Global Macroeconomics
Solving Quantitative Macroeconomic Models

MS.c. in International Economics

Valerio Dionisi

Department of Economics, Management and Statistics
University of Milano-Bicocca

Lecture 1

Recursive Methods and Introduction to `dynare`

April 29, 2024

UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA
AUDENTES FORTUNA IUVAT

Overview

"solving a standard quantitative macroecnomic model is just a big rootfinding problem"

- finding the *root* of a function means to find x such that $f(x) = 0$

$\longleftrightarrow$ a typical problem is $f(x) = g(x)$, so that the *root* is simply $f(x) - g(x) = h(x)$

- economic problems are:
  ① ***univariate***, solution to a fixed-point equation;
  ② ***multivariate***, solution to a system of equilibrium conditions.

- roadmap of *rootfinding methods* (not covered in this course)
  - univariate case $\rightarrow$ BISECTION, shrink the interval until finding the *root* (`fzero`);
  - for the multivariate case:
    - NEWTON'S METHOD, iterate $n > 1$ times to find the zero of the Taylor-approximated system until $|\, f(\mathbf{x}_n) < \epsilon \,|$ (command `fsolve`);
    - FUNCTION ITERATION, transform rootfinding into a *fixed-point* problem, $\mathbf{x} = \mathbf{x} + f(\mathbf{x})$
  - COMPLEMENTARITY PROBLEM, when $max \;\; f(x) \;\; s.t. \;\; \{x > a\,;\, x < b\}$

☆ *important*: the results of optimization problem are rootfinding problem!

Outline

Table of Contents

Introduction

- in macroeconomics, a quintessential model is just a set of nonlinear equations

$$E_t \left[f\left(x_t, x_{t+1}, u_t, u_{t+1}\right) \right] = 0, \forall t$$

with $x_t = \begin{pmatrix} x_t^c \\ x_t^s \end{pmatrix}$ and

  - x_t^c: $m \times 1$ vector of endogenous (*predetermined*) control variables;
  - x_t^s: $n \times 1$ vector of endogenous (*forward*, *jump*) state variables;
  - u_t: $k \times 1$ vector of exogenous state variables.

- ***Goal***: find the path $\{x_t\}_{t=0}^{\infty}$ satisfying model equations given initial conditions $x^s_{t=0-1}$ and exogenous shocks $\{u_t\}_{t=0}^{\infty}$

Recursive methods

- main curse of modern quantitative macroeconomics models: *high dimensionality*
  - many variables over many periods and many *histories* of exogenous variables;
  - cannot solve infinitely many unknowns … use *recursion* idea.
- ***Recursive methods*** $\longleftrightarrow$ few variables are sufficient to summarize past history

$$x_t = g\left(x^s_{t-1}, u_t\right)$$

  - solving a model means finding $g(\cdot)$, called the *policy function*
- high dimensionality of $g(\cdot)$ requires some form of *approximation*
  - ***Local methods***: approximate equilibrium objects locally around some focus point
    - ⋆ *linearization*, *LQ methods*, ***perturbation*** *methods*
  - ***Global methods***: approximate global properties of equilibrium objects
    - ⋆ *discretization* (grids), ***projection*** *methods* (splines, integral approximation (Chebyshev, …))

What is *perturbation*?

- *key idea*
  - rewrite the model in terms of a *perturbation parameter*, $\sigma \geq 0$

$$E_t \left[f\left(x_t, x_{t+1}, u_t, u_{t+1}\,;\, \sigma\right) \right] = 0, \forall t$$

  - Taylor expansion in terms of the variables and the perturbation parameter
  - solve the approximated model using a dedicated approach

- thus, the model solution is as a function of σ

$$x_t = g\left(x^s_{t-1}, u_t\,;\, \sigma\right)$$

- usually, the perturbation parameter arises from

$$u_{t+1} = \mathcal{B}\left(u_t\right) + \sigma \underset{k\times k}{\mho}\, e_t$$

where $k \times k$ is the dimension of the matrix if (x_t, u_t) are vectors of variables

