (for "decision rule") is ordered as follows: static variables (t), backward-looking variables (t and $t-1$), mixed variables ($t-1$, t, and $t+1$), and forward-looking variables (t and $t+1$)
    - You can circumvent this by typing *"oo_.dr.inv_order_var(2)"* inside of *"oo_.dr.ghx"* or *"oo_.dr.ghu"*. Then you get the coefficient of the second variable. So you have *"oo_.dr.ghx(oo_.dr.inv_order_var(2))"*
  - The steady state values are hidden under *"oo_dr.ys"*

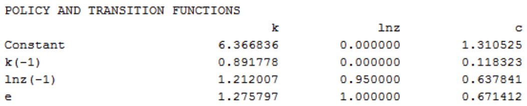
Handy tips

- If you need to pull out the policy functions, you can draw the coefficients from the matrices in the aforementioned way
  - But that is super inconvenient, isn't it?
- Fortunately, there is better option thanks to Wouter Den Haan
  - Go to his website and download *"disp_dr.m"*. Then save it in your folder instead of the file with an identical name
  - By typing *"load dynarerocks"*, you load the policy rules into a matrix called *"decision"*

Handy tips



```
>> load dynarerocks
>> decision

decision =

    6.3668         0    1.3105
    0.8918         0    0.1183
    1.2120    0.9500    0.6378
    1.2758    1.0000    0.6714
```



```
POLICY AND TRANSITION FUNCTIONS
                                      k          lnz            c
Constant                       6.366836     0.000000     1.310525
k(-1)                          0.891778     0.000000     0.118323
lnz(-1)                        1.212007     0.950000     0.637841
e                              1.275797     1.000000     0.671412
```

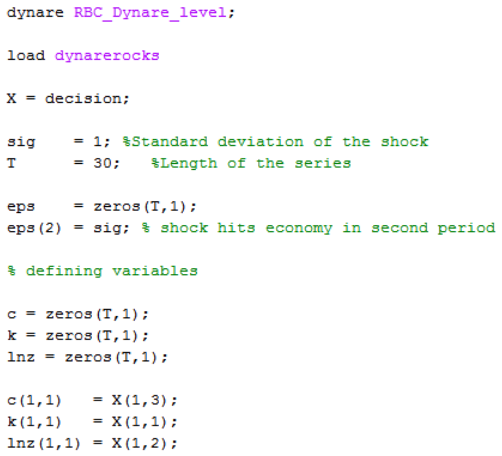
Figure 26: Drawing policy functions by Wouter's *"dynarerocks"*

Handy tips

- Different ways of coping with the steady state
  - Usual way is to provide to Dynare initial values and let it to compute the steady state on its own
  - But sometimes it can be superior to derive the steady state by hand and write it down to separated Matlab file that Dynare then just reads. It is simple, look at notes of E. Sims
    - E.g. if you want to change values of parameters and do not always change also values of initial guesses for the steady state
  - Sometimes you might find useful to use homotophy. If so, look at slides of Wouter Den Haan

Handy tips

- Construct the IRFs on your own

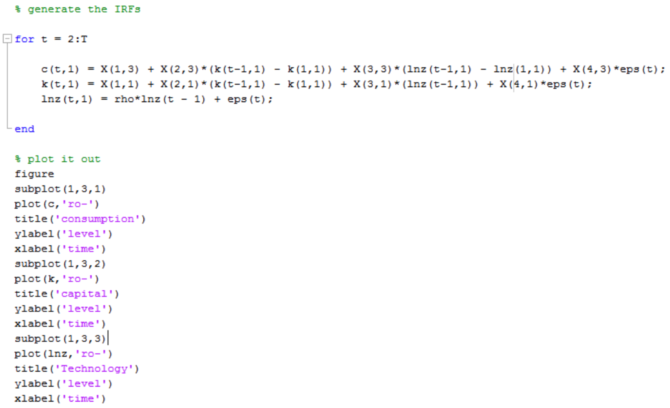


```
dynare RBC_Dynare_level;

load dynarerocks

X = decision;

sig    = 1; %Standard deviation of the shock
T      = 30;   %Length of the series

eps    = zeros(T,1);
eps(2) = sig; % shock hits economy in second period

% defining variables

c = zeros(T,1);
k = zeros(T,1);
lnz = zeros(T,1);

c(1,1)   = X(1,3);
k(1,1)   = X(1,1);
lnz(1,1) = X(1,2);
```

Figure 27: Making the IRFs myself

Handy tips

- Construct the IRFs on your own



```
% generate the IRFs

for t = 2:T

    c(t,1) = X(1,3) + X(2,3)*(k(t-1,1) - k(1,1)) + X(3,3)*(lnz(t-1,1) - lnz(1,1)) + X(4,3)*eps(t);
    k(t,1) = X(1,1) + X(2,1)*(k(t-1,1) - k(1,1)) + X(3,1)*(lnz(t-1,1)) + X(4,1)*eps(t);
    lnz(t,1) = rho*lnz(t - 1) + eps(t);

end

% plot it out
figure
subplot(1,3,1)
plot(c,'ro-')
title('consumption')
ylabel('level')
xlabel('time')
subplot(1,3,2)
plot(k,'ro-')
title('capital')
ylabel('level')
xlabel('time')
subplot(1,3,3)
plot(lnz,'ro-')
title('Technology')
ylabel('level')
xlabel('time')
```

Figure 28: Making the IRFs myself

Handy tips

- Construct the IRFs on your own

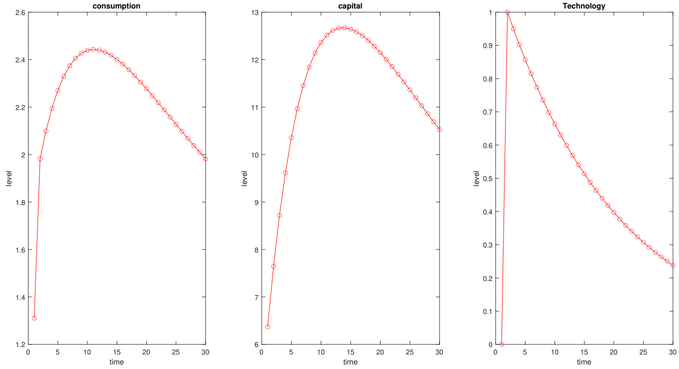


Figure 29: Making the IRFs myself

Handy tips - extensions

- Determinacy analysis
- Optimal monetary policy - welfare loss function
- A zero lower bound using piecewise linear/regime-switching perturbation (Guerrieri and Iacoviello, 2015; Eggertsson and Woodford, 2003; Eggertsson et al., 2021) or news shock method of Holden (2016, 2019)
  - The first and the last ones can be applied in Dynare, the others outside of it, yet it is still the perturbation technique
  - If feasible up to number of state variables, solve it out non-linearly using global techniques
- Estimation of parameters (GMM, Bayesian estimation)
- Bunch of other things
- Do not hesitate to drop me an email if you want to help out with something or to ask about some extensions
  - **frantisek.masek@uniroma1.it**