Steps

- *Step 0*: find the system of equations characterizing the model, *i.e.*,

$$E_t\left[f\left(x_t, x_{t+1}, u_t, u_{t+1}\right)\right] = 0,\ \forall t$$

- *Step 1*: solve the model with no uncertainty[1] $(\sigma = 0)$, *i.e.*, the steady-state system

$$f\left(x, x, u, u\right) = 0$$

- *Step 2*: find an approximation of $g(\cdot)$ around the above steady state
  - *i.e.*, approximate the policy functions

$$g\left(x^s_{t-1}, u_t\,;\,\sigma\right)$$

    around

$$g\left(x^s, u\,;\,\sigma\right)$$

  - how? j^{th}-order Taylor series expansion, or *pruning*[2]

- *Step 3*: resolution $\left\{\text{Blanchard and Kahn (1980), Uhlig (1999), Sims (2002), \dots}\right\}$

[1] This can typically be done with *rootfinding* (*e.g.*, Newton method). Be aware of multiple steady states (select only the best).

[2] Used if higher-order perturbations are worse than linear, *i.e.*, when $j > 1$ leads to different shape than true policy function.

Table of Contents

What is `dynare`?

- `dynare` is a collection of `MatLab` functions
  - to solve and simulate nonlinear models with forward-looking variables under rational expectations
- freely available on `www.dynare.org`
  - online you can also find reference manuals, some tutorials and an active forum
- what can `dynare` do?
  - solve deterministic models nonlinearly;
  - solve stochastic models (under local approximations);
  - solve for policy functions;
  - estimate DSGE models: maximum likelihood, Bayesian methods.
- *note*: `dynare` can solve only *equality* constraints
  - with *inequality* constraints, *rootfinding* method for *complementarity problems* (not covered)

How does `dynare` solve for the *policy rules*?

- `dynare` uses perturbation methods
  - there are other numerical methods, such as *value* or *policy function iterations*
- how does *perturbation* work?
  - start from a steady state;
  - take a Taylor expansion of the policy rules around the steady state;
  - find the coefficients of the Taylor expansion sequentially;
    - *e.g.*, for a second-order expansion, solve for the linear terms first and the quadratic then.
- *caveats*:
  - perturbation requires differentiability;
  - it's an approximation around a steady state.

`dynare`: Basics

- main unit in `dynare` is the `<name>.mod` file. In `MatLab`:
  - execute `dynare` with the command `dynare <name>.mod`
  - modify `dynare` with the command `edit <name>.mod`

→ it is the file in which you write down your stochastic model

- the `<name>.mod` file consists of different blocks
  - *variables and parameters* block
  - *model* block
  - *steady state* block
  - *shocks* block
  - *solution* (or *estimation*) block

Preliminary blocks

Variables and Parameters block

- `var` *list endogenous variable names*;
- `varexo` *list exogenous variables (shocks)*;
- `parameters` *list parameter names*;
  - then, *list parameter values*.

- parameter values can also be loaded from an external file, using the syntax `load <filename;>.mat` and `set_param_value ('parametername', parametername)`

Model block

- starts with `model;`
- ends with `end;`
- in between, type equations ending with ";"
  - `x(-1)` for predetermined variables
  - `x(+1)` for expectations
  - you can type an equation over several lines if you do not end it with ";"

- if the model is (*log*-)linear, type `model(linear);`

Steady state block

- `dynare` linearizes around the deterministic steady state
  - this steady state needs to be calculated
- two options:
  - let `dynare` calculate the steady state numerically ...
  - ... or calculate the steady state with paper and pencil and tell `dynare` what it is.
- calculating the steady state is a nonlinear problem
  - it might be difficult for the computer

Steady state block

- *Option 1*: numeric solution

Steady State block, option 1

- start with `initval;`
- ends with `end;`
- in between, add initial values (even previously defined) for all variables.

  - if a variable does not appear, `dynare` assumes it is zero in steady state;
  - if the model is elaborate, take initial values from a simpler model[3].

- *Option 2*: paper and pencil for analytical solutions

Steady State block, option 2

- start with `steady_state_model;`
- ends with `end;`
- in between, add equations where variables are function of parameters only.

  - or separate `<name>_steadystate.m` file with the same name as the `<name>.mod` file
    - inside, provide `dynare` with the steady state values (based on parameters)

[3] This concept is closely related to a more elaborate way on finding the steady state called *homotopy*.

Simulation blocks

Shocks block

- starts with `shocks;`
- ends with `end;`
- in between, declare shock standard deviations (as numerical values or by referring to a parameter name)

`var` *shock name* ; `stderr` *standard deviation value* ;

- note how the shock's value is expressed as being a *standard deviation* of the given innovation in that particular variable

Solution (or Estimation) block

- at the bottom of the `.mod` file add

```
stoch_simul(order=1,IRF=20);
```

- inside the brackets, many more options can be specified (mostly related to which output you want `dynare` to report);
- after the brackets you can list individual variables, if you want output for these variables only;
- otherwise, `dynare` will do it for all variables;
- `stoch_simul()` holds only for stochastic models. See below for deterministic models.

Time convention

pay attention to the timing of the variables!

- in `dynare`, the timing of each variable reflects when that variable is decided
  - *example*: in most models, capital stock is subject to a law of motion. This implies that such capital is not decided at t, but rather at $t-1$
    - in jargon, it is a *predetermined* variable;
    - that is why in `dynare`, the capital stock in the production function is written at $t-1$.
- time indices are given in parenthesis: $x_{t+1} \rightarrow$ `x(+1)`, $x_t \rightarrow$ `x`, $x_{t-1} \rightarrow$ `x(-1)`

⚠ why is it important to struggle with time convention?

  - you should know that Blanchard and Khan (1980) conditions are met only if *the number of non-predetermined variables equals the number of eigenvalues greater than one* ...
    - ... and if this condition is not met, you will get a warning!

- note that if a variable is predetemined, it is not true that this should always be written at $t-1$ throughout the code
  1. consumption may be forward-looking in the Euler, but may also have lag(s) if habit formation is featured in the model;
  2. in a resource constraint, capital on the left-hand-side is at t, but that on the right-hand-side is at $t-1$ since capital belongs both from the law of motion and the production function.

Running the `.mod` file

- in `MatLab`, make sure you are in the right directory, then just type:

  `dynare <name>.mod`

  to run the simulation, or type `edit <name>.mod` to modify it

- depending on the settings inside the `stoch_simul` command, `dynare` generates:
  - policy rules;
  - implied moments (means, standard deviations, correlations);
  - impulse response functions, ...

- `dynare` writes these results to the screen, but also stores them in additional files
  - `oo_.dr.ys` is a $(m+n) \times 1$ vector of steady state values;
  - `oo_.dr.ghx` is a $(m+n) \times n$ matrix of x_t coefficients;
  - `oo_.dr.ghu` is a $(m+n) \times k$ matrix of u_t coefficients;
  - `oo_.irfs` reports IRFs for each variable.

- note:
  - `dynare` reports policy functions in deviations from steady state values;
  - the constant gives the steady state value of a given variable (column).

Other resources

- on the `dynare` website `https://www.dynare.org/resources/` you can get ...
  - forum;
  - quick start/tutorial;
  - manual;
  - or `dynare` implementations of published models at:
    - `https://www.macromodelbase.com/`, directly in its `download` section;
    - `https://github.com/johannespfeifer/dsge_mod`;
    - `https://forum.dynare.org/t/practicing-dynare-2019-updated-version/13389`.

  ... hundreds of replication codes available, learn by using it!

☞ ***concluding remarks***. Simulating in `dynare` is:

- easy and flexible, but be careful with its idiosyncratic behaviour!
- great for prototyping (*local* solutions), but may need more accurate (*global*) solutions.